

FISE 2A ROB - UE 4.2 - Modélisation Robots Mobiles

Introduction

Modélisation des robots

Différents types et niveaux de modélisation , exemples :

- ▶ Modélisation cinématique simple
- ▶ Modélisation dynamique
- ▶ Modélisation hybride : HIL (Hardware In the Loop)

Modélisation des robots

Pourquoi modéliser les robots ?

La modélisation dynamique des robots peut être utile :

- ▶ en début de projet, pour rapidement évaluer et comparer différents concepts sans recourir à une réalisation physique
- ▶ en cours de projet, pour donner accès au robot à une équipe de développement si nombre limité de robots physiques, ou si les robots physiques ne sont pas prêts.
- ▶ en cours de projet, pour tester les algorithmes en environnement contrôlé

Pour être utile, la modélisation doit :

- ▶ approcher au plus près le comportement dynamique du robot
- ▶ restituer de la façon la plus réaliste possible les informations acquises par les capteurs

Le modèle dynamique utilise 3 composantes :

- ▶ Rendu visuel (esthétique) : importé (Solidworks, Blender, Autodesk, ...)
- ▶ Collision : simplification du rendu visuel afin de réduire le temps de calcul des collisions.
- ▶ Dynamique : formes génériques (primitives) simples (cuboïde, sphère, cylindre, ...) et articulations (liaisons) reliant ces formes simples.

Les composantes "Collision" et "Dynamique" sont en général rendues invisibles lors de la simulation.

Déroulement du cours :

- ▶ Série d'exercices sur les éléments nécessaires à la création d'un robot (Modèle dynamique, interface de contrôle/commande, simulation capteurs, interface ROS, ...)
- ▶ TD noté de 4h : modélisation d'un robot simple
- ▶ Projet noté (trois séances de 4h) : simulation du DDBOAT (travail en groupes, résultats sur GitHub)

Modélisation des robots

Logiciels de simulation

Il est possible de réaliser une simulation simplifiée à l'aide de programmes Python ou C++ ...

... mais généralement, on s'appuie sur des solveurs existants pour la résolution des équations dynamiques :

- ▶ open source : Bullet, Newton, ODE, ...
- ▶ avec licence : Vortex

Plusieurs logiciels intègrent ces solveurs. Il suffit "simplement" de définir le modèle dynamique du robot. Ces principaux logiciels sont :

- ▶ Gazebo : open source , généralement connecté à ROS
- ▶ V-REP : gratuit pour l'enseignement, assez simple à utiliser (attention Lua !)
- ▶ Webots : logiciel avec licence, devenu open source en 2019 (Version R2019a)

Modélisation d'un premier véhicule très simple ...

- ▶ V-REP sera utilisé pour ce cours (EN 3A nous utiliserons Gazebo)
- ▶ Récupérer V-REP du projet DART-V2 ou télécharger V-REP PRO EDU 3.6.2 ici :
<https://www.coppeliarobotics.com/previousVersions>
- ▶ La prise en main de V-REP est réalisée sous forme d'un exercice de création d'un premier robot très simple de type tricycle
- ▶ La roue avant est motrice et directrice, les roues arrières sont passives.
- ▶ A terme, les pédales et le guidon pourraient être actionnées par un humanoïde Nao

Modélisation des robots

Exercice 1 : création d'un robot simple tricycle

Ce premier exercice consiste à créer un robot tricycle à partir d'un fichier de CAO récupéré sur la bibliothèque GrabCad library :

<https://grabcad.com/library/tricycle-23>



Modélisation des robots

Exercice 1 : création d'un robot simple tricycle

Démarche :

- ▶ Utilisation de fichiers 3D avec les différentes pièces
- ▶ Représentation simplifiée de la dynamique
- ▶ Définition des articulations
- ▶ Construction : liens entre les éléments dynamiques et les articulations
- ▶ Définition des formes de collision éventuelles
- ▶ Définition du rendu esthétique

Modélisation des robots

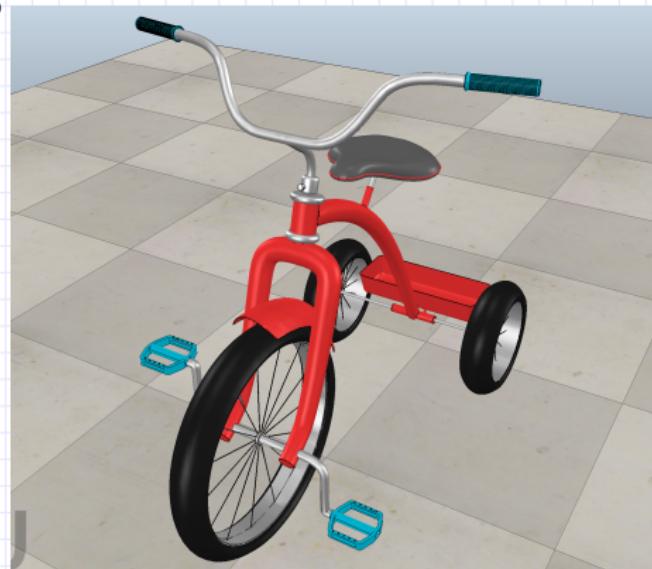
Exercice 1 : passage CAO à V-REP

- ▶ Importation des maillages 3D (meshes) dans V-REP
- ▶ Séparation et renommage des pièces
- ▶ Amélioration du rendu
- ▶ Sauvegarde au format V-REP

Partie fastidieuse !!!

Utilisation de logiciels externes :
Blender, FreeCad, ...

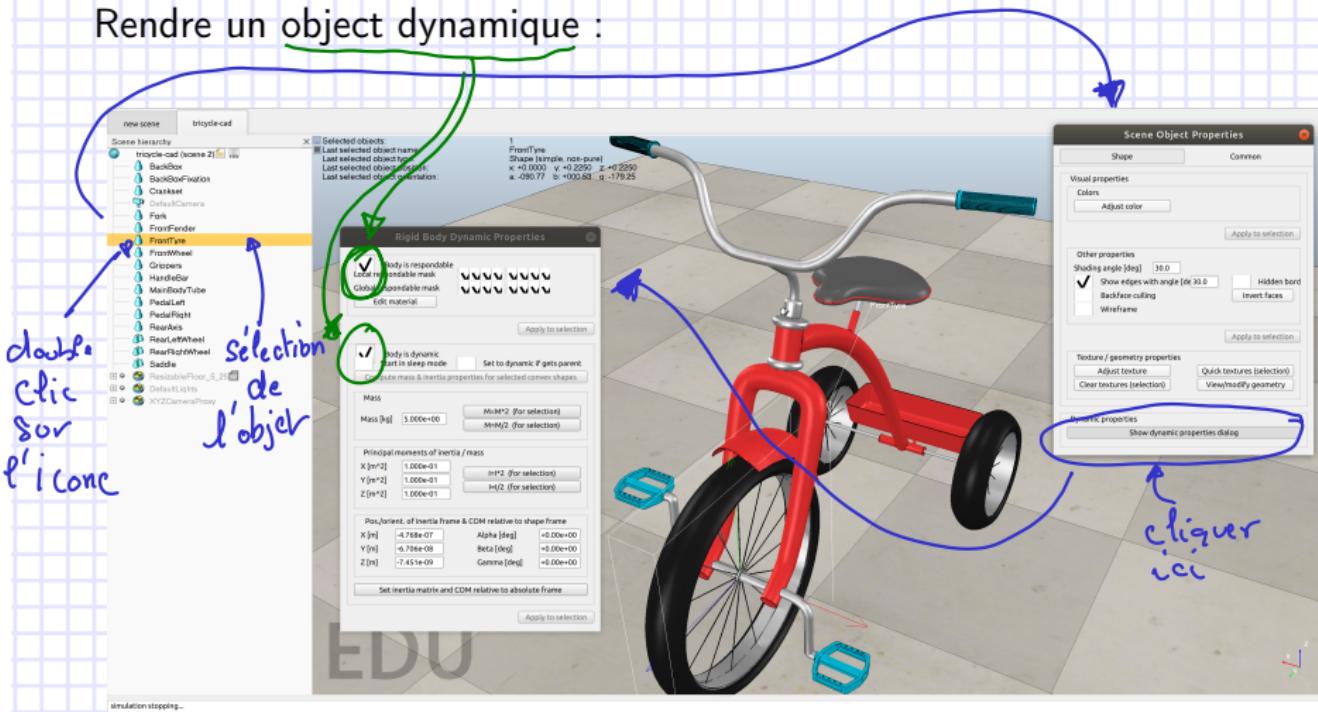
Résultat : tricycle-cad.ttt



Modélisation des robots

Exercice 1 : modèle dynamique

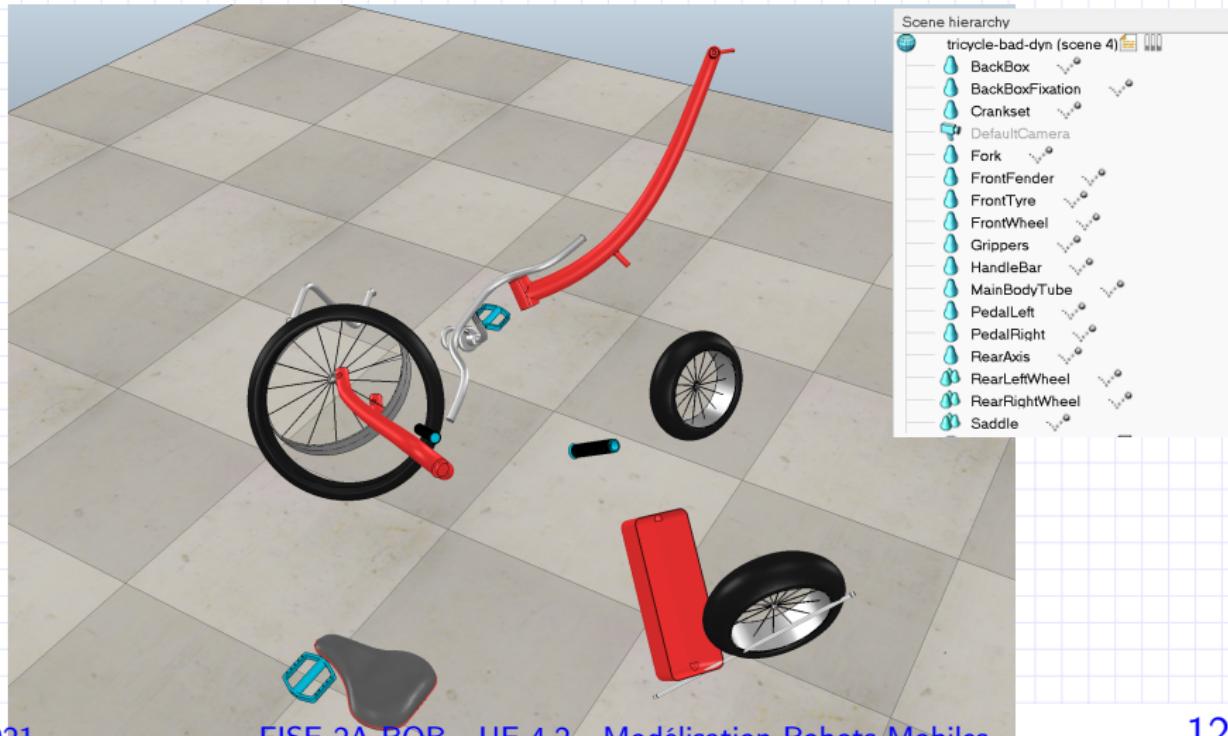
Rendre un object dynamique :



Modélisation des robots

Exercice 1 : modèle dynamique

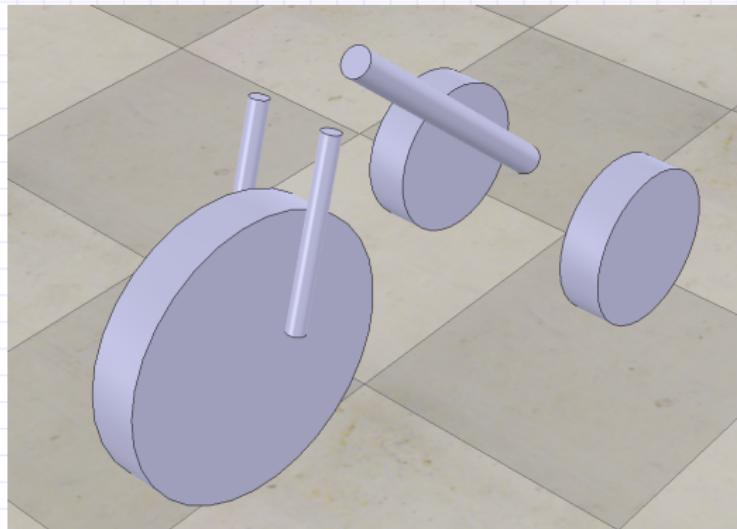
Idée (trop) simple et mauvaise : rendre tous les objets dynamiques !



Modélisation des robots

Exercice 1 : modèle dynamique - primitives

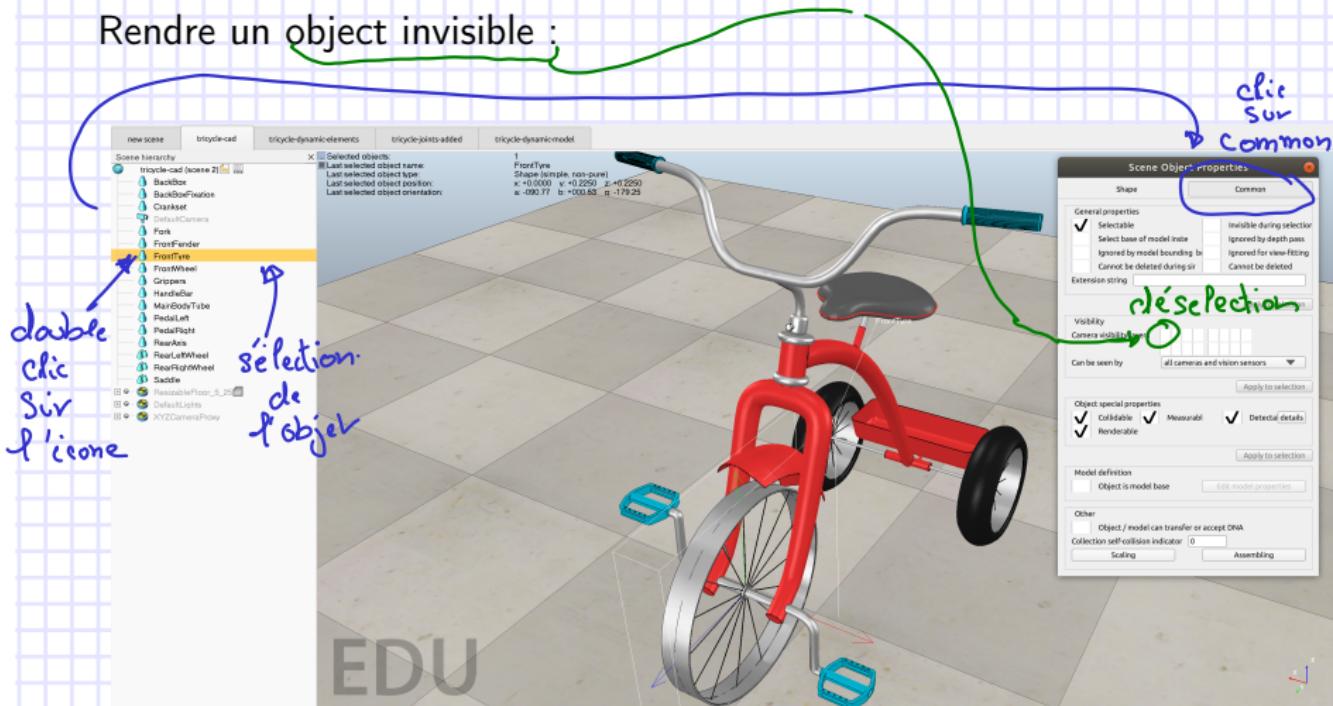
- ▶ Solution : créer un modèle dynamique à partir de formes simples
- ▶ Utiliser uniquement des formes primitives et laisser la masse par défaut
- ▶ Menu V-REP : Add -> Primitive Shape



Modélisation des robots

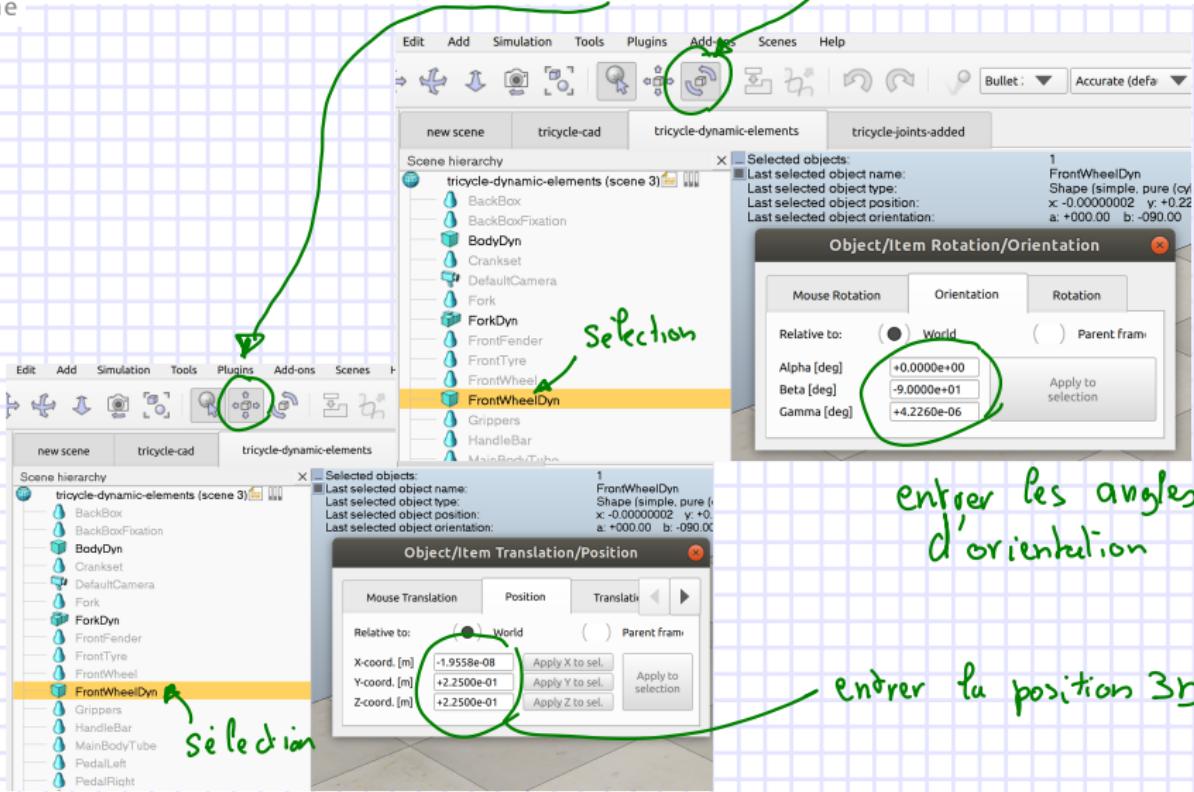
Exercice 1 : gestion de la visibilité

Rendre un object invisible :



Modélisation des robots

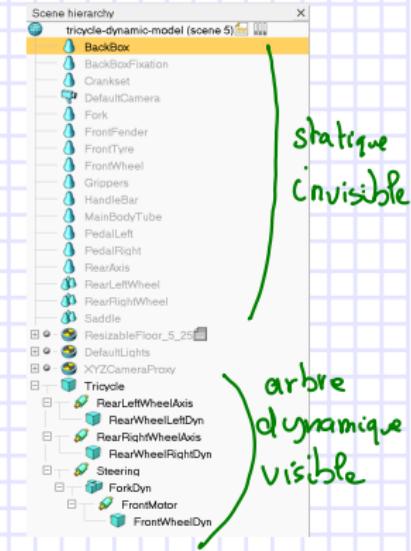
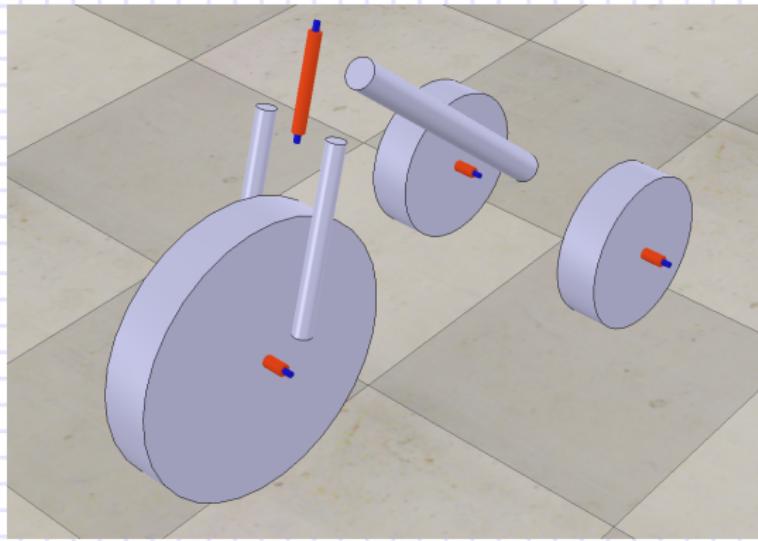
Exercice 1 : modèle dynamique - déplacer et orienter les objets



Modélisation des robots

Exercice 1 : modèle dynamique - articulations

- ▶ Articulations : relient les primitives entre elles selon une arborescence
- ▶ Création des articulations : Menu V-REP : Add -> Joints
- ▶ Définition de l'arborescence, en partant du corps du robot



Modélisation des robots

Exercice 1 : modèle dynamique - articulations

Ajout des articulations dans V-REP :

V-REP : Utilisation de add – > joints pour saisir les articulations

Trois types d'articulations sont disponibles :

- ▶ "revolute" : rotation infinie ou "cyclic" (moteurs) ou limitée angulairement (servos)
- ▶ "prismatic" : translation (vérins)
- ▶ "spherical" : combinaison des trois rotations (passif)

Voir description sur :

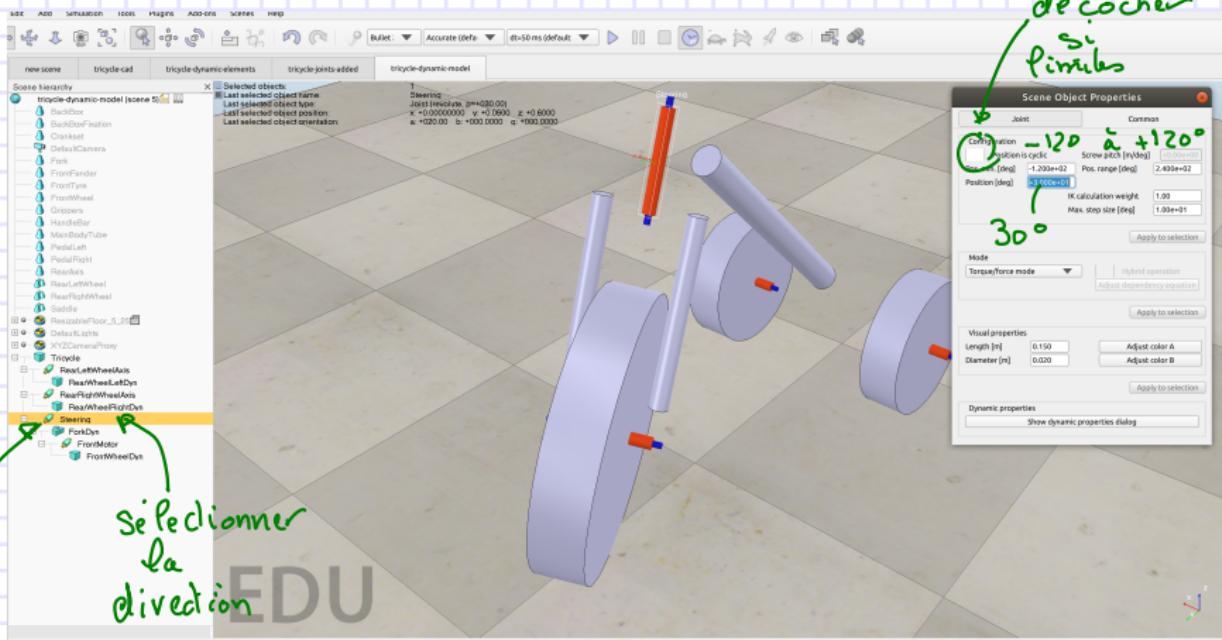
<https://www.coppeliarobotics.com/helpFiles/en/jointDescription.htm>

Pour modifier les paramètres d'une articulation, double-cliquer sur l'articulation dans la fenêtre "scene hierarchy"

Modélisation des robots

Exercice 1 : premiers tests

Changer la direction :

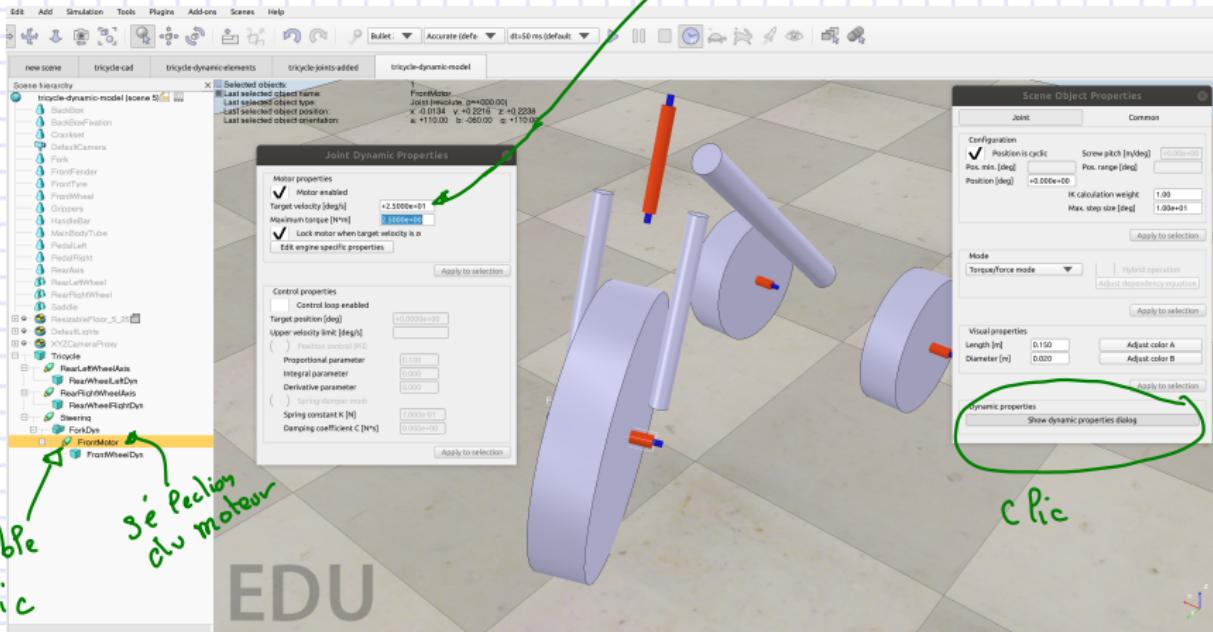


Modélisation des robots

Exercice 1 : premiers tests

Activer le moteur :

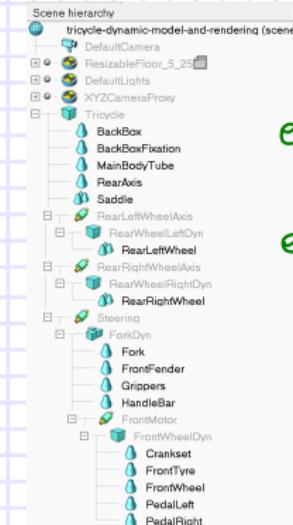
vitesse de rotation en %/s



Modélisation des robots

Exercice 1 : finalisation

Mettre à jour l'arborescence en ajoutant les éléments esthétiques statiques, rendre invisible le modèle dynamique et afficher le modèle esthétique :

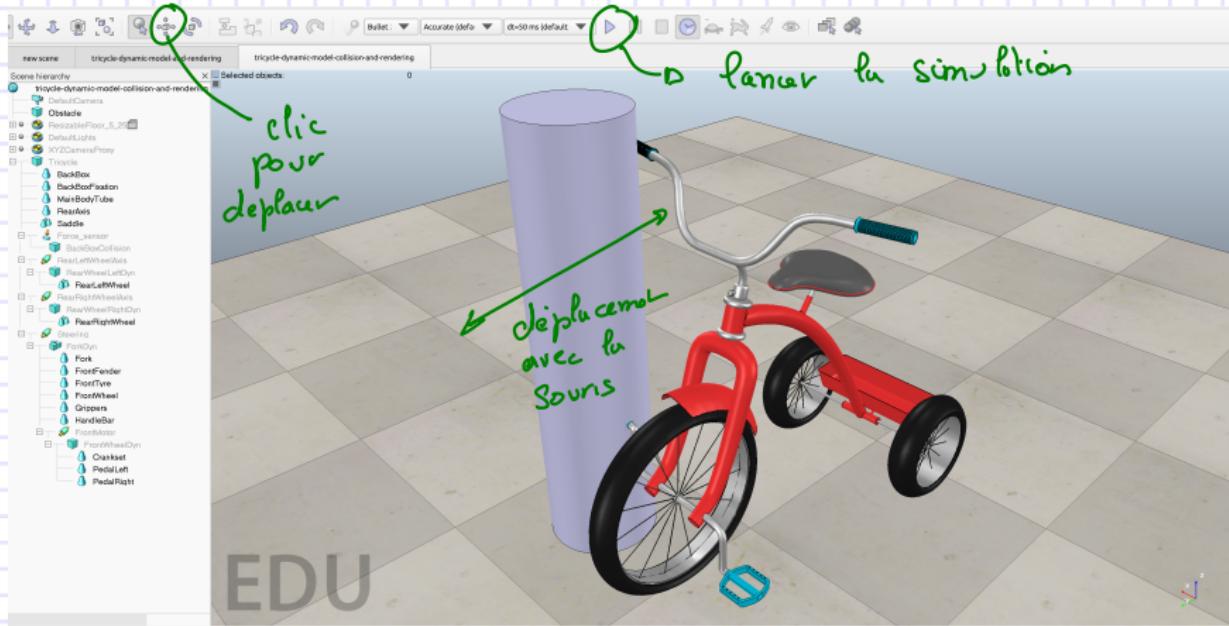


en noir
visible
en grisé
invisible

Modélisation des robots

Exercice 1 : gestion des collisions

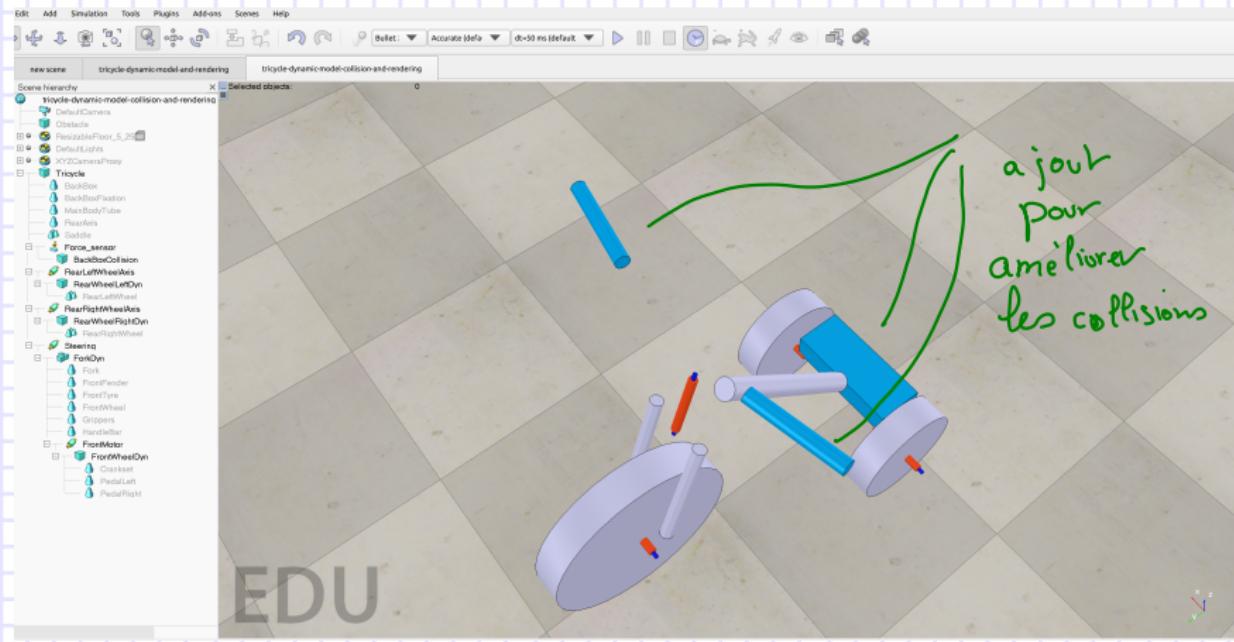
Ajouter un cylindre vertical (primitive) dans la scène et tester les collisions en le déplaçant manuellement :



Modélisation des robots

Exercice 1 : gestion des collisions

Modifier le modèle dynamique en ajoutant des éléments de collision (ex pour le guidon et le pannier arrière) :



Modélisation des robots

Exercice 2 : création d'un robot 4x4 à articulation centrale

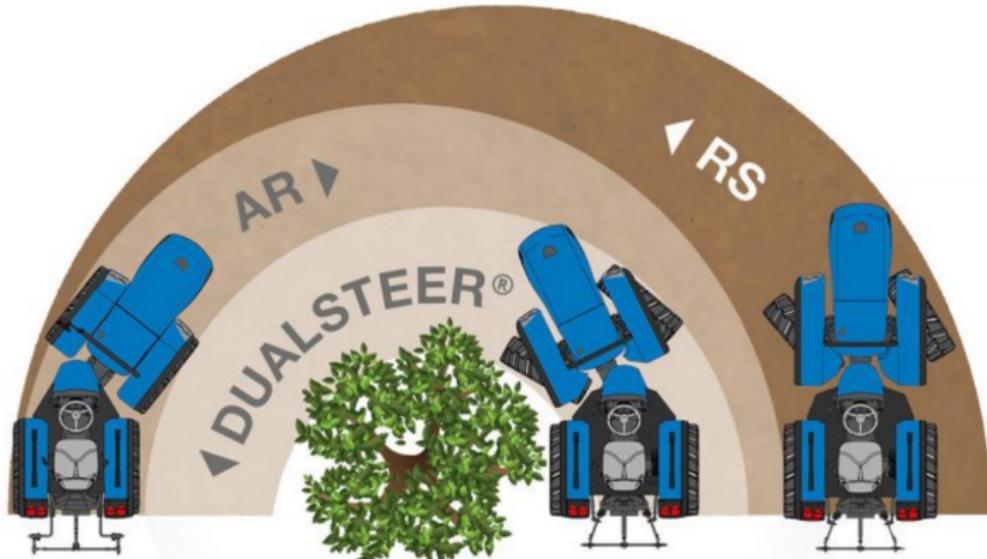
Démarche :

- ▶ Conception du robot : papier, CAO, Blender , ...
- ▶ Représentation simplifiée de la dynamique
- ▶ Définition des articulations
- ▶ Construction : liens entre les éléments dynamiques et les articulations
- ▶ Ajouter un terrain
- ▶ Script (Lua) de test de fonctionnement du robot
- ▶ Définition des formes de collision
- ▶ Définition du rendu esthétique

Modélisation des robots

Exercice 2 : création d'un robot 4x4 à articulation centrale

Le second exercice consiste à créer "from scratch" un robot 4x4 dont le contrôle de la direction est assuré par une articulation centrale actionnée par un ou deux vérins (mode AR sur la figure).



Modélisation des robots

Exercice 2 : création d'un robot 4x4 à articulation centrale

Faire un dessin simple du robot sur papier ou avec un logiciel (Blender, Freecad, Inventor, ...) en le décomposant en éléments de forme géométrique simple. Par exemple :

- ▶ roues : cylindres
- ▶ corps avant et arrière : parallélépipèdes rectangles (cuboïdes)

Ces formes sont ensuite éventuellement regroupées pour créer des éléments plus complexes.

Lorsque toutes les éléments sont définis, ils sont connectés entre eux par des articulations (liaisons) en partant d'un élément de base.

En général, l'élément de base est l'élément principal (ou central) du châssis du robot. Avec l'articulation centrale, il y a deux possibilités, il faudra choisir entre le demi-châssis avant ou arrière

Modélisation des robots

Exercice 2 : création d'un robot 4x4 à articulation centrale

Saisir les éléments du modèle dynamique du robot dans V-REP

- ▶ V-REP : Utilisation de add – > primitive shapes pour saisir les formes simples

- ▶ Pour changer la position d'un objet, utiliser le bouton : 
- ▶ Pour changer l'orientation d'un objet, utiliser le bouton : 
- ▶ Note : garder pour l'instant la masse par défaut

Modélisation des robots

Exercice 2 : création d'un robot 4x4 à articulation centrale

Ajout des articulations dans V-REP :

V-REP : Utilisation de add – > joints pour saisir les articulations

Trois types d'articulations sont disponibles :

- ▶ "revolute" : rotation infinie ou "cyclic" (moteurs) ou limitée angulairement (servos)
- ▶ "prismatic" : translation (vérins)
- ▶ "spherical" : combinaison des trois rotations (passif)

Voir description sur :

<https://www.coppeliarobotics.com/helpFiles/en/jointDescription.htm>

Pour modifier les paramètres d'une articulation, double-cliquer sur l'articulation dans la fenêtre "scene hierarchy"

Modélisation des robots

Exercice 2 : création d'un robot 4x4 à articulation centrale

Création de l'arborescence du modèle dynamique du robot en partant de l'élément de base et en reliant successivement les autres éléments à l'aide des articulations.

Exemple : Corps avant relié à la roue avant droite par une articulation en rotation infinie.

En général, une articulation relie deux formes dynamiques.

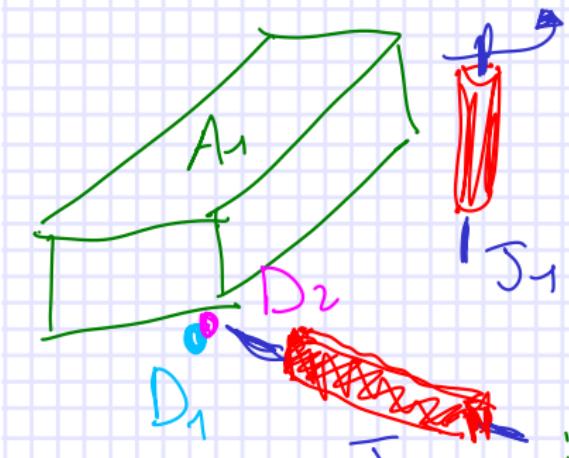
Si les formes doivent être reliées par deux articulations ou plus, on utilise des objets "dummy". Voir détails dans la section **Dynamically enabled joints/force sensors** du lien :

<https://www.coppeliarobotics.com/helpFiles/en/designingDynamicSimulations.htm>

Modélisation des robots

Exercice : création d'un robot 4x4 à articulation centrale

Utilisation des objets "Dummy"

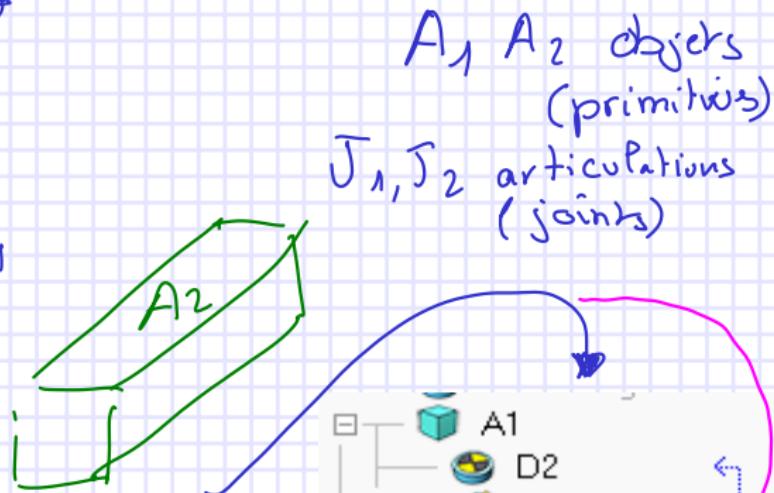


A_1

$\hookrightarrow J_1$

$\hookrightarrow A_2$

J_2 ? "cijour de
dummies"
 D_1 D_2



A_1, A_2 objets
(primitives)

J_1, J_2 articulations
(joints)

Modélisation des robots

Exercice : création d'un robot 4x4 à articulation centrale

Améliorer le "rendering" (aspect visuel) du robot :

Exemple : ajouter de "vraies" roues et un châssis "lego"

- ▶ rechercher une roue sur Internet
- ▶ convertir au format COLLADA (.dae)
- ▶ importer dans V-REP (File->Import->Mesh ...)
- ▶ ajouter cette roue à la roue dynamique (cylindre) déjà existant

Note : Si vos recherches sur Internet et conversions sont trop fastidieuses, vous pouvez récupérer une jante, un pneu et une méga brique LEGO sur MOODLE.

Modélisation des robots

Exercice : création d'un robot 4x4 à articulation centrale

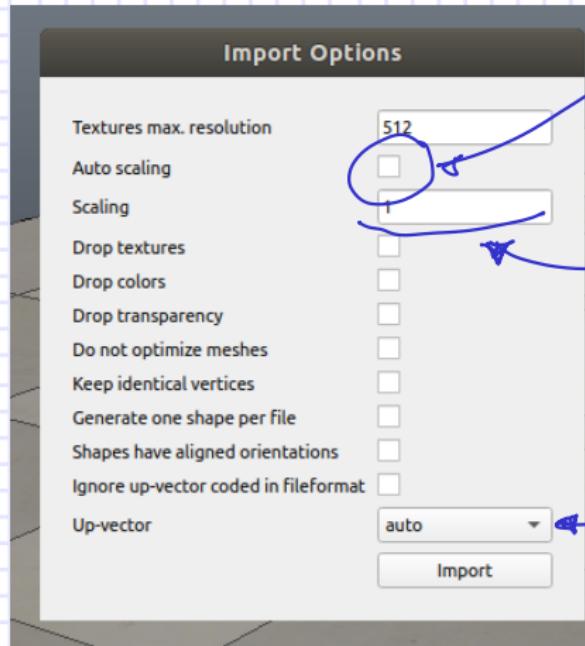
Import d'objets 3D

Si besoin :

enlever
"Auto Scaling"

définir l'échelle

changer l'axe Z



Modélisation des robots

Exercice 2 : création d'un robot 4x4 à articulation centrale

Ajout du terrain :

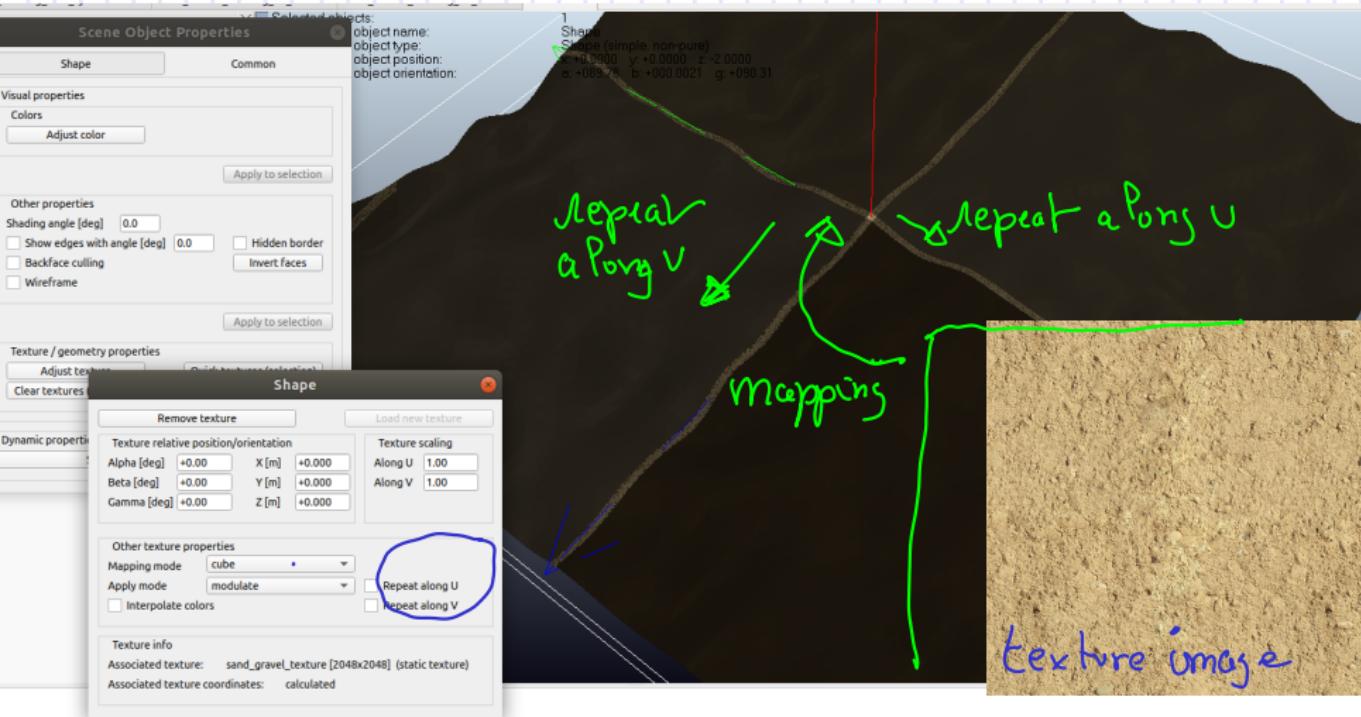
- ▶ Recherche sur internet et conversion
- ▶ Importer dans V-REP (File->Import->Mesh ...) ou avec le plugin **COLLADA Import/export** ...
- ▶ Mettre le terrain à la bonne échelle et à la bonne position
- ▶ Rendre le terrain "collisionnable" afin que le robot ne passe pas à travers!!! (terrain "respondable" en V-REP)
- ▶ Ajouter une texture de roches, de gravier ou de sable à partir d'une images

Note : Si vos recherches sur Internet et conversions sont trop fastidieuses, vous pouvez récupérer un terrain 3D et une image de texture sous MOODLE.

Modélisation des robots

Exercice : création d'un robot 4x4 à articulation centrale

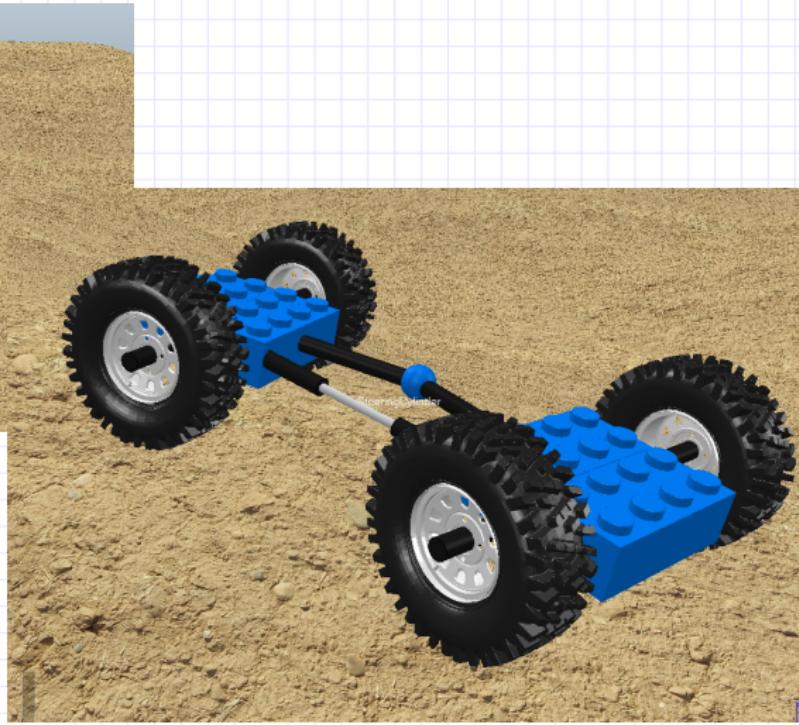
Textures



Modélisation des robots

Exercice : création d'un robot 4x4 à articulation centrale

Exemple de résultat



Modélisation des robots

Exercice 3 : Première mission robotique en simulation

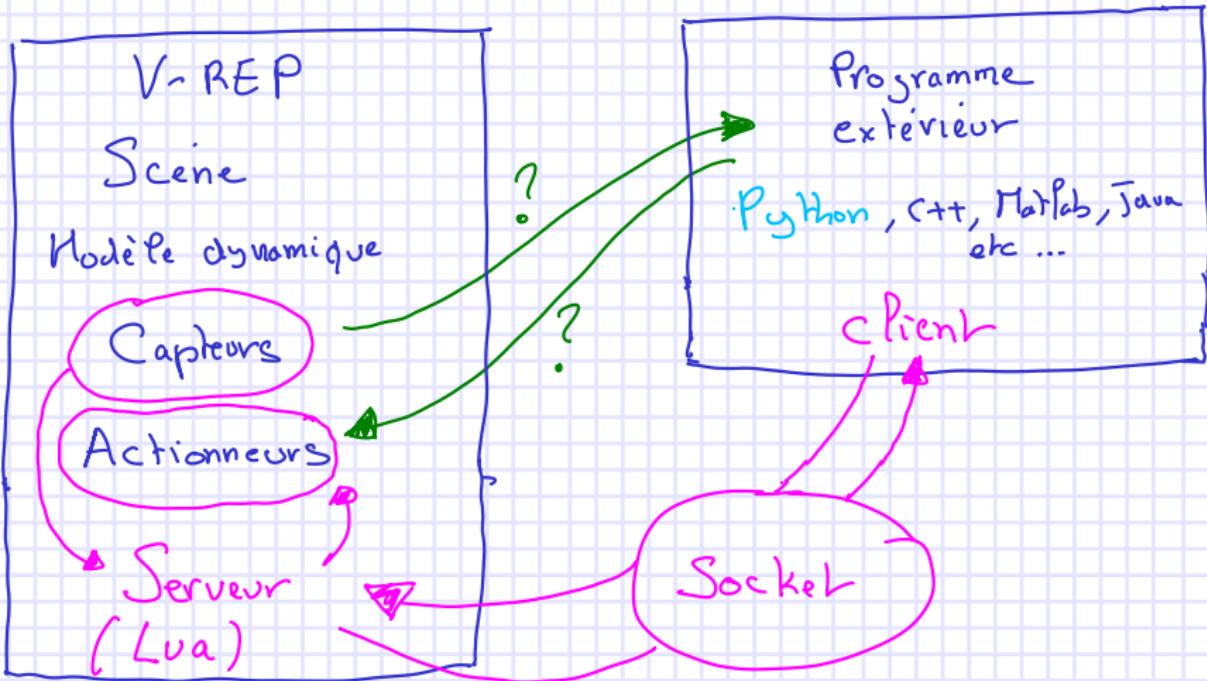
Réaliser une mission robotique simulée à l'aide de V-REP :

- ▶ Approche "fermée" : lorsque le robot est créé, il peut être testé, programmé et contrôlé entièrement dans V-REP en codant en Lua.
- ▶ Approches "ouvertes" : utilisation d'un programme extérieur pour commander le robot :
 - ▶ Utilisation de la "Remote API" de V-REP :
<https://www.coppeliarobotics.com/helpFiles/en/legacyRemoteApiOverview.htm>
 - ▶ Utilisation d'un socket de communication entre V-REP et un programme extérieur.
 - ▶ Utilisation de topics ROS entre V-REP et un programme extérieur ROS.

Modélisation des robots

Exercice 3 : Première mission robotique en simulation

Socket de communication



Modélisation des robots

Exercice 3 : Première mission robotique en simulation

Réalisation des premiers tests directement dans V-REP :

- ▶ Utilisation du code Lua.
- ▶ Choisir une forme du robot et ajouter un script (clic droit) :
Add –> Associated child script –> Threaded
- ▶ Ouvrir le script en double-cliquant sur l'icône (fichier texte en bleu) et copier le contenu dans un fichier extérieur "simple_control.lua"
- ▶ Enlever le contenu et simplement écrire :
`require ("simple_control")`
puis sauvegarder la scène V-REP
- ▶ Ouvrir avec votre éditeur le programme **simple_control.lua**

Modélisation des robots

Exercice 2 : création d'un robot 4x4 à articulation centrale

La fonction à modifier est **sysCall_threadmain()**

Listing 1 – Lua Control Script

```
1 function sysCall_threadmain()
2     — Put some initialization code here
3
4
5     — Put your main loop here, e.g.:
6     —
7     — while sim.getSimulationState()~=sim.simulation_advancing_aboutostop do
8     —     local p=sim.getObjectPosition(objHandle,-1)
9     —     p[1]=p[1]+0.001
10    —     sim.setObjectPosition(objHandle,-1,p)
11    —     sim.switchThread() — resume in next simulation step
12    — end
13 end
```

La documentation complète se trouve au lien suivant :

<https://www.coppeliarobotics.com/helpFiles/index.html>

Modélisation des robots

Exercice 2 : création d'un robot 4x4 à articulation centrale

Déplacer le robot en contrôlant ses articulations

Quelques fonctions utiles :

- ▶ `h = sim.getObjectHandle("NomObjet")` – récupère l'identifiant d'un l'objet
- ▶ `h = sim.setJointTargetVelocity(h,spd)` – définit la vitesse de consigne d'une articulation
- ▶ `p = sim.getJointPosition(h)` – récupère la position d'une articulation
- ▶ `printToConsole(p)` – affiche une valeur sur le terminal
- ▶ ...

Note : il faudra peut-être modifier la force des moteurs

Modélisation des robots

Exercice 3 : Première mission robotique en simulation

Exemple pour "simple_control.lua"

```
require ("math")
function sysCall_threadmain()
    -- Put some initialization code here
    steer_actuator = sim.getObjectHandle("CentralAxis")
    wheel_front_left = sim.getObjectHandle("MotorFrontLeft")
    wheel_front_right = sim.getObjectHandle("MotorFrontRight")
    wheel_rear_left = sim.getObjectHandle("MotorRearLeft")
    wheel_rear_right = sim.getObjectHandle("MotorRearRight")
    speed_actuator = 0.2 -- rad/s
    -- Put your main loop here, e.g.:
    spd = 2.0
    while sim.getSimulationState()~=sim.simulation_advancing_aboutostop do
        sim.setJointTargetVelocity(wheel_front_left,spd)
        sim.setJointTargetVelocity(wheel_front_right,spd)
        sim.setJointTargetVelocity(wheel_rear_left,spd)
        sim.setJointTargetVelocity(wheel_rear_right,spd)
        sim.setJointTargetVelocity(steer_actuator,speed_actuator)
        steer_actuator_val = sim.getJointPosition(steer_actuator)*180.0/math.pi
        printToConsole("central axis angle:",steer_actuator_val) -- print on terminal
        print("central axis angle:",steer_actuator_val) -- print in V-REP GUI
        -- change direction when reaching limits
        if steer_actuator_val > 20 then
            speed_actuator = -0.2
        end
        if steer_actuator_val < -20 then
            speed_actuator = 0.2
        end
        sim.switchThread() -- resume in next simulation step
    end
end

function sysCall_cleanup()
    -- Put some clean-up code here
end
```

Méthode de contrôle bas niveau en Python3 utilisant les sockets.

Le dialogue entre votre programme de commande et le robot simulé se fait en mode client-serveur. Le serveur est le robot et votre programme Python3 est le client.

- ▶ S'inspirer de la scène V-REP
`central_steering_simplified_2021_socket_control.ttt` sur MOODLE
- ▶ Le serveur est écrit en Lua ; le fichier `socket_control.lua` est à mettre dans le même dossier que la scène.
- ▶ Les échanges se font sous forme de chaînes de caractères hexadécimaux. Mettre dans le dossier d'exécution de V-REP le fichier `pack.so` et redémarrer V-REP
- ▶ Il reste à écrire le client en Python3

Modélisation des robots

Exercice 3 : Contrôle du robot par socket

Le message à envoyer est constitué de 4 valeurs :

- ▶ caractère A : code synchro 1
- ▶ caractère Z : code synchro 2
- ▶ flottant : vitesse gauche
- ▶ flottant : vitesse droite

Le message reçu est aussi constitué de 6 valeurs :

- ▶ caractère A : synchro 1
- ▶ caractère Z : synchro 2
- ▶ flottant : temps depuis le début de la simulation
- ▶ flottant : angle roue gauche
- ▶ flottant : angle roue droite
- ▶ flottant : angle articulation centrale

Il faut le coder sous forme de chaîne hexadécimale :

- ▶ `struct.pack / struct.unpack` en Python
- ▶ `bpack / bunpack` en Lua

Modélisation des robots

Exercice : Contrôle du robot

Liens utiles :

Conversion des données du message en une chaîne de caractère hexadécimaux, bibliothèque lpack sur :
<http://webserver2.tecgraf.puc-rio.br/lhf/ftp/lua/>

Sockets en Python3 :

<https://docs.python.org/3/library/socket.html>

Fonction de l'API V-REP (en lua) :

<https://www.coppeliarobotics.com/helpFiles/en/apiOverview.htm>

Modélisation des robots

Exercice 3 : Contrôle du robot par ROS

Méthode de contrôle par échange de messages (topics) avec ROS

Le dialogue entre votre programme de commande et le robot simulé se fait à l'aide de "topics" ROS en modes "publisher" et "subscriber"

- ▶ S'inspirer de la scène V-REP central_steering_simplified_2021_ros_control.ttt sur MOODLE
- ▶ Coté V-REP l'interface ROS est écrit en lua ; le fichier ros_control.lua est à mettre dans le même dossier que la scène.
- ▶ Avant de démarrer V-REP, lancer un "roscore" dans un autre terminal et s'assurer que ROS est correctement initialisé (/opt/ros/melodic/setup.bash exécuté)
- ▶ Il faut modifier ros_control.lua pour l'adapter à votre robot et le rendre commandable par le "topic /cmd_vel"
- ▶ Vérification du bon fonctionnement avec "rostopic list" puis contrôle de la commande avec "rqt" et le "topic /cmd_vel"

Modélisation des robots

Exercice 3 : Première mission robotique en simulation

Exemple "ros_control.lua" : Crédit des "topics"

Attention !!! le script "ros_control.lua" est un "Non Threaded Script"

```
function sysCall_init()
    -- The child script initialization
    print ("init")
    objectName="CentralAxis"
    objectHandle=sim.getObjectHandle(objectName)
    --referenceName="ResizableFloor_5_25"
    referenceName="Cuboid"
    referenceHandle=sim.getObjectHandle(referenceName)
    -- get left and right motors handles
    leftFrontMotor = sim.getObjectHandle("MotorFrontLeft")
    rightFrontMotor = sim.getObjectHandle("MotorFrontRight")
    leftRearMotor = sim.getObjectHandle("MotorRearLeft")
    rightRearMotor = sim.getObjectHandle("MotorRearRight")
    centralAxis = sim.getObjectHandle("CentralAxis")
    rosInterfacePresent=simROS
    -- Prepare the publishers and subscribers :
    if rosInterfacePresent then
        publisher1=simROS.advertise('/simulationTime','std_msgs/Float32')
        publisher2=simROS.advertise('/pose','geometry_msgs/Pose')
        publisher3=simROS.advertise('/centralAxisAngle','std_msgs/Float32')
        subscriber1=simROS.subscribe('/cmd_vel','geometry_msgs/Twist','subscriber_cmd_vel_callback')
    end
end
```

Modélisation des robots

Exercice 3 : Première mission robotique en simulation

Exemple "ros_control.lua" : Publication des "topics"

```
function sysCall_actuation()
    -- Send an updated simulation time message, send the transform of the central axis
    -- and send the angle of the central axis
    if rosInterfacePresent then
        -- publish time, pose and angle topics
        simROS.publish(publisher1,{data=sim.getSimulationTime()})
        simROS.publish(publisher2,getPose("CentralAxis"))
        centralAxisAngle = sim.getJointPosition(centralAxis)*180.0/math.pi
        simROS.publish(publisher3,{data=centralAxisAngle})
        -- send a TF : robot w.r.t. floor
        simROS.sendTransform(getTransformStamped(objectHandle,objectName,referenceHandle,referenceName))
        -- To send several transforms at once, use simROS.sendTransforms instead
    end
end
```

Modélisation des robots

Exercice 3 : Première mission robotique en simulation

Exemple "ros_control.lua" : obtention de la pose et création d'une TF

```
function getPose(objectName)
    -- This function get the object pose at ROS format geometry_msgs/Pose
    objectHandle=sim.getObjectHandle(objectName)
    relTo = -1
    p=sim.getObjectPosition(objectHandle,relTo)
    o=sim.getObjectQuaternion(objectHandle,relTo)
    return {
        position={x=p[1],y=p[2],z=p[3]},
        orientation={x=o[1],y=o[2],z=o[3],w=o[4]}
    }
end

function getTransformStamped(objHandle,name,relTo,relToName)
    -- This function retrieves the stamped transform for a specific object
    t=sim.getSystemTime()
    p=sim.getObjectPosition(objHandle,relTo)
    o=sim.getObjectQuaternion(objHandle,relTo)
    return {
        header={
            stamp=t,
            frame_id=relToName
        },
        child_frame_id=name,
        transform={
            translation={x=p[1],y=p[2],z=p[3]},
            rotation={x=o[1],y=o[2],z=o[3],w=o[4]}
        }
    }
end
```

Modélisation des robots

Exercice 3 : Première mission robotique en simulation

Exemple "ros_control.lua" : Contrôle du robot par un "callback" généré par un "subscriber"

```
function subscriber_cmd_vel_callback(msg)
    -- This is the subscriber callback function when receiving /cmd_vel topic
    -- The msg is a Lua table defining linear and angular velocities
    -- linear velocity along x = msg["linear"]["x"]
    -- linear velocity along y = msg["linear"]["y"]
    -- linear velocity along z = msg["linear"]["z"]
    -- angular velocity along x = msg["angular"]["x"]
    -- angular velocity along y = msg["angular"]["y"]
    -- angular velocity along z = msg["angular"]["z"]
    spdLin = msg["linear"]["x"]*10.0
    spdAng = msg["angular"]["z"]*10.0
    kLin = -0.5
    kAng = -0.2
    spdLeft = kLin*spdLin+kAng*spdAng
    spdRight = kLin*spdLin-kAng*spdAng
    sim.setJointTargetVelocity(leftFrontMotor,spdLeft)
    sim.setJointTargetVelocity(rightFrontMotor,spdRight)
    sim.setJointTargetVelocity(leftRearMotor,spdLeft)
    sim.setJointTargetVelocity(rightRearMotor,spdRight)
    sim.addStatusBarMessage('cmd_vel subscriber receiver : spdLin = '..spdLin..',spdAng='..spdAng..' comm:
and : spdLeft"..spdLeft.." ,act"..spdRight)
end
```

Modélisation des robots

Exercice : Contrôle du robot

Liens utiles :

Documentation V-REP :

<https://www.coppeliarobotics.com/helpFiles/en/roslInterf.htm>

Exemple avec ajout d'une caméra embarquée : <https://www.ensta-bretagne.fr/zerr/dokuwiki/doku.php?id=vrep:ros-com-with-robot#v1>

L'évaluation repose sur deux éléments :

- ▶ Un projet V-REP en groupes de 5 à 6 élèves avec remise du résultat sur GitHub ou GitLab
- ▶ Un TD noté individuel (20 Avril 2021) de création d'un robot sous V-REP avec rendu sur MOODLE

Projet V-REP 2021 - modélisation du robot pédagogique DD-BOAT :

1. Modèle dynamique à partir de sphères réparties dans la coque
2. Flottaison et avance par ajout de forces (archimète, propulsion, ...) dans V-REP
3. Coque : Photogrammétrie et/ou LaserScan
4. Environnement : Lac de Guerlédan
5. Capteurs : Encodeurs, GNSS (GPS), Boussole, Gyros, Accéléros
6. Interface de contrôle/commande : ROS ou socket (client/serveur)

Projet V-REP 2021 - Organisation du travail :

- ▶ Création des 4 groupes et communication des membres pour création d'un canal TEAMS par groupe
- ▶ Création des dépôts git (GitHub ou GitLab) et communication des liens pour suivi du projet
- ▶ Outil de gestion de projet (ex. taiga <https://www.taiga.io/>) et communication du lien pour suivi du projet
- ▶ Définition et répartition des tâches

Comment modéliser un DDBOAT pour le faire flotter et avancer ?

- ▶ Approche "videogame" plutôt que "CFD" (Computational Fluid Dynamics)
- ▶ Modéliser la coque par des sphères et des moteurs par des forces pour simplifier les calculs (débuter avec une seule sphère)
- ▶ ou modéliser la coque avec des facettes (vérifier si le temps de calcul n'est pas prohibitif)
- ▶ liens intéressants :
https://www.gamasutra.com/view/news/237528/Water_interaction_model_for_boats_in_video_games.php
https://www.gamasutra.com/view/news/263237/Water_interaction_model_for_boats_in_video_games_Part_2.php
- ▶ V-REP permet d'ajouter des forces extérieures :
`sim.addForceAndTorque(handle,force,torque)`

Coque esthétique : Photogrammétrie et/ou LaserScan 3D

Scan lasers 3D disponibles ici :

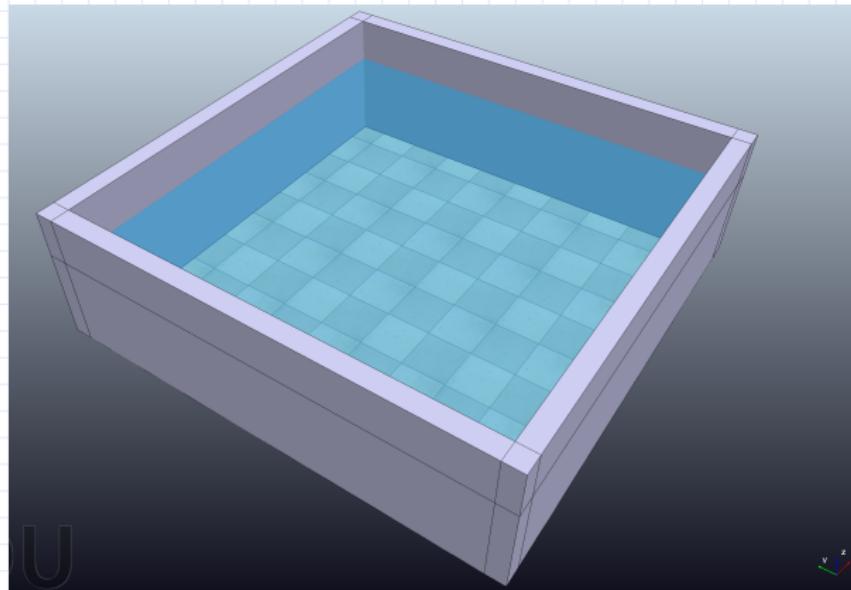
https://www.ensta-bretagne.fr/zerr/tmp/ddboat_xyz.zip

Photogrammétrie : logiciel Mic-Mac de l'IGN

<https://micmac.ensg.eu/index.php/Accueil>



Environnement simplifié de type bassin rectangulaire



Modélisation des robots

Projet V-REP 2021

Environnement complet : Créer un polygone simplifiant le contour du lac et modéliser les cotés par des "cuboids"



Simulation des capteurs

- ▶ Boussole : utilisation de la pose du DDBOAT dans V-REP
(sim.setObjectPosition, sim.getObjectOrientation,
sim.getObjectQuaternion ...) et conversion en degrés dans le repère géographique (0 degré au NORD). Ajout d'un bruit.
- ▶ GNSS (GPS) : transformation en latitude longitude de la position en coordonnées cartésienne dans V-REP (sim.setObjectPosition) et ajout d'un bruit (ne pas le coder en lua dans V-REP ! programme externe python ou noeud ROS)
- ▶ Gyros, Accéléros : calcul à partir des vitesses linéaires et angulaires données par V-REP (sim.getObjectVelocity) et ajout de bruit.
- ▶ Encodeurs : relier par une relation empirique la force moteur appliquée au DDBOAT et la vitesse de rotation du moteur
- ▶ Caméra vidéo : voir exemple <https://www.ensta-bretagne.fr/zerr/dokuwiki/doku.php?id=vrep:ros-com-with-robot#v1>

