Laboratorio n°1

Sara Portillo, Shamari Johnson

Estudiantes de III año de Ing. Mecatrónica, Curso IM 340, Grupo M3-3, Escuela de Mecatrónica

FIEC , Universidad de Panamá

- Simulador Coppelia

  Como sabemos, a lo largo del curso estaremos trabajando con este simulador, CoppeliaSim es un simulador de robótica de propósito general, desarrollado por la empresa: Coppelia Robotics. Tiene un entorno de desarrollo integrado, que se basa en una arquitectura de control distribuido, donde cada objeto/modelo puede controlarse individualmente.

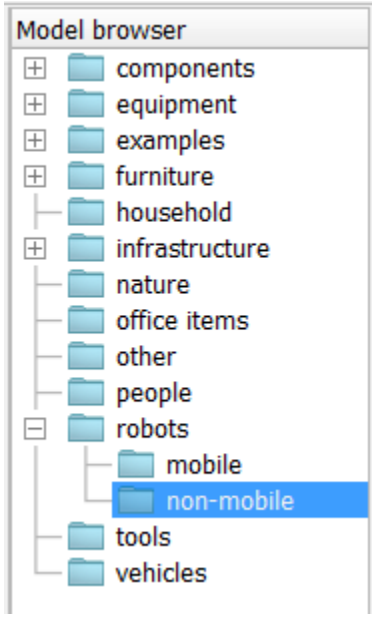
  CoppeliaSim se utiliza mayormente para el desarrollo rápido de algoritmos, simulaciones de automatización de fábricas, creación rápida de prototipos y verificación, monitoreo remoto y educación relacionada con la robótica.

- Botones y algunas funciones básicas
  - Una ventaja que hemos encontrado hasta ahora es que es bastante intuitivo su uso, entre algunos primeros botones tenemos:
    - Botones de simulación: se encargan principalmente de lo que su nombre indica, con ellos podemos iniciar, poner pausa, detener, modificar el tiempo y observar la simulación de manera más pausada o rápida dependiendo de lo que queramos

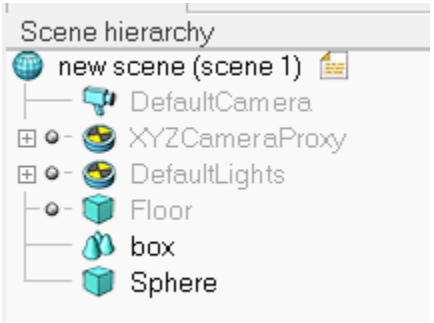
    - Botones de cámara: mediante ellos podemos girar o mover la vista que tenemos de nuestra pantalla o la “cámara” que estamos observando

- Botones de movimiento para el objeto : mediante estos podemos desplazar o girar el objeto, también seleccionar alguno de los que ya se encuentran en nuestra pantalla, etc

- Panel de modelos: mediante esta ventana podemos seleccionar alguno de los equipos que se encuentran disponibles en el simulador como, por ejemplo, los brazos robóticos, drones, carros, etc.
  Cabe destacar que también podemos añadir cosas más básicas en un inicio mediante la ventana “add” que nos muestra formas geométricas



- Ventana de jerarquía: esta básicamente nos muestra todos los objetos que tenemos ya en pantalla y la jerarquía con la que estos funcionan (cuales se pueden mover en base a otros)

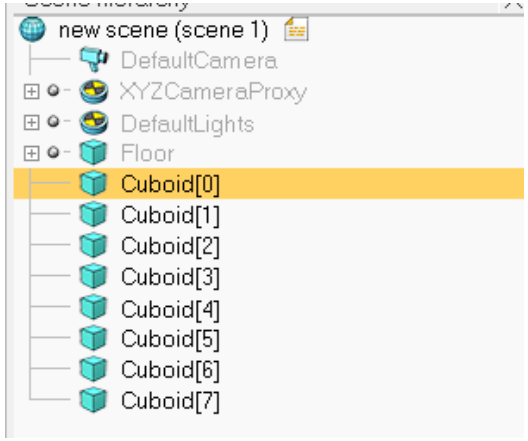


Propiedades de los materiales

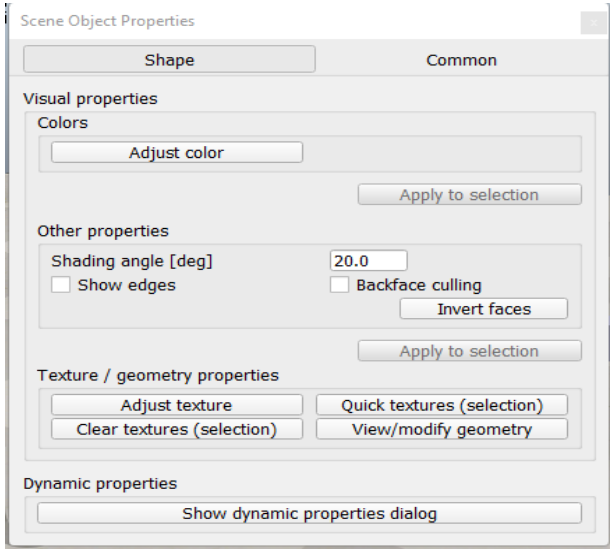
Como avance para este laboratorio decidimos estudiar primeramente las propiedades de los objetos en CoppeliaSim. Algunas de las propiedades físicas más importantes de los metales son la densidad, el color, el tamaño y la forma, el peso específico del material y la porosidad entre otras. Para esta experiencia hay que tener en consideración las siguiente conceptos:

- Reactivos
- No reactivos
- Dinámicos
- Estáticos

Primeramente para poder modificar las características de cualesquiera objetos en el programa o cambiar sus parámetros, es necesario ubicarse en la escena de jerarquía en la cual nos aparecerá cada uno de los elementos empleador en el programa.

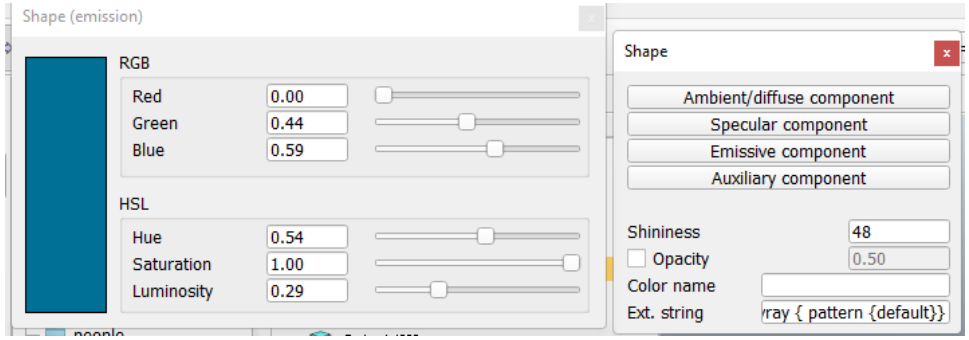


Dentro de esta clickeamos en el icono del objeto en este cabo el cubo lo cual nos abrirá a una ventana que muestra sus propiedades de este

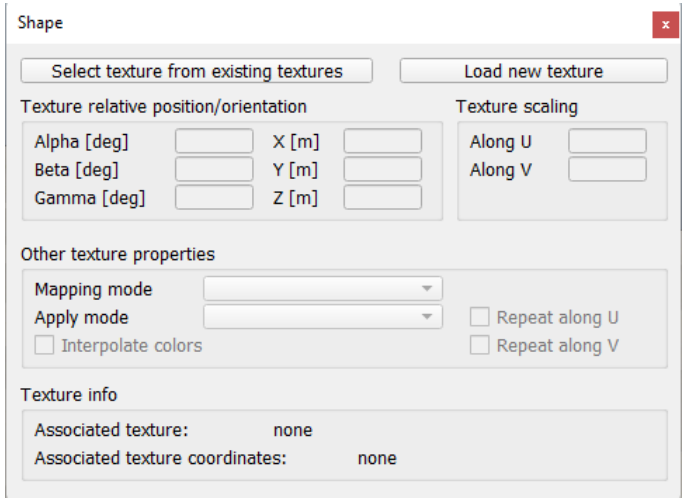


Formas(shapes)

- En la sección adjust color: como su nombre lo dice permite modificar el color del objeto.



- En la parte inferior nos permite agregar texturas



En la sección de propiedades comunes

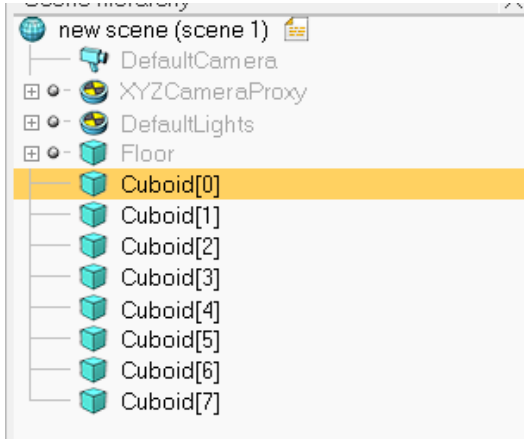
Propiedades de los materiales

Como avance para este laboratorio decidimos estudiar primeramente las propiedades de los objetos en CoppeliaSim. Algunas de las propiedades físicas más importantes de los metales son la densidad, el color, el tamaño y la forma, el peso

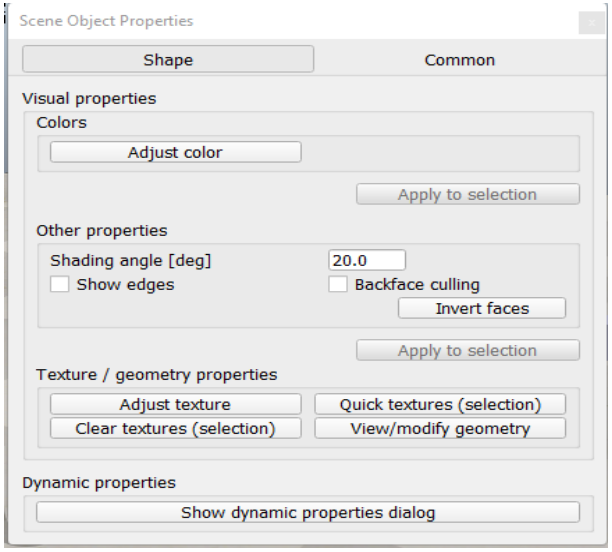
específico del material y la porosidad entre otras. Para esta experiencia hay que tener en consideración los siguientes conceptos:

- Reactivos
- No reactivos
- Dinámicos
- Estáticos

Primeramente, para poder modificar las características de cualesquiera objetos en el programa o cambiar sus parámetros, es necesario ubicarse en la escena de jerarquía en la cual nos aparecerá cada uno de los elementos empleador en el programa.

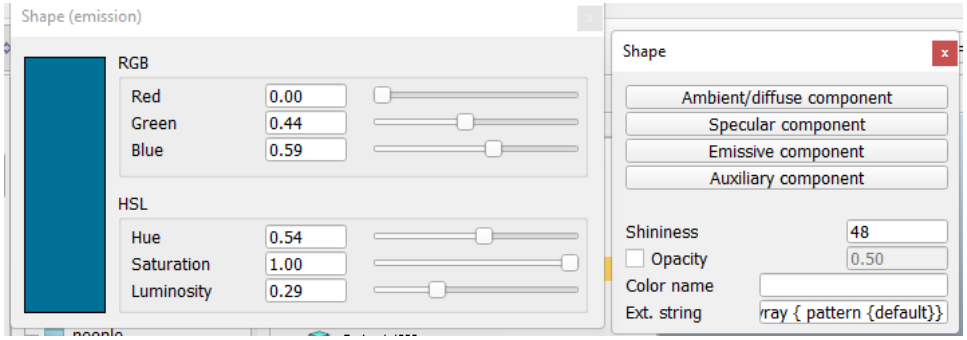


Dentro de esta clickeamos en el icono del objeto en este cabo el cubo lo cual nos abrirá a una ventana que muestra sus propiedades de este

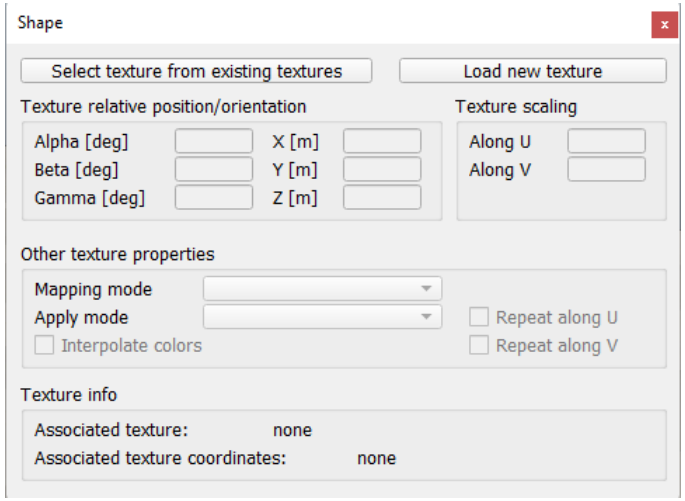


Formas(shapes)

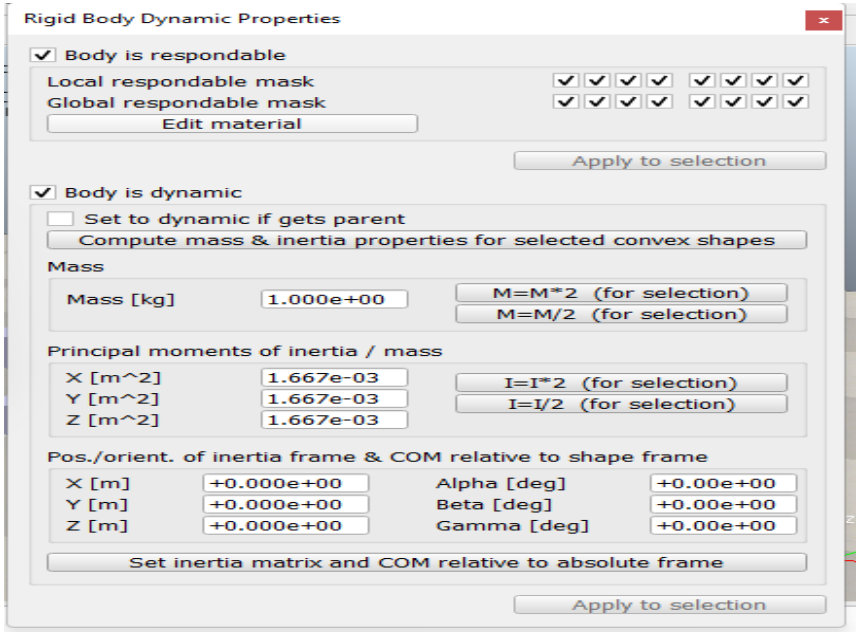
- En la sección adjust color: como su nombre lo dice permite modificar el color del objeto.



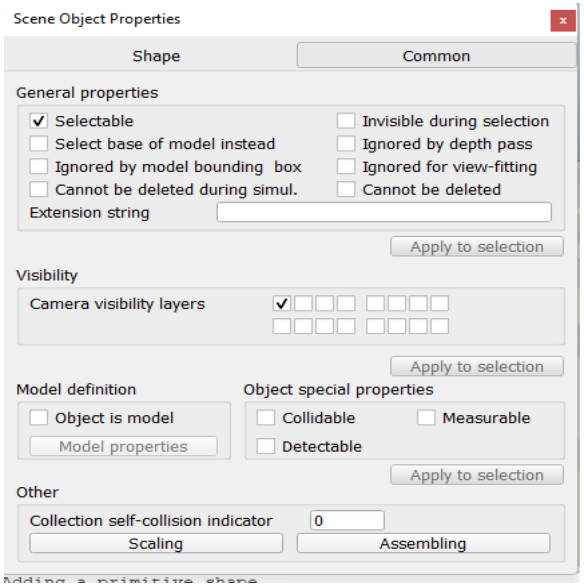
- En la parte inferior nos permite agregar texturas



- En **show dinamic properties**, Podemos determinar si el objeto es dinámico por lo se ve afectada por la gravedad, colisiones entre otras reacciones o no, lo cual sería estático en este caso dichos factores son irrelevantes en simulación. En el caso de ser dinámicos se puede modificar valores como la masa del objeto y su momento de inercia (con estas se pueden realizar cálculos y obtener ecuaciones matemáticas). También se puede determinar en esta pestaña si es reactivo (body is responsable) esto quiere decir que al momento de chocar un objeto con otro se podrá observar la reacción en el cual influye la fuerza colisión.


Rigid Body Dynamic Properties
Body is respondable
Local respondable mask
Global respondable mask
Edit material
Apply to selection
Body is dynamic
Set to dynamic if gets parent
Compute mass & inertia properties for selected convex shapes
Mass
Mass [kg]
1.000e+00
M=M*2 (for selection)
M=M/2 (for selection)
Principal moments of inertia / mass
X [m^2]
Y [m^2]
Z [m^2]
1.667e-03
1.667e-03
1.667e-03
I=I*2 (for selection)
I=I/2 (for selection)
Pos./orient. of inertia frame & COM relative to shape frame
X [m]
Y [m]
Z [m]
+0.000e+00
+0.000e+00
+0.000e+00
Alpha [deg]
Beta [deg]
Gamma [deg]
+0.00e+00
+0.00e+00
+0.00e+00
Set inertia matrix and COM relative to absolute frame
Apply to selection

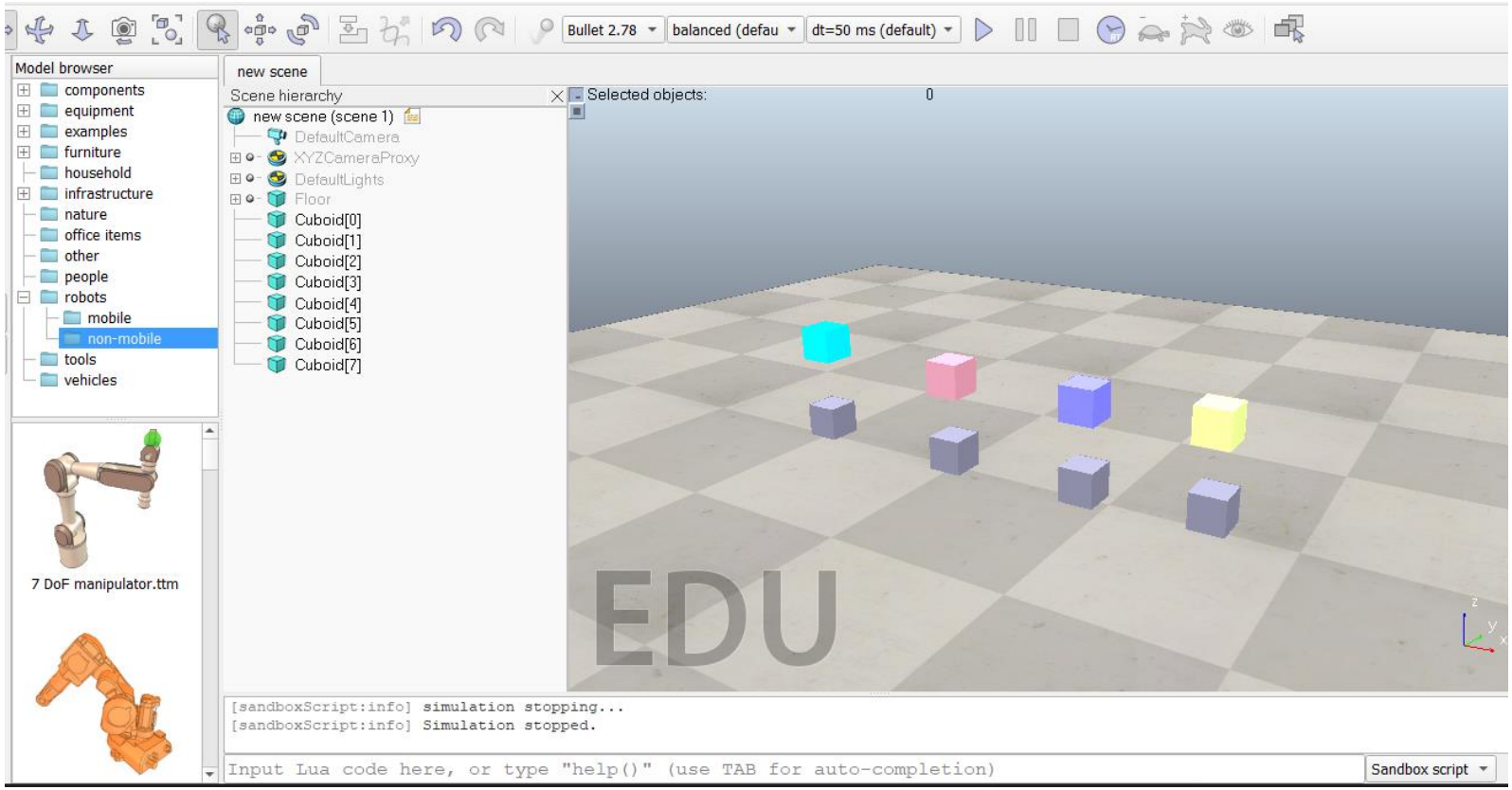
En la sección de propiedades comunes



- En esta sección determinamos si el objeto puede ser colacionable, medido, y detectable.
- También en que láminas de visibilidad se encuentran

Ya considerando todo los conceptos y parámetros anteriores procedemos a realizar una pequeña experiencia en el cual utilizamos como objeto 8 cubos en las cuales poseerán diferentes cualidades para observar su comportamiento. Ene el cual:

1. El primer par ambos serán dinámicos y reactivo
2. El segundo par uno será dinámico y no reactivo mientras que el otro será dinámico y reactivo
3. El tercer par un objeto será estáticos y reactivo
4. El ultimo ambos son no reactivos y dinámico


Bullet 2.78
balanced (defau
dt=50 ms (default)
Model browser
components
equipment
examples
furniture
household
infrastructure
nature
office items
other
people
robots
mobile
non-mobile
tools
vehicles
7 DoF manipulator.ttm
new scene
Scene hierarchy
new scene (scene 1)
DefaultCamera
XYZCameraProxy
DefaultLights
Floor
Cuboid[0]
Cuboid[1]
Cuboid[2]
Cuboid[3]
Cuboid[4]
Cuboid[5]
Cuboid[6]
Cuboid[7]
Selected objects: 0
EDU
[sandboxScript:info] simulation stopping...
[sandboxScript:info] Simulation stopped.
Input Lua code here, or type "help()" (use TAB for auto-completion)
Sandbox script

Simulación Cinemática Inversa

La cinemática inversa se refiere a determinar los parámetros comunes que proporcionan una posición deseada del efector final. Especificación del movimiento de un robot de manera que su extremo efector logra una tarea deseada es conocido como planificación de movimientos.

SIMULACIÓN

Diseñar un sistema que por medio de un sensor pueda detectar un color especifico de la caja, haciendo detener la banda transportadora para que esta manera el robot cree un camino vació para transportar el objeto a la segunda banda.

PARÁMETROS ESPECÍFICOS

A continuación, se presentará la medición planificados de movimiento para que el robot cumpla las funciones en específicos.

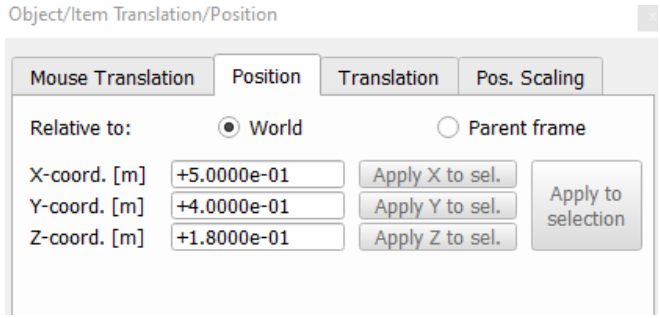
Posición de la primera banda transportadora

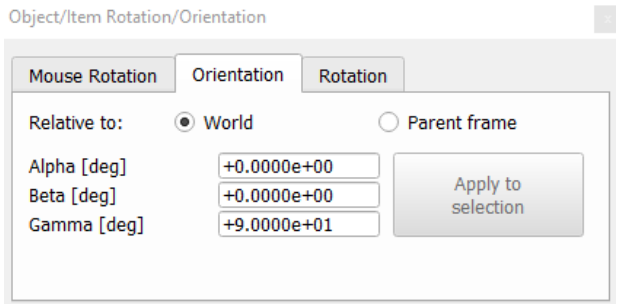
Posición: X= -0.7; Y= -0.5; Z=0.1

Posición y orientación de la segunda banda transportadora

Orientación: Alpha=0° ; Beta=0°; Gamma=90°

Posición: X=0.5; Y=0.4; Z=0.18



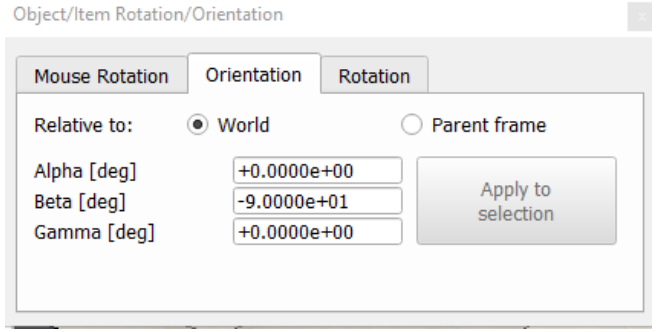


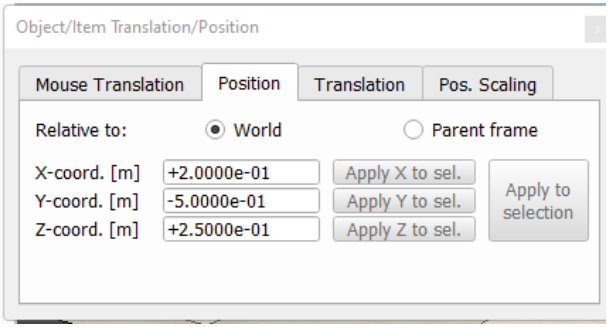
PRIMERA BANDA TRANSPORTADORA

Posición y orientación sensor de proximidad (Belt1)

Orientación: Alpha=0° ; Beta=-90°; Gamma=0°

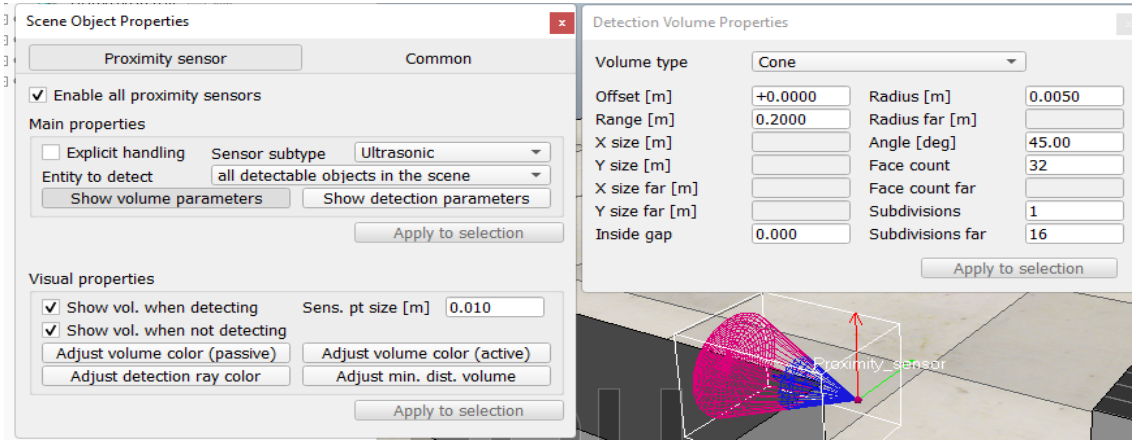
Posición: X=0.2; Y=0.5; Z=0.25





Parámetros de volumen del sensor

En esta sección dentro de la propiedad del objeto (sensor), accionamos “muestra el volumen de parámetros”, en el cual determinamos la forma geométrica de un cono y radio en el cual el sensor podrá detectar el tamaño de las cajas y la banda se detenga como muestra la imagen 1.



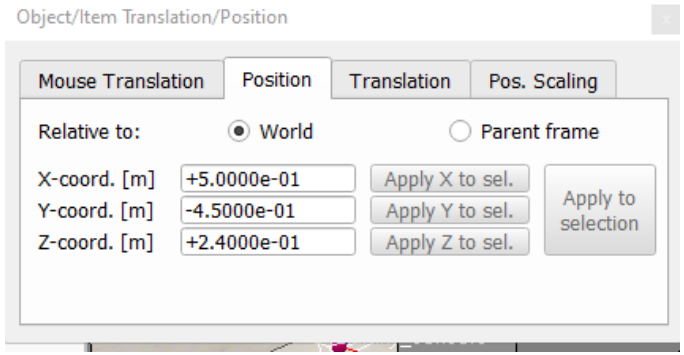
SEGUNDA BANDA TRANSPORTADORA

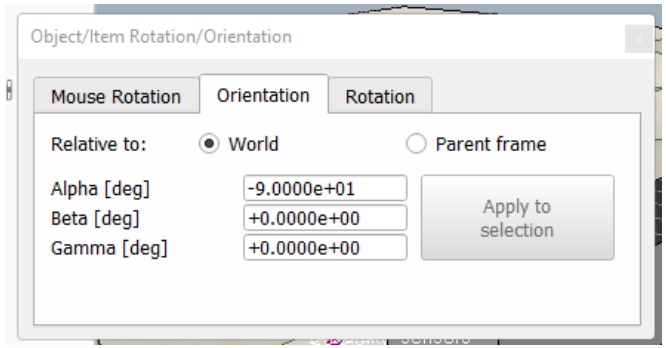
Para la segunda banda tendremos dos sensores una al inicio de la banda y otra a la salida, con la finalidad de cuando el robot transporte las cajas a la segunda banda, los sensores las detecte y corra la banda hasta cierta distancia dándole paso a otras cajas.

Posición y orientación del sensor de la proximidad (Belt2_1)

Orientación: Alpha= -90° ; Beta=0°; Gamma=0°

Posición: X=0.5; Y= -0.45; Z=0.24

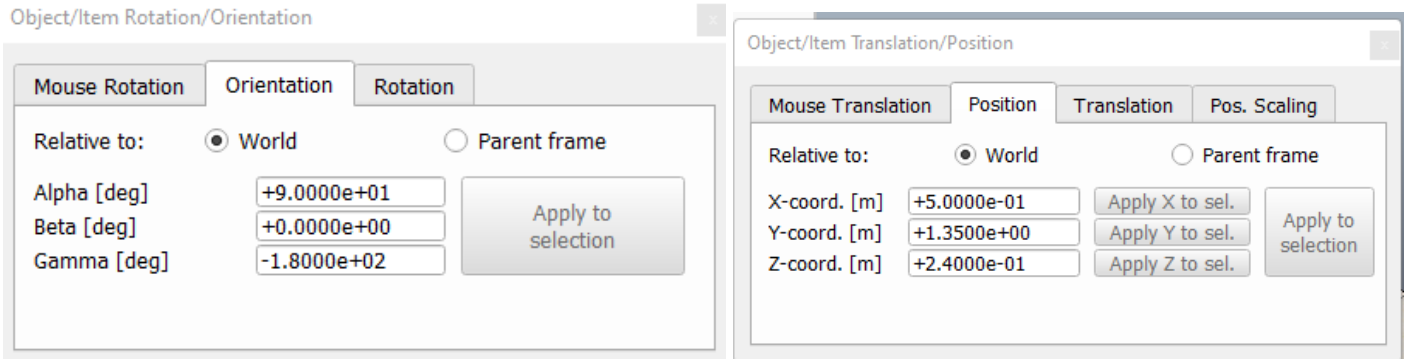




Posición y orientación Belt2_2

Orientación: Alpha=0° ; Beta=-90°; Gamma=0°

Posición: X=0.2; Y=0.5; Z=0.25



Parámetros de volumen

En esta sección dentro de la propiedad del objeto (sensor), accionamos “muestra el volumen de parámetros”, en el cual determinamos la forma geométrica de rayo en el cual el sensor podrá detectar la distancia en de las cajas como muestra la imagen1, con el fin de que la banda corra y se detenga hasta que el segundo sensor lo detecte.

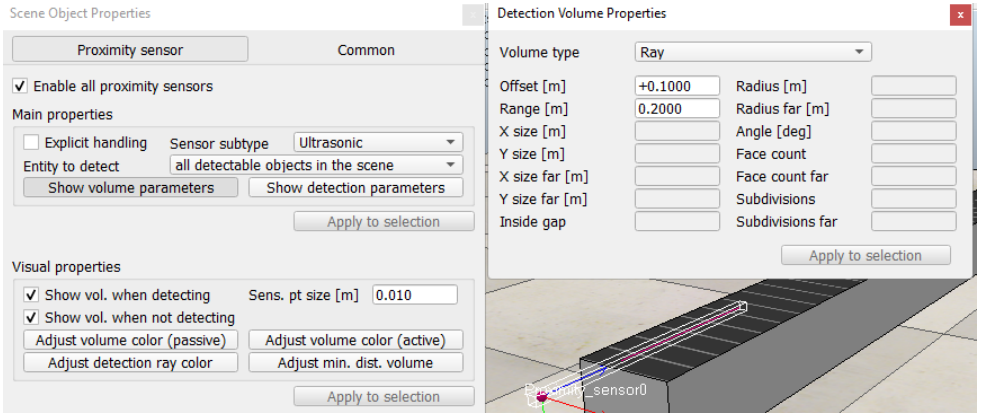
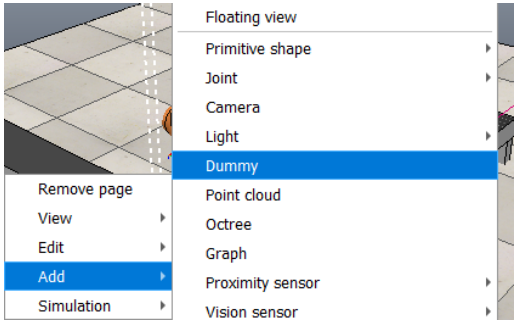


Ilustración 1

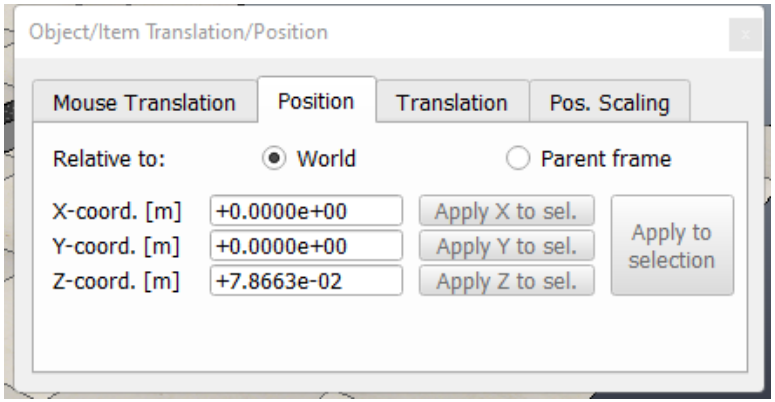
DUMMY

Teniendo en cuenta que un dummy es un punto con orientación y se puede ver como un marco de referencia. Creamos un dummy box el cual colocamos en una posición dentro de la caja de referencia que se crea igualmente en primitive shape.



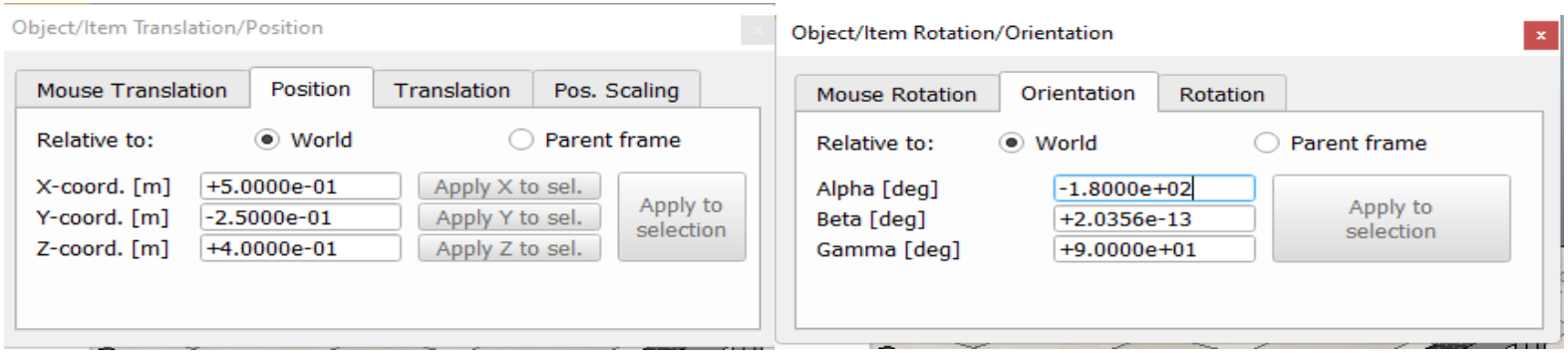
KUKA

El robot que emplearemos es un robot KUKA iiwa 14 en el cual obtendrá las siguientes coordenadas de posición como muestra la siguiente imagen:



DUMMY (reléasePos)

La creación de este dummy tiene la finalidad de tomar el pun to de referencia en el cual el robot depositará la caja en la segunda banda transportadora.



DUMMYs

Igualmente se crea nuevamente 3 dummys mas, el cual el connector será el punto de referencia en el cual se sujetará la caja, el target es el punto de referencia para tomar la caja de la primera banda transportadora, y tip es el punto de referencia del efector del robot. Estos dummys tiene la finalidad de crear caminos vacios para que el robot opere.

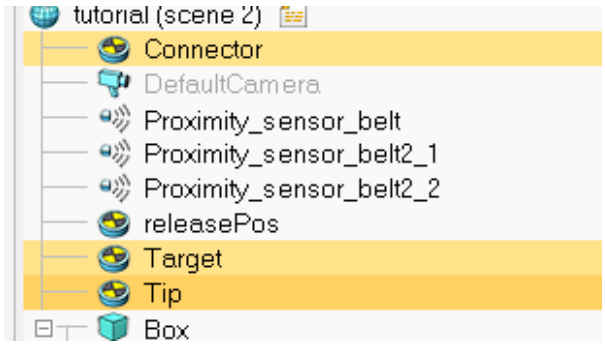


Ilustración 2

A continuación se muestrearan las ubicaciones en las cuales dichos dummys forman parte de la Jerarquía como muestra la siguiente imagen:

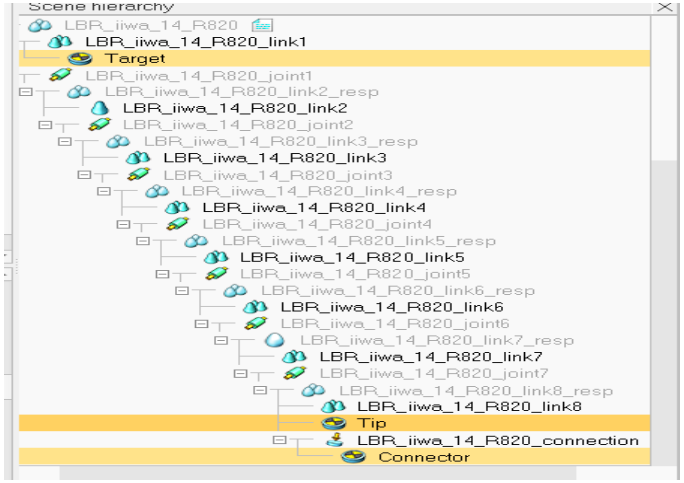
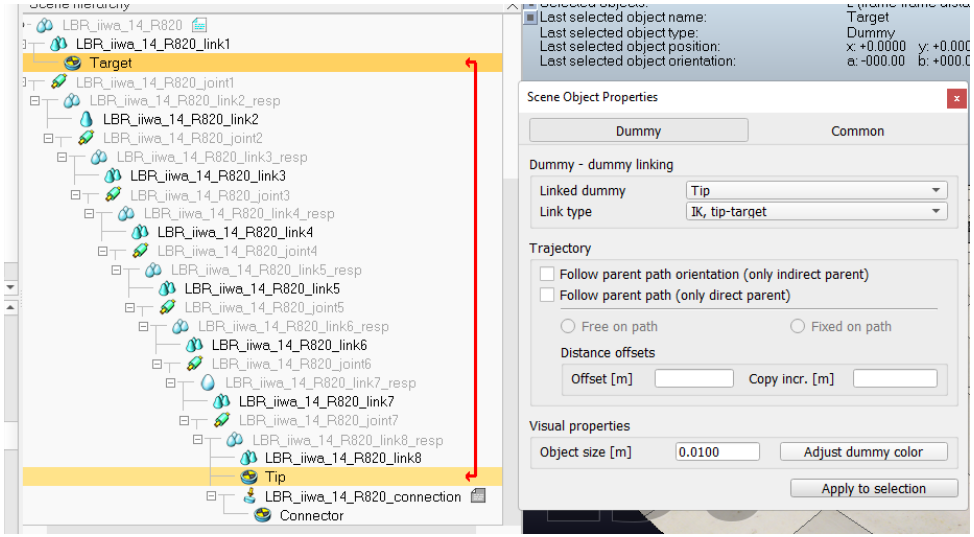
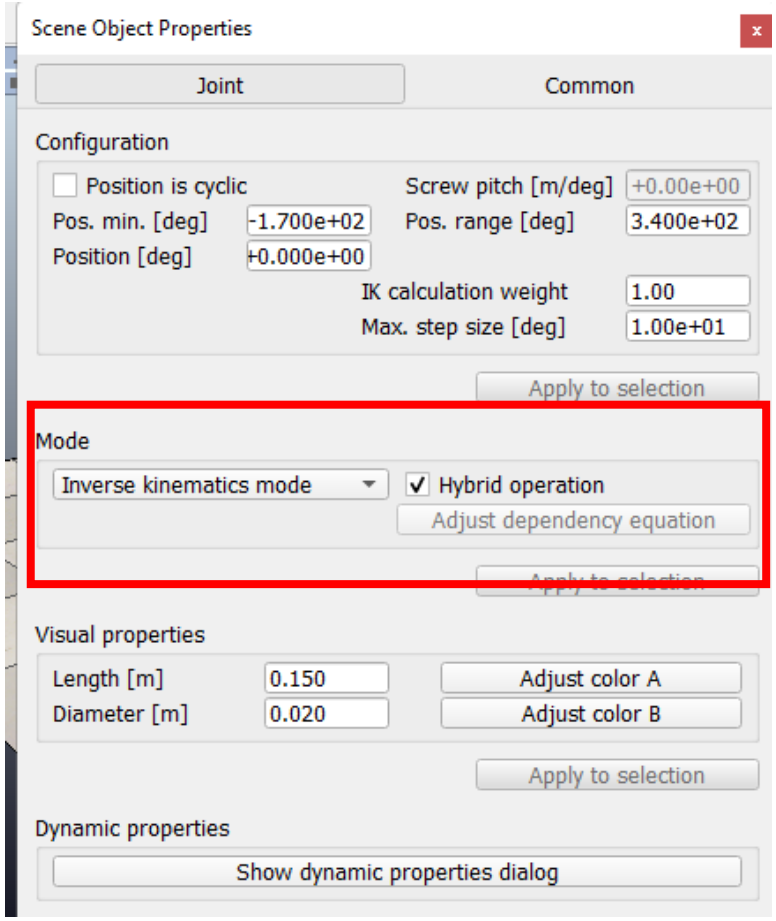


Ilustración 3

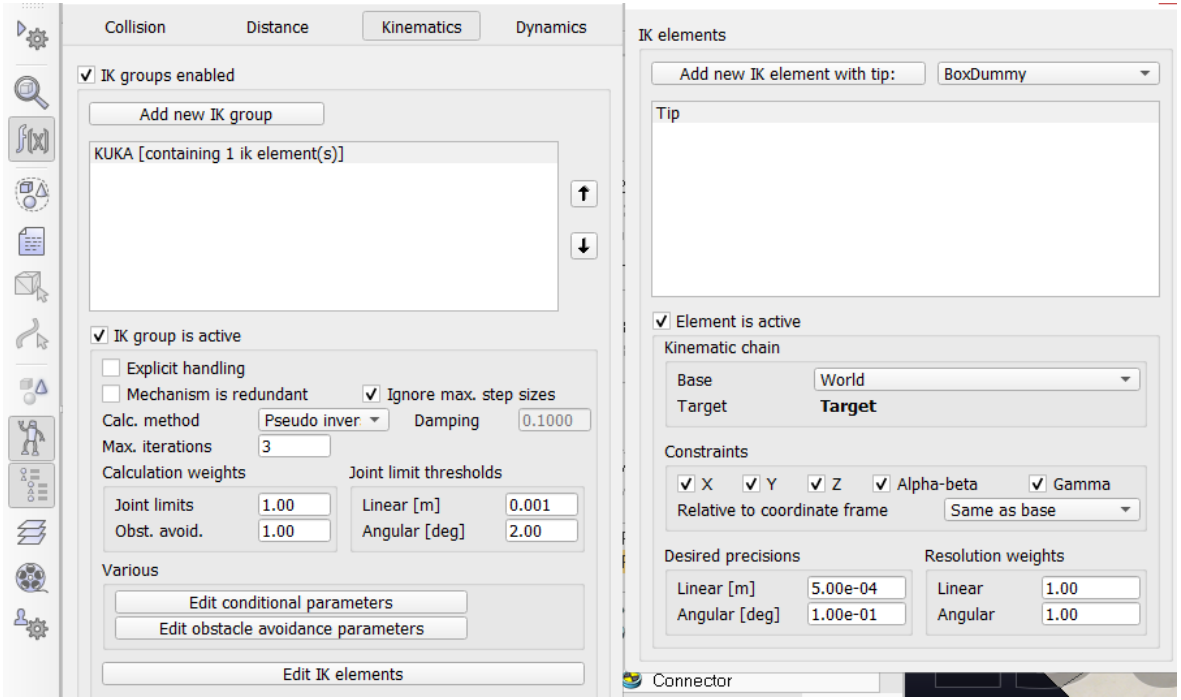
En esta sección hacemos conectamos el tip con el target en las propiedades del objeto (linked dummy), con la finalidad de tener dependencia respecto a orientación y posición una de la otra a la hora del robot movilizarse.



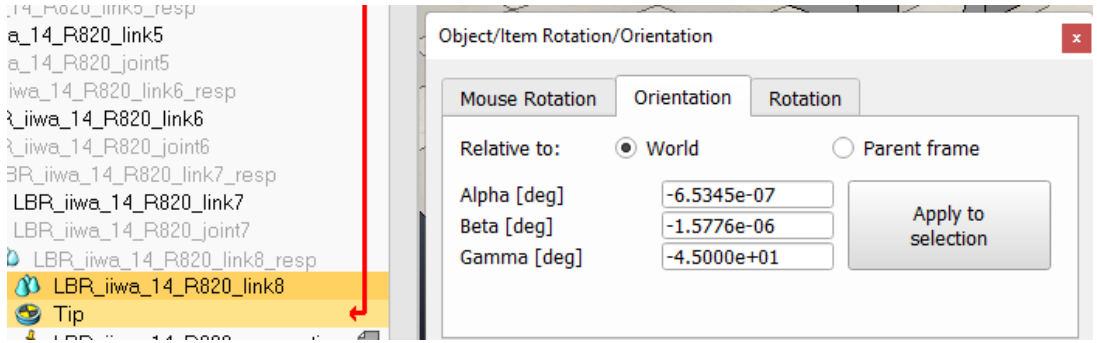
En las propiedades del objeto activamos para todos los joints el modo cinemático inversa, para que actúe de la manera deseada y también activar operación hibrida para cumplir un proceso modular y escalable, el cual es adaptado a las necesidades.



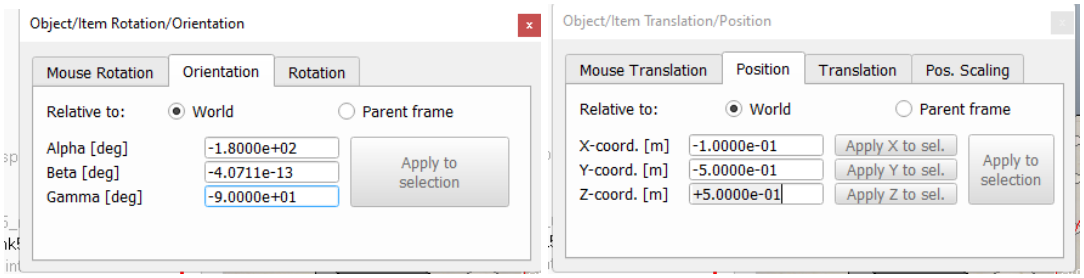
En modulo de las propiedades de cálculo cinemático agregamos un nuevo grupo IK que es el robot KUKA cuya función es para que opere con los cálculos realizados que se programan con el código, y en agregar elemento agregamos el dummy tip el cual es el que se moverá junto al robot.



Como anteriormente se mencionó el Tip dependerá del efectuador es por eso por lo que se posiciona y orienta de la misma manera que el efectuador, para ello tocamos el botón apply to selection el cual cumplirá la función de adaptarlos de igual ubicación.



Las posiciones y orientación en el cual es dummy target toma lugar es la siguiente:



Las siguientes ilustraciones muestran los grados en las cuales las articulaciones deben orientarse como punto de referencia, para que el robot opere como se desea.

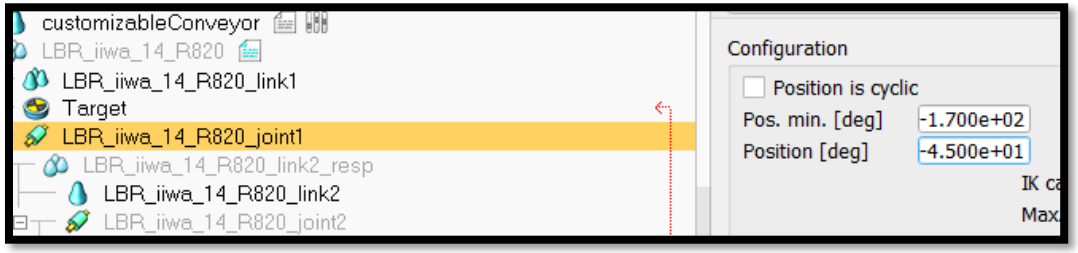


Ilustración 4: orientación de la primera articulación

Ilustración 5: orientación de la segunda articulación.

Ilustración 6: orientación de la cuarta articulación.

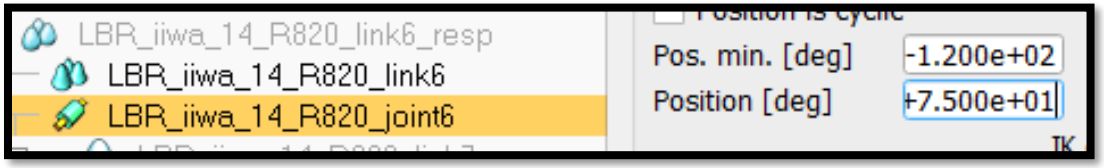


Ilustración 7: orientación de la sexta articulación.

Ya establecidas los parámetros de orientación y posición deseados para el robot obtendremos los siguientes resultados como muestra la siguiente imagen:

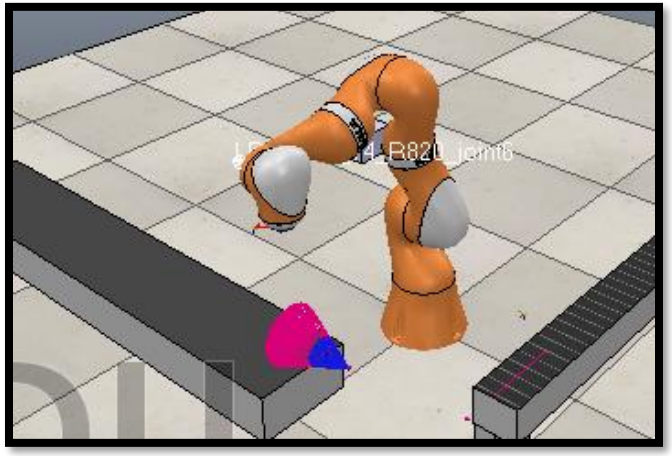


Ilustración 8: posición y orientación de referencia.

A partir de todas las posiciones y orientaciones establecidas por medio de programación se realiza la función planificada:

PROGRAMCION DEL LA BANDA TRANSPORTADORA

```
function sysCall_init()
    -- User Parameters
    beltSpeed = 0.4
    T_insert = 1
    insertCoordinate = {-1.3,-0.5,0.25}
    goodPercentage = 0.19
    goodColor = {0.345,0.859,0.192}

    -- Initialize auxiliary variables
    T_last_inserted = 0
    deltaTime = 0
    hasStopped = false
    boxList = {}
    boxDummyList = {}
    boolList = {}

    -- Initialize handles, set beltSpeed
    box = sim.getObjectHandle("Box")
```

```
boxDummy = sim.getObjectHandle("BoxDummy")

forwarder=sim.getObjectHandle('ConveyorBelt_forwarder')
proximity = sim.getObjectHandle("Proximity_sensor_belt1")

belt2script = sim.getScriptHandle("customizableConveyor")

sim.setScriptSimulationParameter(sim.handle_self,"conveyorBeltVelocity",beltSpeed)



-- Insert the first box during initializiation
insertBox()

-- Get handles and postions of dummies
targetDummy = sim.getObjectHandle("Target")
idlePos = sim.getObjectPosition(targetDummy,-1)
idleOrient = sim.getObjectOrientation(targetDummy,-1)

releasePosHandle = sim.getObjectHandle("releasePos")
releasePos = sim.getObjectPosition(releasePosHandle,-1)
```

```
    releaseOrient = sim.getObjectOrientation(releasePosHandle,-1)

    -- Get release path handle
    releasePath = createPath("releasePath",idlePos,idleOrient,releasePos,releaseOrient)

    -- Get robot script handle
    robotScriptHandle = sim.getScriptHandle("LBR_iiwa_14_R820")
    sim.setScriptVariable("releasePath",robotScriptHandle,releasePath)

    -- Create "Dummy Path" (will be removed)
    path = sim.createPath(1);
    sim.setObjectName(path,"pickupPath") --camino del dummy tip de la m?quina y el dummy de la caja
end
function sysCall_cleanup()

end
function sysCall_actuation()
    beltVelocity=sim.getScriptSimulationParameter(sim.handle_self,"conveyorBeltVelocity")

    -- Here we "fake" the transportation pads with a single static rectangle that we dynamically reset
```

```
    -- at each simulation pass (while not forgetting to set its initial velocity vector) :

    relativeLinearVelocity={beltVelocity,0,0}
    -- Reset the dynamic rectangle from the simulation (it will be removed and added again)
    sim.resetDynamicObject(forwarder)
    -- Compute the absolute velocity vector:
    m=sim.getObjectMatrix(forwarder,-1)
    m[4]=0 -- Make sure the translation component is discarded
    m[8]=0 -- Make sure the translation component is discarded
    m[12]=0 -- Make sure the translation component is discarded
    absoluteLinearVelocity=sim.multiplyVector(m,relativeLinearVelocity)
    -- Now set the initial velocity of the dynamic rectangle:
    sim.setObjectFloatParameter(forwarder,sim.shapefloatparam_init_velocity_x,absoluteLinearVelocity[1])
    sim.setObjectFloatParameter(forwarder,sim.shapefloatparam_init_velocity_y,absoluteLinearVelocity[2])
    sim.setObjectFloatParameter(forwarder,sim.shapefloatparam_init_velocity_z,absoluteLinearVelocity[3])
end
function sysCall_sensing()
    -- Read Proximity sensor (0= nothing detected, 1 = object detected)
    local res = sim.readProximitySensor(proximity)
```

```
-- Check if possible to insert an new box
if (sim.getSimulationTime()-T_last_inserted > T_insert) and not hasStopped then
    insertBox()
end

-- If proximity sensor detects an object, stop the belt, stop inserting objects
if res == 1 and not hasStopped then
    if boolList[1] then
        sim.setScriptSimulationParameter(sim.handle_self,"conveyorBeltVelocity",0)
        deltaTime = sim.getSimulationTime()-T_last_inserted
        hasStopped = true

            -- Generate new pickupPath
        updatePickupPath(boxDummyList[1]) --actualizar el  camino para diferentes posiciones y orientacion
            -- Remove first object and dummy handle from table
        objs = removeFirstObject()
            -- Set pickupDummy-handle in robot script
        sim.setScriptVariable("pickupDummy",robotScriptHandle,objs[2])
            -- Set a signal such that robot knows that object is available
        sim.setIntegerSignal("objectAvailable",1)
```

```
        else
            local box = table.remove(boxList,1)
            local boxDummy = table.remove(boxDummyList,1)
            table.remove(boolList,1)

            sim.removeObject(box)
            sim.removeObject(boxDummy)
        end
    end

    -- If proximity sensor detects nothing and belt has stopped, start belt, continue inserting
    if res == 0 and hasStopped then
        sim.clearIntegerSignal("objectAvailable")
        sim.setScriptSimulationParameter(sim.handle_self,"conveyorBeltVelocity",beltSpeed)
        hasStopped = false
        T_last_inserted = sim.getSimulationTime()-deltaTime
    end
end

function removeFirstObject()
```

```
    -- Obtain handles by removing from tables
    local box = table.remove(boxList,1)
    local boxDummy = table.remove(boxDummyList,1)
    table.remove(boolList,1)

    -- Add handles to the belt2 tables
    sim.callScriptFunction("addObject",belt2script,{box,boxDummy}) --pasa la infromacion de una banda a otra

    -- Return handles
   return {box,boxDummy}
end

function insertBox()
    -- Generate random numbers
    local rand1 = math.random()
    local rand2 = math.random()
    local rand3 = math.random()

    -- Generate random disturbances on position and orientation
    local dx = (2*rand1-1)*0.1
```

```
local dy = (2*rand2-1)*0.1
local dphi = (2*rand3-1)*0.5
local disturbedCoordinates = {0,0,0}
disturbedCoordinates[1] = insertCoordinate[1]+dx
disturbedCoordinates[2] = insertCoordinate[2]+dy
disturbedCoordinates[3] = insertCoordinate[3]

-- Copy and paste box and boxDummy
local insertedObjects = sim.copyPasteObjects({box,boxDummy},0)

-- Update last inserted box time
T_last_inserted = sim.getSimulationTime()

-- Move and rotate
sim.setObjectPosition(insertedObjects[1],-1,disturbedCoordinates)
sim.setObjectOrientation(insertedObjects[1],-1,{0,0,dphi})

-- Store handles to boxes and dummies
table.insert(boxList,insertedObjects[1])
table.insert(boxDummyList,insertedObjects[2])
```

```
    -- Decide if object is good or bad
    local decision = math.random()
    if decision <= goodPercentage then
            -- Object is good, assign goodColor
        sim.setShapeColor(insertedObjects[1],nil,sim.colorcomponent_ambient_diffuse,goodColor)
        table.insert(boolList,true)
    else
            -- Object is bad, assign random color
        sim.setShapeColor(insertedObjects[1],nil,sim.colorcomponent_ambient_diffuse,{rand1,rand2,rand3})
        table.insert(boolList,false)
    end

end

function createPath(name,startPoint,startOrient,endPoint,endOrient)
    -- Create Path Object
    local path = sim.createPath(1)

    -- Create buffer variables
```

```
    local buffer = {startPoint[1],startPoint[2],startPoint[3],startOrient[1],startOrient[2],startOrient[3], 1,0,0,0,0,
                    endPoint[1],endPoint[2],endPoint[3],endOrient[1],endOrient[2],endOrient[3],             1,0,0,0,0}

    -- Insert 2 control points (start and endpoint)
    sim.insertPathCtrlPoints(path,0,0,2,buffer)

    -- Rename the object
    sim.setObjectName(path,name)
    -- Return handle to path
   return path
end

function updatePickupPath(dummy) --
    -- Obtain handle to last pickupPath
    local path = sim.getObjectHandle("pickupPath")
    -- Remove the path
    sim.removeObject(path)
    -- Obtain position of dummy to be reached
    local dummyPos = sim.getObjectPosition(dummy,-1) --posicion
    -- Obtain orientation of dummy to be reached
```

```
  local dummyOrient = sim.getObjectOrientation(dummy,-1)--orientaci?n
    -- Create new path
    createPath("pickupPath",idlePos,idleOrient,dummyPos,dummyOrient)
end



function addObject(obj)
    -- Insert box and boxDummy handle at the end of tables
    table.insert(boxList,obj[1])
    table.insert(boxDummyList,obj[2])
end

function removeObject()
    -- Remove first objects from tables, then remove objects from scene
    sim.removeObject(table.remove(boxList,1))
    sim.removeObject(table.remove(boxDummyList,1))
end
```

PROGRAMACIÓN DEL ROBOT

```
-- Velocity and acceleration on path
```

```
nominalVel = 0.25
nominalAcc = 0.5

-- Get object and script handles
target = sim.getObjectHandle("Target")
connector = sim.getObjectHandle("Connector")
belt1_script = sim.getScriptHandle("ConveyorBelt")

-- Initialize variables
pickupDummy = -1
releasePath = -1

function sysCall_threadmain()
    while sim.getSimulationState()~=sim.simulation_advancing_abouttostop do
        -- Pause script until a signal is applied on integer signal "objectAvailable"
        sim.waitForSignal("objectAvailable")
        -- Obtain current pickupPath-handle
        path = sim.getObjectHandle("pickupPath")
        -- Follow the pickupPath
        sim.followPath(target,path,3,0,nominalVel,nominalAcc)
```

```
        -- Wait one second to mimic a connection process
        sim.wait(1)
        -- Connect the connector to pickupDummy
        sim.setLinkDummy(connector,pickupDummy)
        -- Set link type
        sim.setObjectInt32Parameter(connector,sim.dummyintparam_link_type,sim.dummy_linktype_dynamics_loop_closure)
        -- Follow back the pickup path
        sim.followPath(target,path,3,1,-nominalVel,-nominalAcc)
        -- Follow release path
        sim.followPath(target,releasePath,3,0,nominalVel,nominalAcc)
        -- Wait 0.25 seconds
        sim.wait(0.25)
        -- Disconnect pickupDummy from connector
        sim.setLinkDummy(connector,-1)
        -- Follow back releasePath to "idle" position
        sim.followPath(target,releasePath,3,1,-nominalVel,-nominalAcc)
    end
end
```

INCONVENIENTES DE LA EXPERIENCIA

Se logró la mayor parte del objetivo, en cambio nuestra mayor dificultad es la programación ya que no contamos con una gran experiencia en dicha área, a pesar de que gran parte de ello se obtiene de internet es grande la confusión para el entendimiento completo del código. Sin embargo, el principal objetivo de que el robot puede transportar la caja se cumplió más no la posición adecuada, es por esto que sigue en investigación.