Tarea 3 - Simulación de un robot con ruedas
Equipo 1 - Kobuki

Daniel Esteban Molano Garzón
Estudiante de Maestría en Ingeniería - Ingeniería Mecánica
dmolanog@unal.edu.co

Camilo Esteban Zambrano Pereira
Estudiante de Maestría en Automatización Industrial
czambrano@unal.edu.co

Cristhian David Sandoval Diaz
Estudiante de Ingeniería Mecatrónica
csandovald@unal.edu.co

Juan Sebastián Dueñas Salamanca
Estudiante de Ingeniería Mecatrónica
jsduenass@unal.edu.co

15 de marzo de 2024

Universidad Nacional de Colombia - Sede Bogotá
Departamento de Ingeniería Mecánica y Mecatrónica
Fundamentos de Robótica Móvil (2027322)
Profesores: Dr. Ing. Ricardo Emiro Ramírez Heredia - PhD Pedro Fabián Cárdenas
2024-I

Índice

1. Introducción. **3**

2. Objetivos.[1] **3**

3. Herramientas.[1] **3**

4. Desarrollo y análisis de resultados. **4**

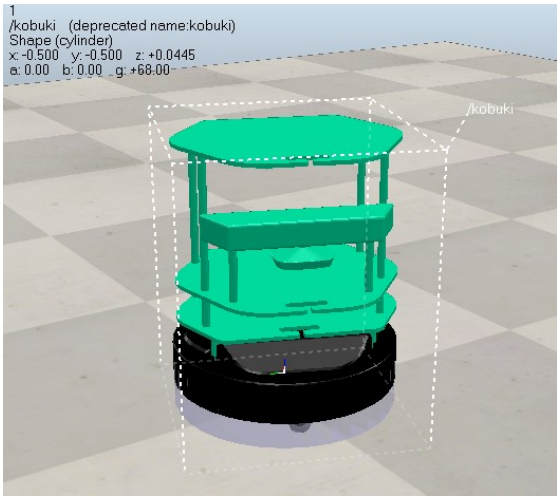
Referencias **7**

Anexos **7**

1. Introducción.

Este laboratorio tenía como objetivo conocer como simular un robot móvil, empleando el software *MATLAB* y el entorno de simulación *CoppeliaSim*. Específicamente se realizaron 2 casos de simulación en el cual el robot móvil debía describir un cuadrado en su trayectoría, para ello se emplearon distintos algoritmos en *MATLAB* los cuales mediante la lógica adecuada permitían la descripción de la trayectoria deseada.

(a) Robot Kobuki.



(b) Robot Kobuki modelado en CoppeliaSim.

Figura 1

A continuación, se observa de manera general la trayectoria que debe seguir el robot durante la ejecución de la simulación para los dos casos de estudio.

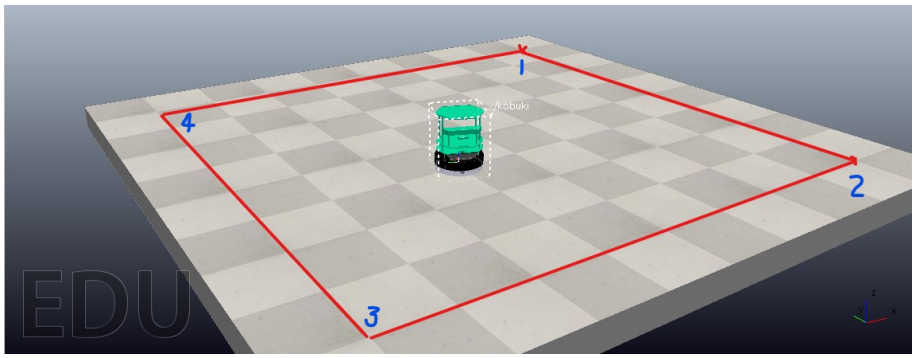


Figura 2: Trayectoria de simulación.

2. Objetivos.[1]

- Familiarizar a los estudiantes con el entorno de simulación *CoppeliaSim*.
- Interactuar con robots controlados desde *MATLAB*.

3. Herramientas.[1]

- *CoppeliaSim Edu*.
- *MATLAB*.

4. Desarrollo y análisis de resultados.

Para este caso el objetivo era generar un script de ***MATLAB*** el cual debía asignar la velocidad de las ruedas y de acuerdo a ello permitir el movimiento del robot ***Kobuki (Turtlebot 2)*** en el entorno de simulación *CoppeliaSim*. Específicamente se requiría completar un cuadrado, al llegar a los vértices del mismo, el robot debía parar por un instante y posteriormente realizar una rotación para continuar el recorrido.
Inicialmente se verificó la conexión y se asignaron los *handles* para el conjunto del robot y cada rueda de tracción diferencial (en este caso 2). A continuación, se detalla la sección del código para realizar dicha acción.

```
%% Programa para el caso 1

%Programa para verificar el uso de handles y desde MATLAB leer variables en Coppelia Sim.
%Establecer la conexión
vrep=remApi('remoteApi'); % usar el archivo prototipo (remoteApiProto.m)
vrep.simxFinish(-1); % si se requiere, cerrar todas las conexiones abiertas.
% asigna el handle de identificación de cliente clientID
clientID=vrep.simxStart('127.0.0.1',19999,true,true,5000,5);
if (clientID>-1)
    disp('Conexión exitosa')
end
%Algoritmo
% Consulta el handle del objeto Caja1 en la escena Esc01 y lo asigna al handle caja_m.
[returnCode,kobuki_m]=vrep.simxGetObjectHandle(clientID,'kobuki',vrep.simx_opmode_blocking);

% Ruedas
[returnCode,wR]=vrep.simxGetObjectHandle(clientID,'jointR',vrep.simx_opmode_blocking);
[returnCode,wL]=vrep.simxGetObjectHandle(clientID,'jointL',vrep.simx_opmode_blocking);
```

Posteriormente, se establecieron las condiciones de operación del robot, así como su posición y orientación inicial. Para ambos casos, los docentes asignaron al equipo de trabajo 1, como posición de partida $(2, 2)$ con orientación $-\mathbf{y}$.

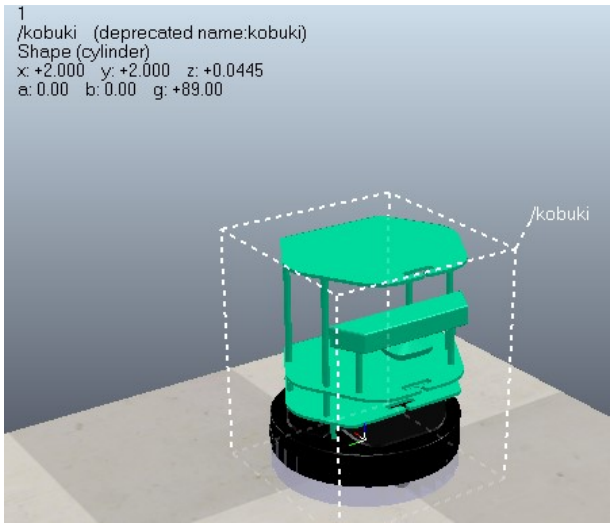


Figura 3: Kobuki en su posición y orientación inicial.

```
r= 0.0684/2;                    %Radio de las ruedas motrices [m]
l= 0.1025;                      %Distancia del centroide del robot a la rueda [m]
```

```
vel_ang = 10;                       %Velocidad angular de la rueda 1  [rad/s]
vel_lin = vel_ang*r;                %Velocidad lineal del robot[m/s]
x = 4;                              %Distancia a recorrer [m]
d_duration = x/vel_lin;             %Duración del recorrido [s]

% Calculo de la velocidad angular de las ruedas a partir del jacobiano del
% sistema y la velocidad angular objetivo
vel_rotation = [1/r l/r;1/r -l/r]*[0; pi/2];

vrep.simxGetPingTime(wR)
vrep.simxGetPingTime(wL)
% Se establece la posición y orientación inicial del robot
Pos_ini=[2,2,0.0445];
Or_ini=[0,0,pi/2];
[returnCode]= vrep.simxSetObjectPosition(clientID, kobuki_m,-1,Pos_ini, ...
vrep.simx_opmode_blocking);
[returnCode]=vrep.simxSetObjectOrientation(clientID,kobuki_m,-1,Or_ini, ...
vrep.simx_opmode_blocking);
```

Finalmente, se empleó un bucle *for* para generar la trayectoria de acuerdo a la velocidad y el tiempo estimado.

```
for c = 1:4
    vrep.simxSetJointTargetVelocity(clientID, wL, vel_ang, vrep.simx_opmode_oneshot);
    vrep.simxSetJointTargetVelocity(clientID, wR, vel_ang, vrep.simx_opmode_oneshot);
    pause(d_duration);
    vrep.simxSetJointTargetVelocity(clientID, wL, 0, vrep.simx_opmode_oneshot);
    vrep.simxSetJointTargetVelocity(clientID, wR, 0, vrep.simx_opmode_oneshot);
    pause(1)
    vrep.simxSetJointTargetVelocity(clientID, wL, vel_rotation(1), vrep.simx_opmode_oneshot);
    vrep.simxSetJointTargetVelocity(clientID, wR, vel_rotation(2), vrep.simx_opmode_oneshot);
    pause(1)
    vrep.simxSetJointTargetVelocity(clientID, wL, 0, vrep.simx_opmode_oneshot);
    vrep.simxSetJointTargetVelocity(clientID, wR, 0, vrep.simx_opmode_oneshot);
    pause(1)
end
vrep.simxSetJointTargetVelocity(clientID, wR, 0, vrep.simx_opmode_oneshot)
vrep.simxSetJointTargetVelocity(clientID, wL, 0, vrep.simx_opmode_oneshot)
```

Algo a recalcar es que fue necesario, cambiar la expresión `vrep.simx_opmode_blocking` por la expresión `vrep.simx_opmode_oneshot`, ello puesto que inicialmente una de las ruedas de acuerdo al orden de programación secuencial se activaba primero, por lo que al tener este desfase en el arranque se iba a generar una desviación en la trayectoria. La expresión `vrep.simx_opmode_oneshot` permitía asignar la velocidad de las ruedas sin la necesidad de esperar que **CoppeliaSim** enviara una respuesta ante la activación de cada rueda.

Para el segundo programa se utilizo una estructura muy parecida a la primera en especial en cuanto a conexión pero se desarrollaron 2 funciones para facilitar le proceso. Una que permite obtener la posición del robot y una segunda que permite mandar comandos y de velocidad lineal y angular transmitirlo a velocidad de rueda que debe ejecutar el robot.

```
function vel_wheels = move_robot(vrep,clientID,vel_lineal,vel_angular)
    r= 0.0684/2;            %Radio de las ruedas motrices [m]
```

```
    L= 0.1025;              %Distancia del centroide del robot a la rueda [m]
    %Velocidad angular de la rueda 1 [rad/s]
    %Velocidad lineal del robot[m/s]
    vel_wheels = [-1/r L/r; -1/r -L/r]*[vel_lineal; vel_angular];

    % Ruedas
    [returnCode_R,wR]=vrep.simxGetObjectHandle(clientID,'jointR',vrep.simx_opmode_blocking);
    [returnCode_L,wL]=vrep.simxGetObjectHandle(clientID,'jointL',vrep.simx_opmode_blocking);

    vrep.simxSetJointTargetVelocity(clientID, wR, vel_wheels(1), vrep.simx_opmode_oneshot);
    vrep.simxSetJointTargetVelocity(clientID, wL, vel_wheels(2), vrep.simx_opmode_oneshot);
end

function [pos,orient] = get_robot_pose(vrep,clientID,robot_id, world_id)
    [returnCode, orient] = vrep.simxGetObjectOrientation(clientID, robot_id, world_id,...
    vrep.simx_opmode_blocking);
    [returnCode, pos] = vrep.simxGetObjectPosition(clientID, robot_id,...
    world_id, vrep.simx_opmode_blocking);
end
```

El siguiente paso es incorporar estas funciones en un ciclo de control que permita dirigir el robot, Para cada uno de los vértices objetivo calculamos un error de orientación. Siendo el ángulo objetivo el ángulo formado entre la coordenada del bot (calculada utilizando la función get pose) y el vértice objetivo. A continuación, usamos esa medida de error para realizar un control proporcional y definir una velocidad rotacional la cual se ejecuta y se va actualizando hasta alcanzar el ángulo objetivo con un margen de error previamente establecido. Una vez el bot esta orientado puede moverse de manera lineal hasta alcanzar el vértice objetivo sin embargo se añade el termino de velocidad angular para que funcione como un compensador y ayude a orientar al robot al momento de presentarse desviaciones respecto a la posición objetivo.

```
for i=1:4

    pose(end+1,:) = [pos(1:2),orient(3)];;

    vertice_objetivo = vertices(i+1,:);

    error_orient = orient_objetivo - orient(3);

    while  abs(error_orient) > error_min(2)
        orient_objetivo = atan2(vertice_objetivo(2) - pos(2), vertice_objetivo(1) - pos(1));
        [pos, orient] = get_pose();
        error_orient = orient_objetivo - orient(3);
        vel_wheels = move_robot(vrep,clientID, 0, error_orient);
        log_error(end+1,:) = [ (vertice_objetivo(1:2) - pos(1:2)), error_orient];
        pose(end+1,:) = [pos(1:2),orient(3)];

        message = "robot orient "+(orient(3)*180/pi) + " target orient " +...
        (orient_objetivo*180/pi);
        message = message + " error orient " + (error_orient*180/pi);
        disp(message)
        disp(vel_wheels)
```

```
    %pause(1)

end

error_pos = norm(vertice_objetivo(1:2) - pos(1:2));

while  abs(error_pos) > error_min(1)
    [pos, orient] = get_pose();
    orient_objetivo = atan2(vertice_objetivo(2) - pos(2), vertice_objetivo(1) - pos(1));
    error_orient = orient_objetivo - orient(3);
    delta_pos = vertice_objetivo(1:2) - pos(1:2);
    error_pos = norm(vertice_objetivo(1:2)-pos(1:2));
    log_error(end+1,:) =[ vertice_objetivo(1:2) - pos(1:2), error_orient];
    vel_wheels = move_robot(vrep,clientID, 0.5*error_pos , 0.3*error_orient);
    pose(end+1,:) = [pos(1:2),orient(3)];
    message =sprintf('pos [%2.3f %2.3f] target pos [%2.3f %2.3f] error pos [%2.3f %2.3f]',.
    pos(1),pos(2), vertice_objetivo(1), vertice_objetivo(1), delta_pos(1),delta_pos(2));

    disp(message)
    disp(vel_wheels)
end
```

El código fuente puede ser encontrado en: https://github.com/mobile-robotics-unal/matlab-coppelia-simulation.

Referencias

[1] R. E. Ramirez. P. F. Cardenas. Guía: Conexión entre CoppeliaSim y Matlab. Curso FRM, Universidad Nacional de Colombia,2024.

[2] CoppeliaRobotics. Legacy remote API functions (Matlab). `https://manual.coppeliarobotics.com/`.

[3] Corke, P. I., Jachimczyk, W., & Pillat, R. (2011). Robotics, vision and control: fundamental algorithms in MATLAB (Vol. 73, p. 2). Berlin: Springer.

[4] Siegwart, R., Nourbakhsh, I. R., Scaramuzza, D. (2011). Introduction to autonomous mobile robots. MIT press.