PONTIFICIA
UNIVERSIDAD
CATÓLICA DE
VALPARAÍSO

pucv.cl

Robótica e inteligencia artificial

Módulo 2
Robótica Móvil S10

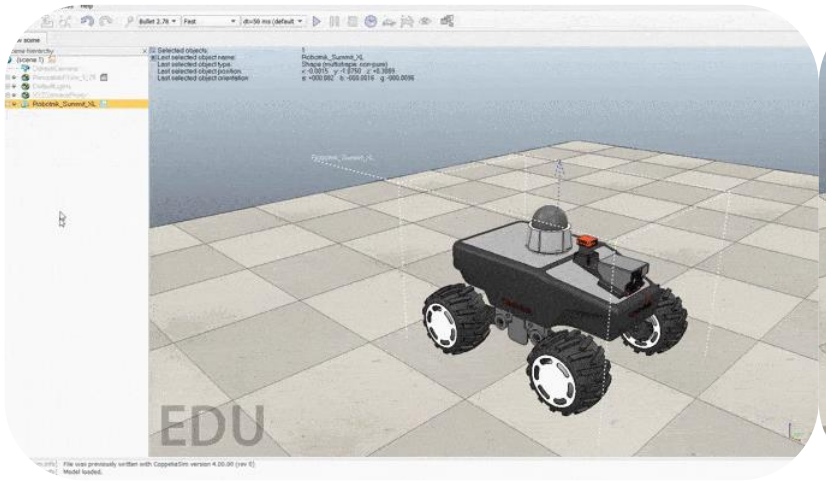
PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO

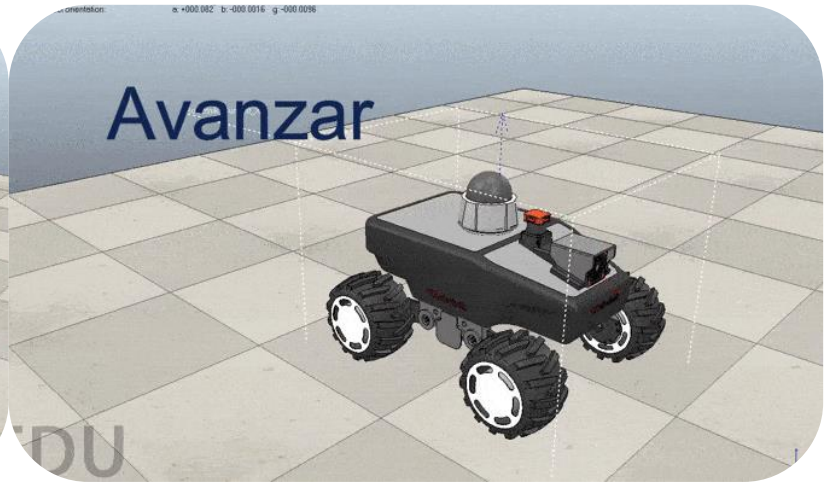
ROBÓTICA MÓVIL
SESIÓN 10

Programación de algoritmos en CoppeliaSim

El robot Summit XL de Robotnik posee cuatro ruedas Y lo convierten en un robot diferencial, ya que cada 1 de estos actuadores tipo motor de giro continuo son independientes unos de otros. El robot carece de un sistema de volante que le dé la dirección de giro, por lo que las acciones de viraje deben ser ejecutadas por medio de la asincronía de sus cuatro ruedas.

El método de programación del robot es sencillo y fácil de implementar al solo establecer parámetros de potencia y sentido de giro para cada una de sus ruedas.



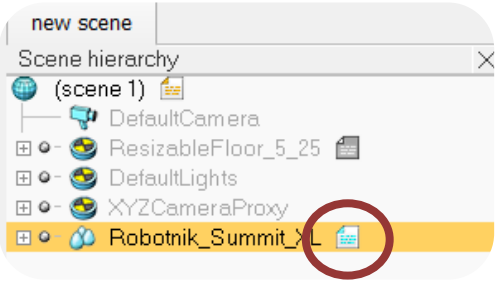


Programación de algoritmos en CoppeliaSim

Programación de Summit XL

Para programar un algoritmo en el robot, se debe abrir el código de este.

El código tiene una estructura ordenada y objetiva, donde se debe obedecer una sintaxis para generar correctamente el algoritmo que se desea.



caded child script (Robotnik_Summit_XL)

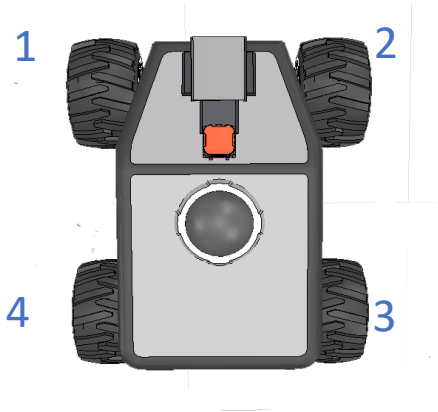
```
function sysCall_threadmain()
    motorHandles={-1,-1,-1,-1}

    motorHandles[1]=sim.getObjectHandle('joint_front_left_wheel')
    motorHandles[2]=sim.getObjectHandle('joint_front_right_wheel')
    motorHandles[3]=sim.getObjectHandle('joint_back_right_wheel')
    motorHandles[4]=sim.getObjectHandle('joint_back_left_wheel')



    while true do

     sim.setJointTargetVelocity(motorHandles[1],1)
     sim.setJointTargetVelocity(motorHandles[2],-1)
     sim.setJointTargetVelocity(motorHandles[3],-1)
     sim.setJointTargetVelocity(motorHandles[4],1)
     sim.wait(2)

     sim.setJointTargetVelocity(motorHandles[1],-1)
     sim.setJointTargetVelocity(motorHandles[2],1)
     sim.setJointTargetVelocity(motorHandles[3],1)
     sim.setJointTargetVelocity(motorHandles[4],-1)
     sim.wait(2)
     end
end
```

Programación de algoritmos en CoppeliaSim

Programación de Summit XL

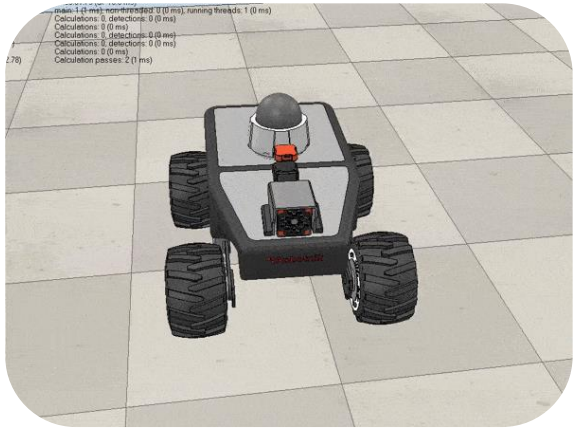
Si se desea mover simultáneamente las ruedas de un robot, el programa debe tener indicado los siguientes aspectos.

- El motor a mover.
- La potencia del motor a mover.
- La duración de la acción.



..readed child script (Robotnik_Summit_XL)

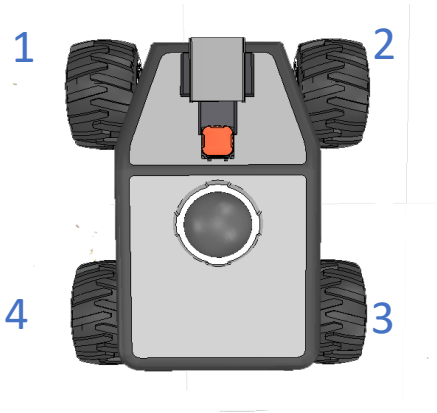
```
function sysCall_threadmain()
    motorHandles={-1,-1,-1,-1}

    motorHandles[1]=sim.getObjectHandle('joint_front_left_wheel')
    motorHandles[2]=sim.getObjectHandle('joint_front_right_wheel')
    motorHandles[3]=sim.getObjectHandle('joint_back_right_wheel')
    motorHandles[4]=sim.getObjectHandle('joint_back_left_wheel')



    while true do

     sim.setJointTargetVelocity(motorHandles[1],1)
     sim.setJointTargetVelocity(motorHandles[2],0)
     sim.setJointTargetVelocity(motorHandles[3],0)
     sim.setJointTargetVelocity(motorHandles[4],0)
     sim.wait(1)
     end
end
```



Programación de algoritmos en CoppeliaSim

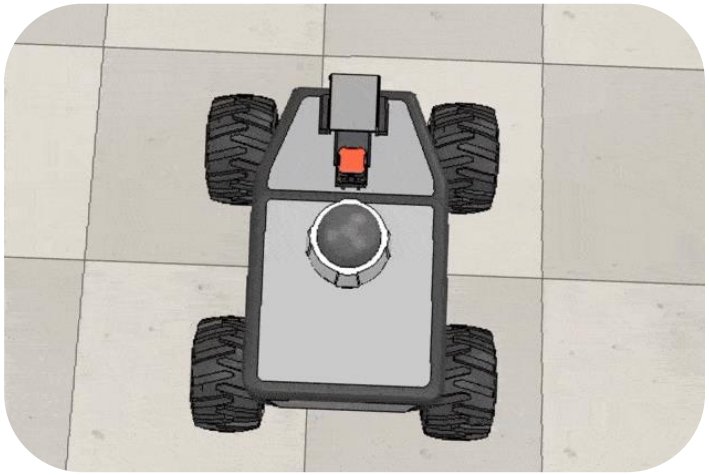
Programación de Summit XL

Ejemplo de parámetros de potencia y sentido para cada rueda.



hreaded child script (Robotnik_Summit_XL)

```lua
function sysCall_threadmain()
    motorHandles={-1,-1,-1,-1}

    motorHandles[1]=sim.getObjectHandle('joint_front_left_wheel')
    motorHandles[2]=sim.getObjectHandle('joint_front_right_wheel')
    motorHandles[3]=sim.getObjectHandle('joint_back_right_wheel')
    motorHandles[4]=sim.getObjectHandle('joint_back_left_wheel')



    while true do

     sim.setJointTargetVelocity(motorHandles[1],1)
     sim.setJointTargetVelocity(motorHandles[2],-1)
     sim.setJointTargetVelocity(motorHandles[3],3)
     sim.setJointTargetVelocity(motorHandles[4],-3)
     sim.wait(1)
    end
end
```

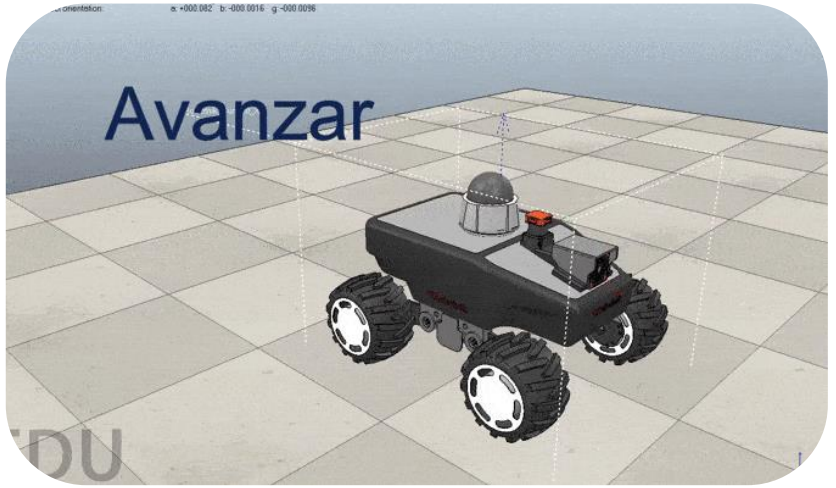


Programación de algoritmos en CoppeliaSim

Programación de Summit XL

Gran cantidad de aplicaciones en robótica funcionan a través de un sistema de Loop o Bucle, lo que permite que el programa del robot posea una autonomía mayor. Esto significa que el algoritmo del robot se vuelve a ejecutar al terminar la última línea de código o comando de acción, volviendo al inicio y ejecutando nuevamente el algoritmo.

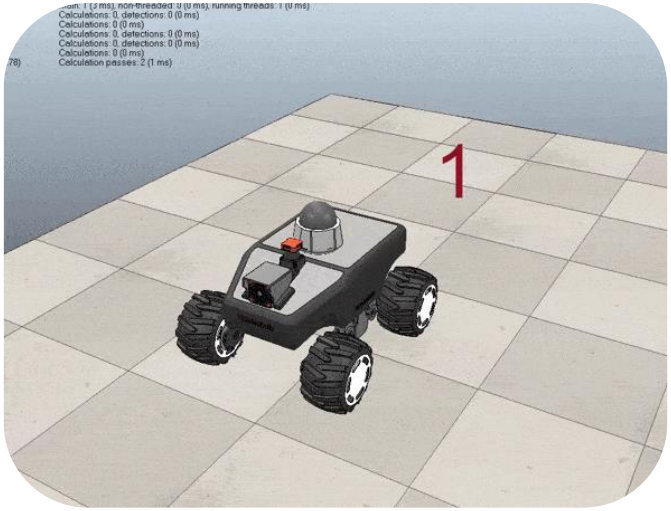
```
function sysCall_threadmain()
    motorHandles={-1,-1,-1,-1}

    motorHandles[1]=sim.getObjectHandle('joint_front_left_wheel')
    motorHandles[2]=sim.getObjectHandle('joint_front_right_wheel')
    motorHandles[3]=sim.getObjectHandle('joint_back_right_wheel')
    motorHandles[4]=sim.getObjectHandle('joint_back_left_wheel')



    while true do

     sim.setJointTargetVelocity(motorHandles[1],1)
     sim.setJointTargetVelocity(motorHandles[2],-1)
     sim.setJointTargetVelocity(motorHandles[3],-1)
     sim.setJointTargetVelocity(motorHandles[4],1)
     sim.wait(1)

     sim.setJointTargetVelocity(motorHandles[1],-1)
     sim.setJointTargetVelocity(motorHandles[2],1)
     sim.setJointTargetVelocity(motorHandles[3],1)
     sim.setJointTargetVelocity(motorHandles[4],-1)
     sim.wait(1)

     end
end
```



Programación de algoritmos en CoppeliaSim

Programación de Summit XL

Para la confección del algoritmo, si se desea realizar un movimiento y luego otro diferente, basta con copiar el algoritmo de movimiento previo y pegarlo como una siguiente acción, luego sencillamente se cambian los parámetros de la acción logrando otro movimiento adicional.




```
while true do

 sim.setJointTargetVelocity(motorHandles[1],1)
 sim.setJointTargetVelocity(motorHandles[2],-1)
 sim.setJointTargetVelocity(motorHandles[3],-1)
 sim.setJointTargetVelocity(motorHandles[4],1)
 sim.wait(1)

 sim.setJointTargetVelocity(motorHandles[1],-1)
 sim.setJointTargetVelocity(motorHandles[2],1)
 sim.setJointTargetVelocity(motorHandles[3],1)
 sim.setJointTargetVelocity(motorHandles[4],-1)
 sim.wait(1)

  sim.setJointTargetVelocity(motorHandles[1],-1)
 sim.setJointTargetVelocity(motorHandles[2],-1)
 sim.setJointTargetVelocity(motorHandles[3],-1)
 sim.setJointTargetVelocity(motorHandles[4],-1)
 sim.wait(1)

  sim.setJointTargetVelocity(motorHandles[1],1)
 sim.setJointTargetVelocity(motorHandles[2],1)
 sim.setJointTargetVelocity(motorHandles[3],1)
 sim.setJointTargetVelocity(motorHandles[4],1)
 sim.wait(1)

end
```

1

2

3

4

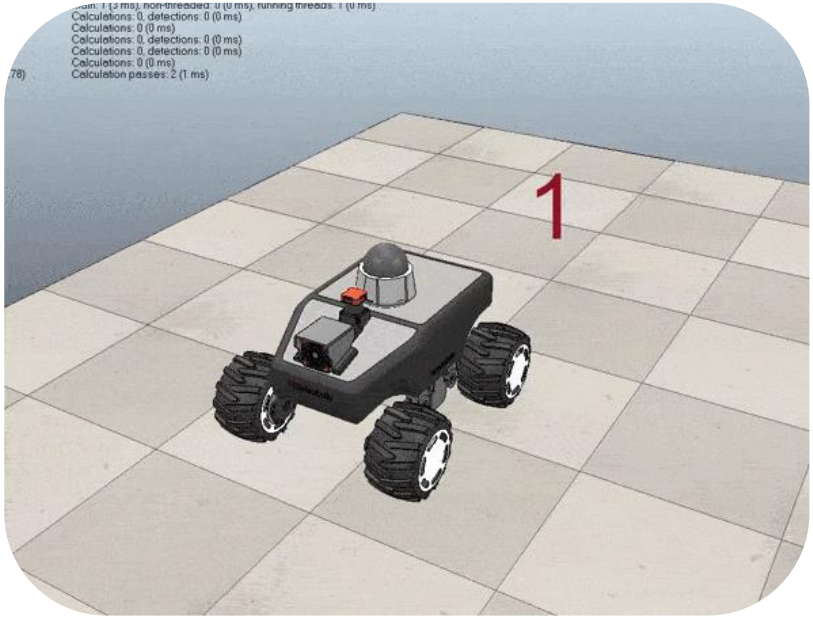
Programación de algoritmos en CoppeliaSim

Actividad de aplicación (Robot diferencial)

Desplazamiento sencillo a través de un robot diferencial de cuatro ruedas.

La intención de esta actividad es generar los cinco movimientos necesarios para poder desplazar el robot a través de una superficie regular.

- Avanzar
- detenerse
- retroceder
- giro horario
- giro antihorario



Actividad de aplicación (Robot diferencial)

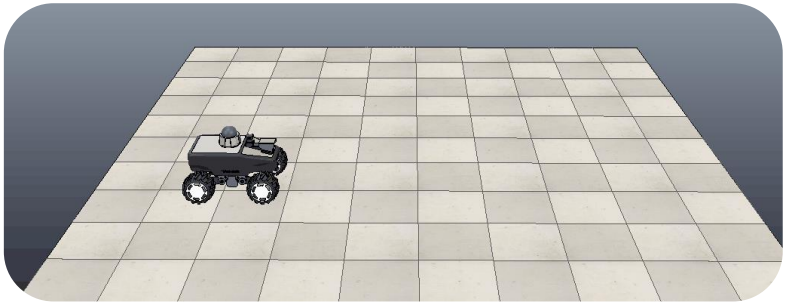
Una vez logrados las cinco acciones básicas de movimiento se realizará un análisis de la precisión de tales movimientos.

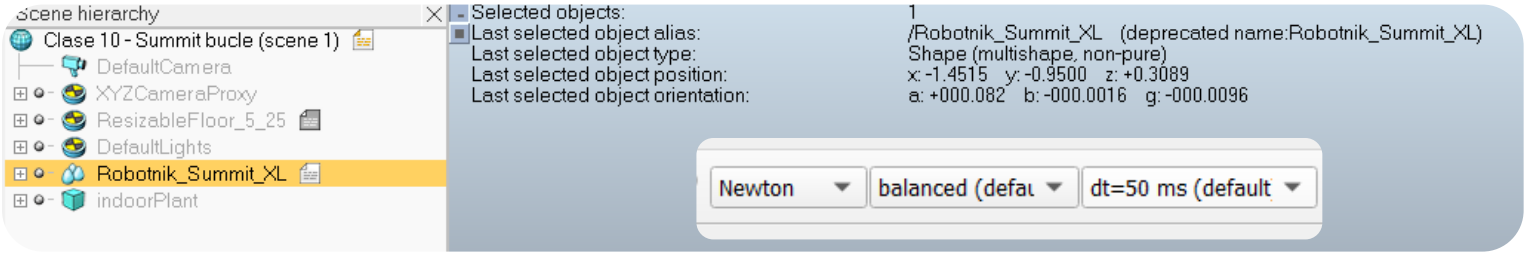
Para lograr este análisis, el robot deberá generar un movimiento sencillo de avanzar aproximadamente cuatro veces su propia longitud para luego volver a su punto de partida. Luego, se deberá ejecutar dos veces más el mismo bucle de movimiento. Finalmente, se deberá analizar el error presente entre las coordenadas iniciales y finales del robot con el objetivo de establecer los parámetros necesarios para que este error sea mínimo.

La coordenada del objeto en la simulación puede visualizarse Al seleccionar el objeto en cuestión obteniendo coordenadas cartesianas de este en el plano.

Realizar la maniobra
en el menor tiempo posible

Actividad de aplicación



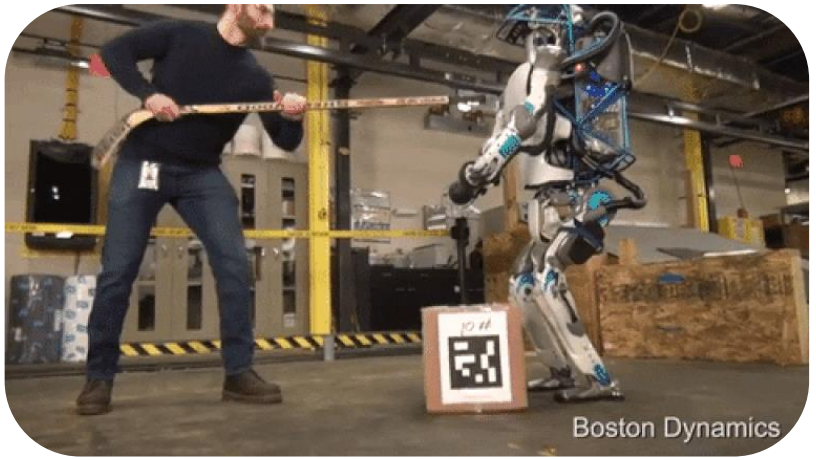


Nivel de autonomía del robot

Un robot que posee una programación básica de movimiento y ejecución de esta en un bucle se considera con una autonomía básica de funcionamiento, es decir, no posee la capacidad de toma de decisiones ni corrección de sus acciones para lograr el objetivo.

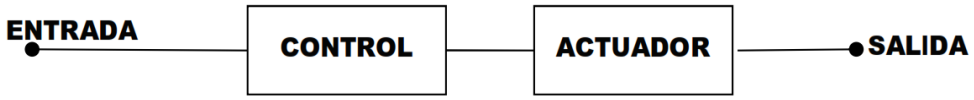
En efecto, un robot con un nivel de autonomía baja no podrá alcanzar su objetivo si es que los parámetros del entorno cambian o si existen perturbaciones que modifiquen su ruta programada.

Como se ha mencionado antes, algunos robots estáticos no requieren de una alta autonomía o capacidad de decisión, ya que su entorno es controlado y se minimizan las perturbaciones que puedan afectar al robot.



Nivel de autonomía del robot

Si un desplazamiento es realizado helio a través de un algoritmo de autonomía baja, se considera que el sistema no posee retroalimentación que permita tomar una decisión para corregir alguna acción y lograr un desplazamiento versátil y eficaz.



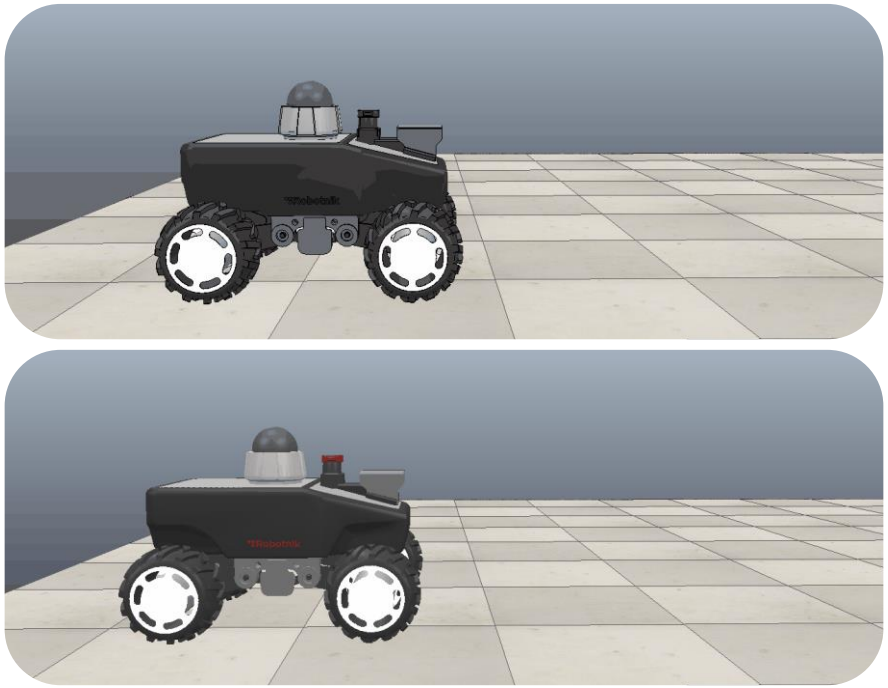
Este sistema es utilizado debido a su sencillez respecto a la implementación y al bajo requerimiento de procesamiento de la máquina robótica.

Luego, para poder implementar este sistema es necesario identificar los errores que pueden aparecer asociados a parámetros de velocidad y desplazamiento (intrínsecos del control).

Errores de desplazamiento

Un error común de desplazamiento se presenta cuando el control de sistema indica una acción de activación de actuadores Indicando velocidades o potencias elevadas sin considerar un proceso de aceleración gradual que permita controlar eficazmente la inercia presente en el sistema robot.

De esta manera se eliminan errores de desplazamiento al intentar sacar del reposo a robot o lograr la detención eficaz de este.



Errores de desplazamiento

Si el código del programa es modificado para que la variable encargada de establecer la potencia a de cada actuador pueda incrementar su valor, gradualmente se podrá lograr un efecto de aceleración, lo que permitirá disminuir el efecto de la inercia en el robot.

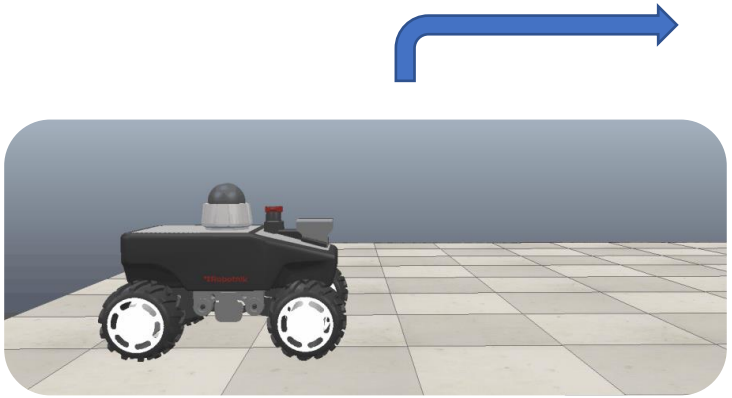
$$a(aceleración) = \frac{\Delta V\ (Cambio\ de\ velocidad)}{t\ (Tiempo)}$$

$$a = \frac{\Delta V}{t}$$

Errores de desplazamiento

Una manera sencilla de implementar una función incremental respecto al movimiento es crear una variable auxiliar que eleve gradualmente la potencia aplicada a los motores.

Evidentemente, existen diferentes métodos de programación que lograrán cumplir el mismo objetivo.



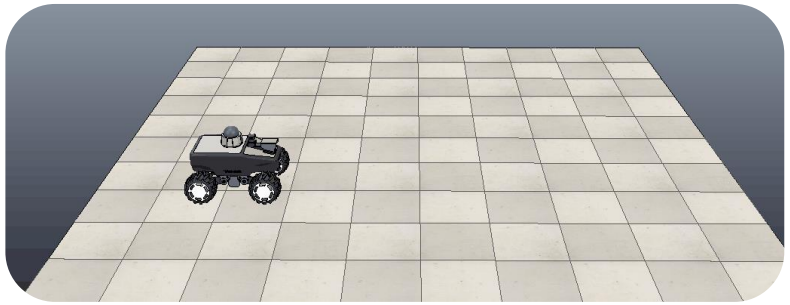
```
while true do
10
11
12      for cont=1,5,1 do --alcance progresivo de la velocidad
13                        --final
14      cont=cont+1
15      sim.setJointTargetVelocity(motorHandles[1],cont)
16      sim.setJointTargetVelocity(motorHandles[2],-cont)
17      sim.setJointTargetVelocity(motorHandles[3],-cont)
18      sim.setJointTargetVelocity(motorHandles[4],cont)
19      sim.wait(0.3)
20
21      end
22
23      sim.setJointTargetVelocity(motorHandles[1],5)
24      sim.setJointTargetVelocity(motorHandles[2],-5)
25      sim.setJointTargetVelocity(motorHandles[3],-5)
26      sim.setJointTargetVelocity(motorHandles[4],5)
27      sim.wait(0.3)
28
29
30
end
```

Actividad de aplicación (Robot diferencial disminución de error de movimiento)

Implementando este sistema de disminución del error de movimiento se debe volver a realizar el ejercicio anterior. Superar un obstáculo de forma sencilla y volver al punto de partida para luego ejecutar dos veces más el mismo bucle de movimiento y posteriormente obtener las coordenadas de posición finales.

Para analizar eficiencia se deberá tomar en cuenta el tiempo requerido para ejecutar la acción, pudiendo cambiar parámetros de aceleración y potencia de motores Con la intención de obtener un mínimo error de posición final con el menor tiempo de ejecución posible.

Realizar la maniobra
en el menor tiempo posible

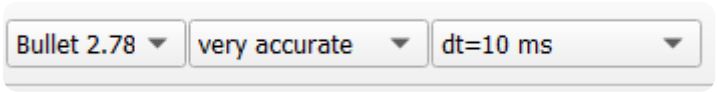


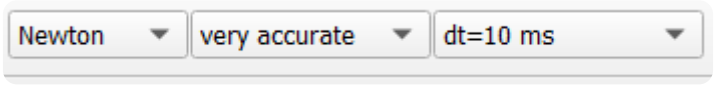
Actividad de aplicación

Error de simulación

CoppeliaSim presenta algunos errores de simulación cuando el motor de cálculo intenta realizar demasiadas iteraciones sobre resultados de movimiento, generando vibraciones que afectan la posición del robot, y por ende la posición de este en la simulación.

Para evitar estos errores, cuando sea necesario, es posible seleccionar entre motores de cálculo físico del simulador, y evitar tales resultados erróneos.





Sin error