**Universidad Politécnica de Valencia**

**Escuela Técnica Superior de Ingeniería Informática**

**Grado en Ingeniería Informática**

**Asignatura:**

Mecatrónica

**Asignación:**

Práctica No.6

**Profesor:**

Alejandro Vignoni

**Integrantes:**

Almengor, Alexander

Silgo, Juan José.

**Grupo:**

PL-1\_OB1

**Fecha:**

18 de abril del 2021

**Tabla de Contenido**

[Memoria de la Práctica No. 6 3](#_Toc69923523)

[Parte A | Simulink 3](#_Toc69923524)

[1. Ejercicio No.1 3](#_Toc69923525)

[2. Ejercicio No.2 3](#_Toc69923526)

[3. Ejercicio No.3 4](#_Toc69923527)

[4. Ejercicio No.4 5](#_Toc69923528)

[Curva de Bezier y Spline Cúbica Natural para 3pts 5](#_Toc69923529)

[Curva de Bezier y Spline Cúbica Natural para 4pts 5](#_Toc69923530)

[5. Ejercicio No.5 6](#_Toc69923531)

[a) PtosControlA [0 0; 1000 500;0 1000]; %mm Total\_t = 4; %Segundos 6](#_Toc69923532)

[Bezier 6](#_Toc69923533)

[Spline 6](#_Toc69923534)

[b) PtosControlB=[200 -200; 800 1000; 1500 500; 0 600 ]; %mm Total\_t = 20; %Segundos 7](#_Toc69923535)

[Bezier 7](#_Toc69923536)

[Spline 7](#_Toc69923537)

[c) PtosControlC= [3 3;3 6;5 5;3 5]; mm Total\_t = 20; %Segundos (Banderín de Golf); 8](#_Toc69923538)

[Bezier 8](#_Toc69923539)

[Spline 8](#_Toc69923540)

[Parte B | CoppeliaSim 10](#_Toc69923541)

[1. Algoritmo del Punto descentrado en Matlab 10](#_Toc69923542)

[2. Simulación en CoppeliaSim 11](#_Toc69923543)

[3. Gráficos de Trayectorias 13](#_Toc69923544)

[4. Error Cuadrático Medio de Seguimiento 14](#_Toc69923545)

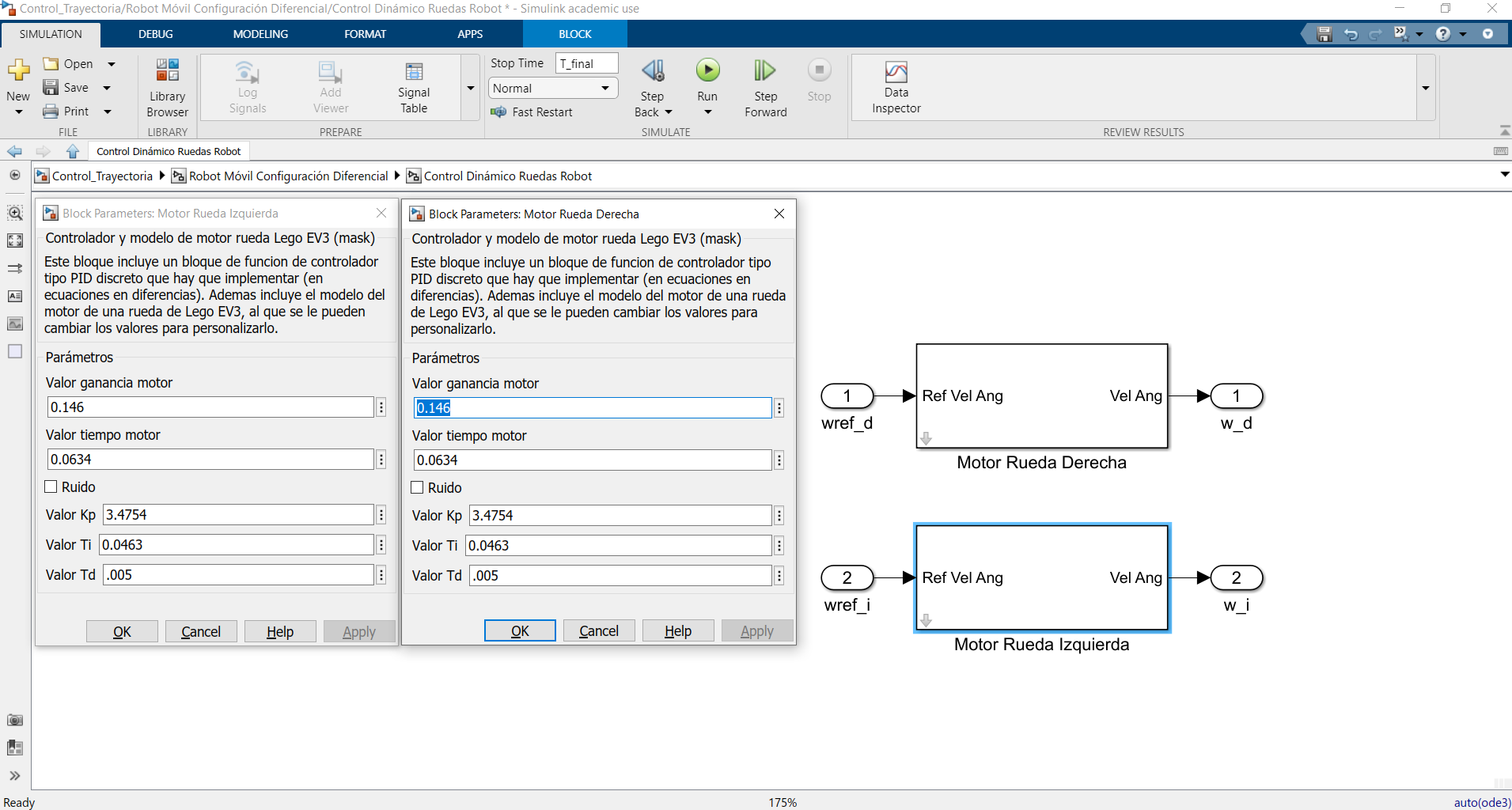
# Memoria de la Práctica No. 6

## Parte A | Simulink

### Ejercicio No.1

Integración del control PID de velocidad angular del motor del Lego EV3 (Práctica 5) en el modelo cinemático de un robot con configuración diferencial (creado en la Practica 2).

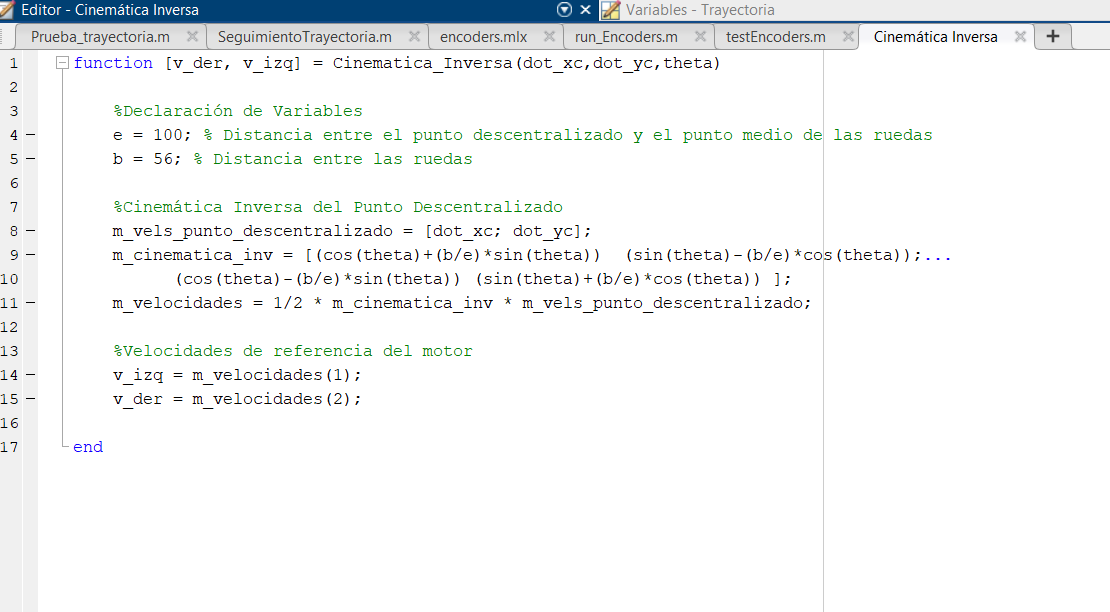
Se agregaron los parámetros obtenidos en la práctica 4 y 5 respectivamente al modelo cinemático del robot.



**Parámetros de Ambos Motores**

### Ejercicio No.2

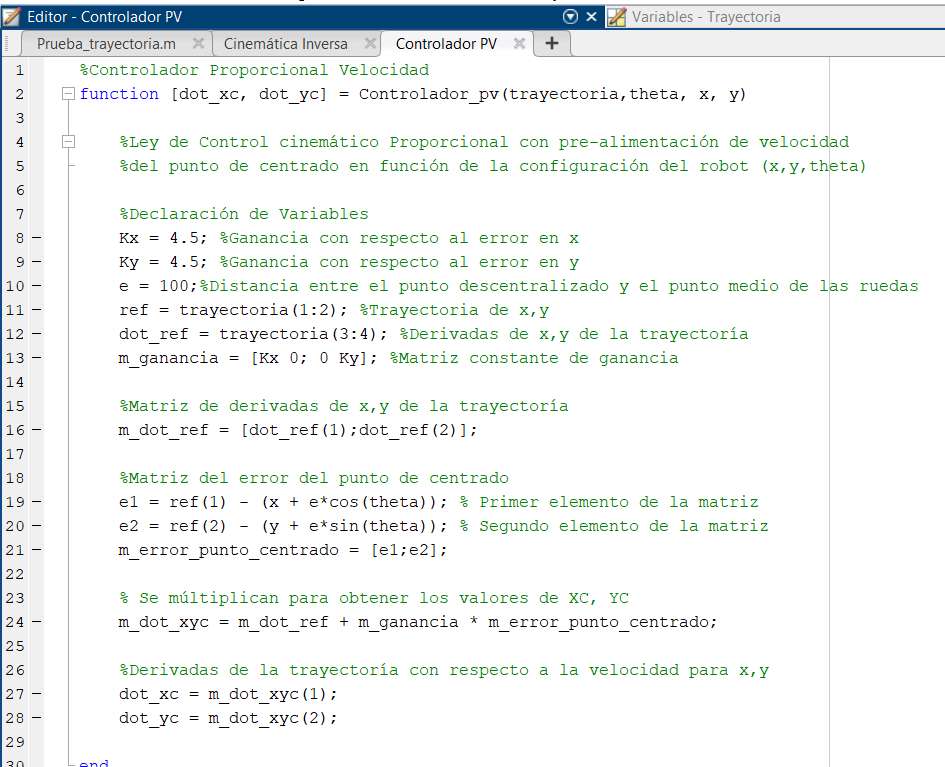
Implementación de la cinemática inversa de punto descentralizado (𝑥ሶ𝑐 , 𝑦ሶ𝑐) como bloque generador de las referencias de velocidad de las ruedas del robot (𝑣𝐿 , 𝑣𝑅).



### Ejercicio No.3

Implementación del control de seguimiento de trayectoria tipo proporcional con velocidad (PV) para generar las derivadas de la posición del punto descentralizado (𝑥ሶ𝑐 , 𝑦ሶ𝑐) a partir de la configuración actual del robot 𝑥, 𝑦, 𝜃 y de la trayectoria de la referencia (𝑥𝑟𝑒𝑓, 𝑦𝑟𝑒𝑓) y sus derivadas (𝑥ሶ𝑟𝑒𝑓, 𝑦ሶ𝑟𝑒𝑓).

Se decidió implementar este segmento del circuito en una función de Matlab, la cual replica el comportamiento de la ecuación de la Ley de control cinemático proporcional con pre-alimentación de velocidad del punto descentrado, en función de la configuración del robot (x, y, theta).

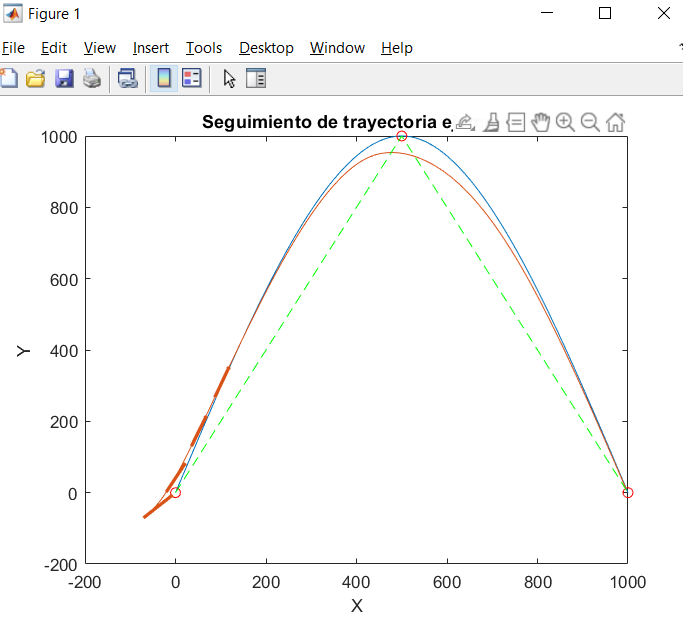


### Ejercicio No.4

Generación de la trayectoria de la referencia (𝑥𝑟𝑒𝑓, 𝑦𝑟𝑒𝑓) y sus derivadas (𝑥ሶ𝑟𝑒𝑓, 𝑦ሶ𝑟𝑒𝑓) utilizando las funciones de la practica 1, cambiando el tiempo real.

Para el desarrollo de este ejercicio se reutilizó las funciones de la práctica No.1 de Curva de Bezier y la Spline Cúbica Natural, tanto para tres puntos y cuatro puntos, se utilizo como puntos de control los brindados en la práctica.

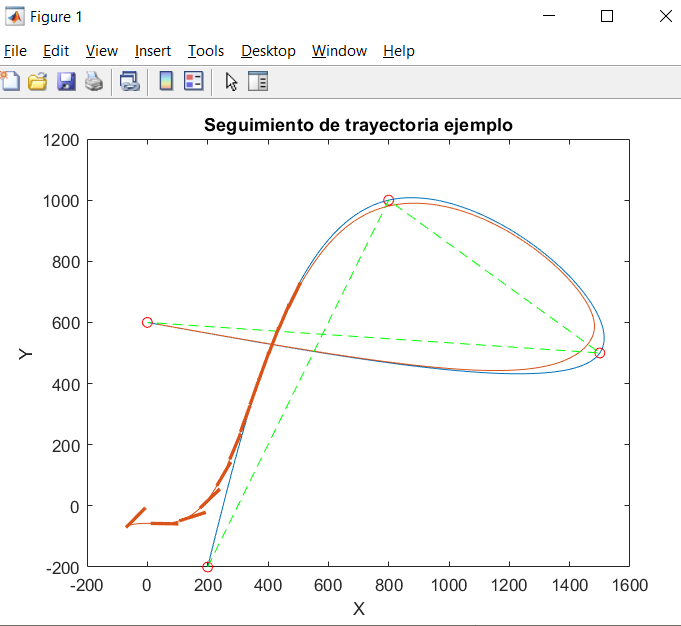
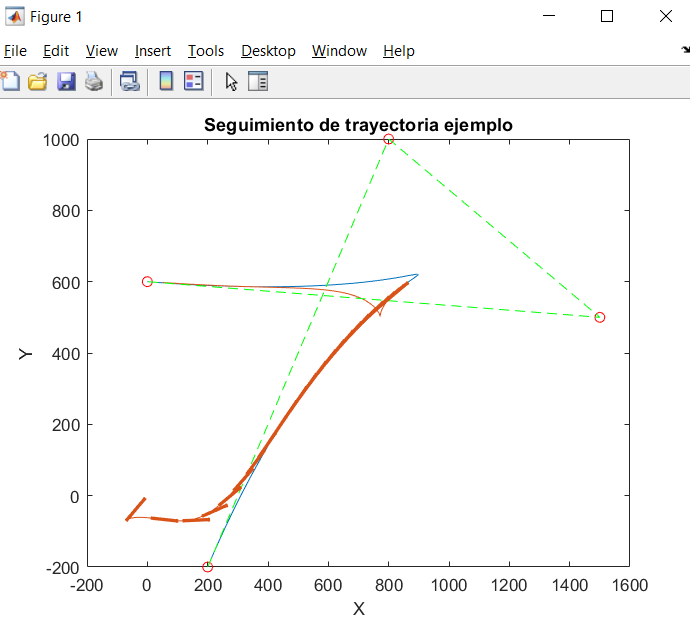
#### Curva de Bezier y Spline Cúbica Natural para 3pts



**Spline**

**Bezier**

#### Curva de Bezier y Spline Cúbica Natural para 4pts



**Spline**

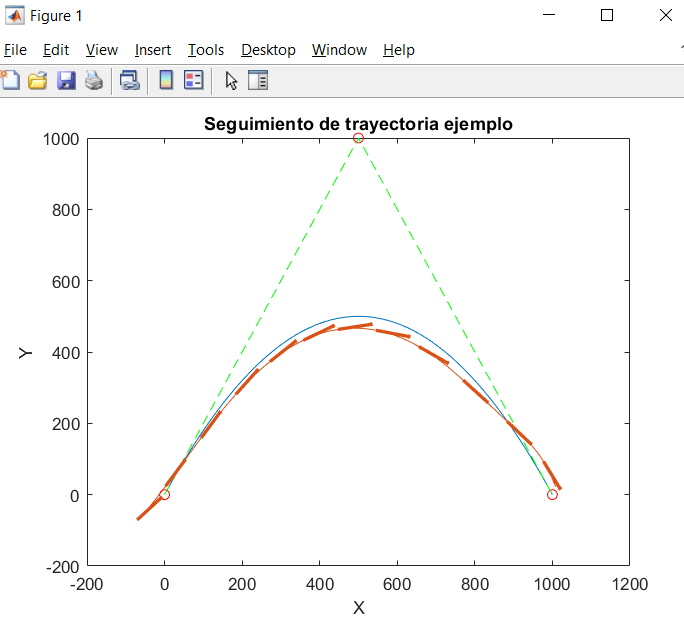
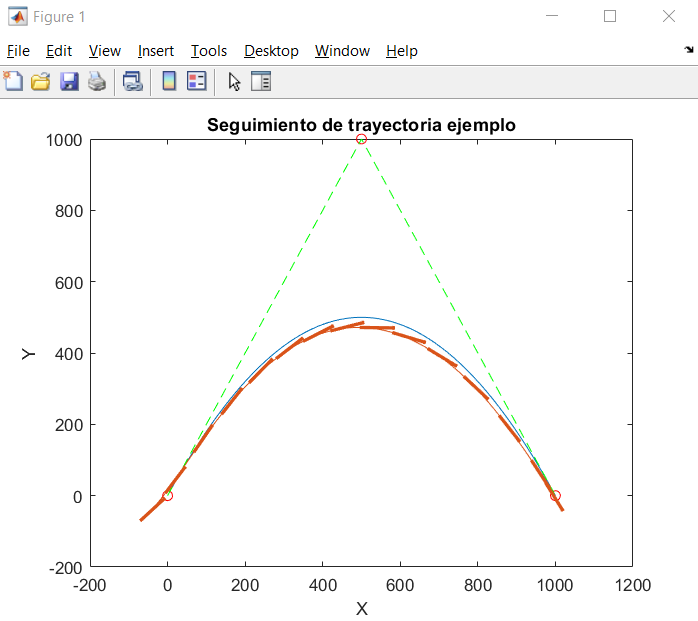
**Bezier**

### Ejercicio No.5

Seguimiento de trayectoria de los puntos propuestos: Realizar un script que defina los puntos de partida y llame a la función correspondiente para obtener las curvas de aproximación. Luego llame a la ejecución del simulink. Calcula el Error (como función del tiempo) y el ITAE de la trayectoria obtenida. Incluye la representación gráfica de los puntos de partida y las curvas generadas

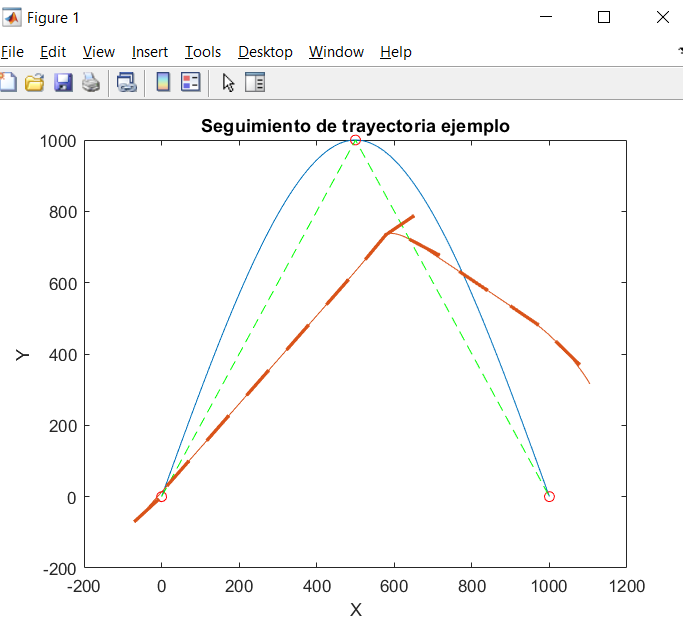
#### PtosControlA [0 0; 1000 500;0 1000]; %mm Total\_t = 4; %Segundos

##### Bezier



En el gráfico izquierdo podemos ver que al realizar la prueba utilizando el algoritmo de Bezier para tres puntos, empleando 4 segundos se pierde precisión al final del trayecto; sin embargo, si aumentamos el tiempo en un segundo, es decir 5 segundos y utilizando una ganancia de 4.5 para los valores de las variables kx y ky, se puede mejorar la precisión.

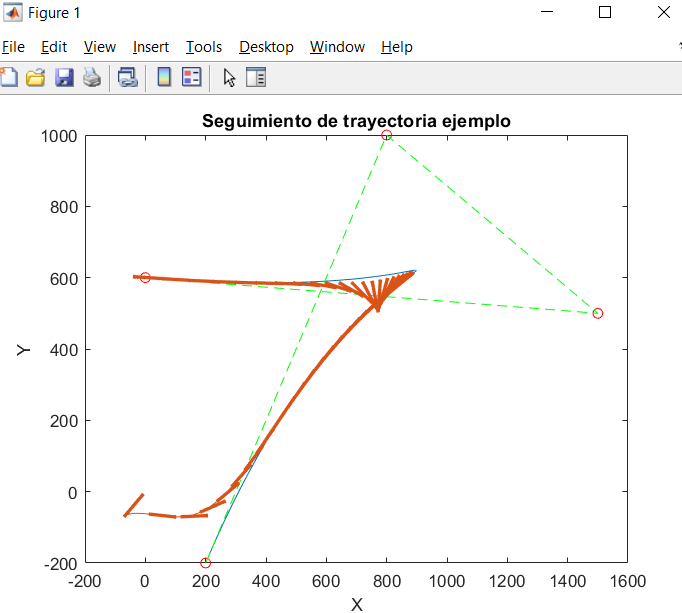
##### Spline

En el caso del cálculo de las trayectorias utilizando Spline, podemos observar que para un tiempo de 4 segundos y una ganancia de 4.5 para las variables kx y ky, el seguimiento de la trayectoria perdía precisión; sin embargo, al ajustar los parámetros del tiempo a 8 segundos, es decir el doble, se obtuvo un mejor resultado más preciso.

#### PtosControlB=[200 -200; 800 1000; 1500 500; 0 600 ]; %mm Total\_t = 20; %Segundos

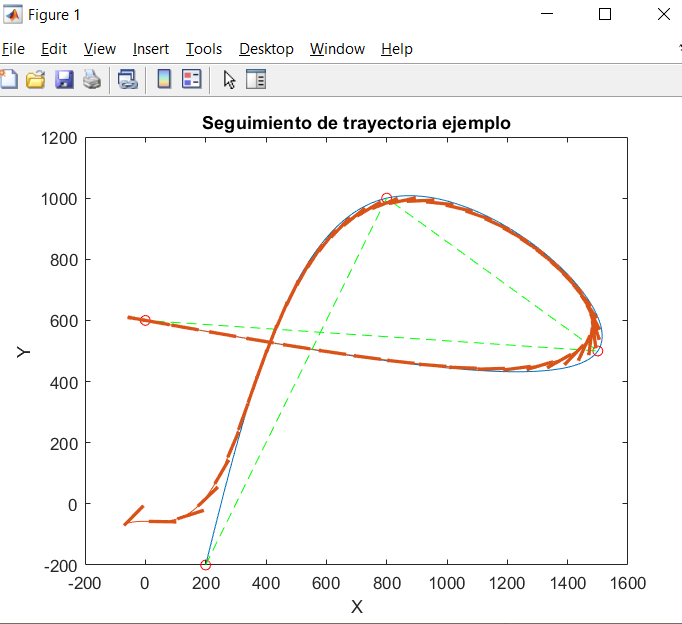
##### Bezier

Al realizar el seguimiento de trayectoria utilizando el algoritmo de Bezier para cuatro puntos, y una ganancia para las variables kx y ky de 5, obtuvimos un seguimiento de la trayectoria bastante preciso, esto también se debe a que el tiempo es de 20 segundos, a mayor tiempo mejor la precisión.



##### Spline

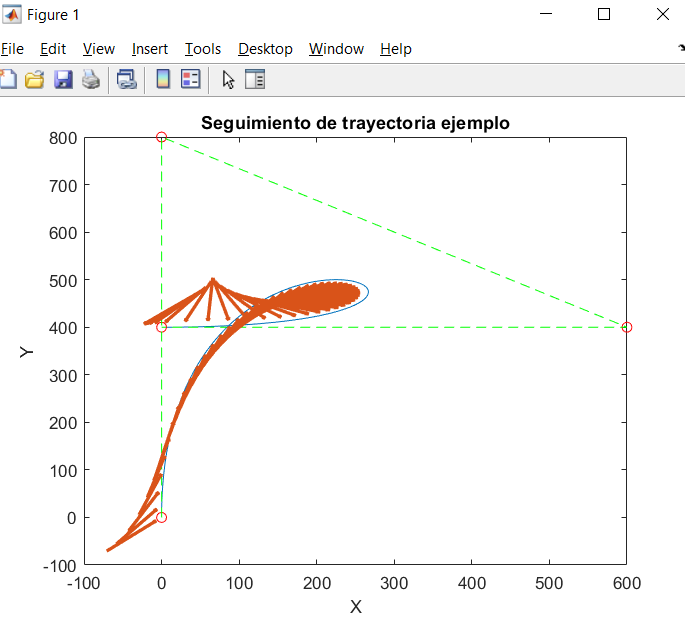
Utilizando el algoritmo de Spline para cuatro puntos obtuvimos un seguimiento de trayectoria preciso.



#### PtosControlC= [3 3;3 6;5 5;3 5]; mm Total\_t = 20; %Segundos (Banderín de Golf);

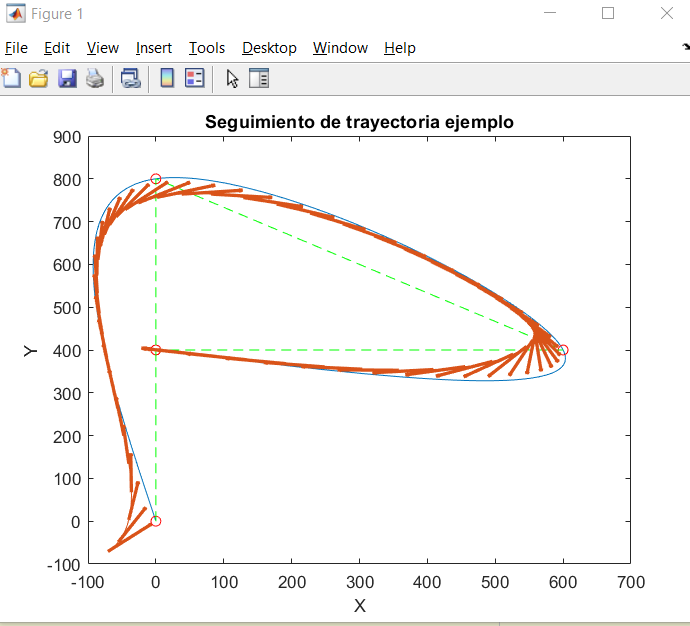
##### Bezier

Para la prueba realizada con Bezier intentado seguir una trayectoria en forma de banderín de Golf, los resultados nos muestran que a 20 segundos con una ganancia en las variables kx y ky de 3, la trayectoria seguida se aproxima bastante al último punto, pero como el comportamiento natural de Bezier es de una aproximación, se puede ver que el recorrido termina antes.

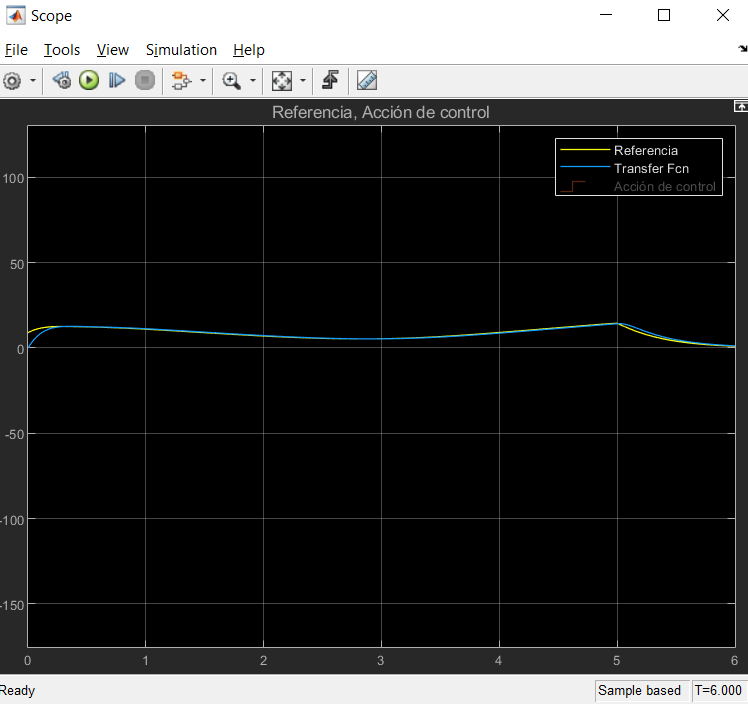


##### Spline

Al repetir el mismo ejercicio con los mismos parámetros salvo que los valores de las variables kx y ky, fueron de 4.5, y empleando un algoritmo de generación de Trayectorias de movimiento como Spline, el recorrido fue mucho más preciso y se recorre por completo la figura.



### ITAE



Como se puede observar en la imagen el robot apenas tiene un retardo inicial en llegar a la referencia, manteniéndose a una distancia despreciable de ella durante el resto de la trayectoria. Con esto concluimos que, aunque se precisaría de un cálculo exhaustivo o una medición dedicada para visualizar correctamente este valor, el ITAE es mínimo en este caso, entendiendo que es aceptable.

## Parte B | CoppeliaSim

### Algoritmo del Punto descentrado en Matlab

Implementar al algoritmo del punto descentrado en Matlab siguiendo el esqueleto del archivo SeguimientoTrayectoria.m completando las líneas correspondientes.

Se implementó el algoritmo de punto descentrado en Matlab.

%% Seguimiento trayectoria punto descentralizado

% obtenemos el valor de los encoders (grados) para ver cuanto se ha movido

posruedaDerecha = double(MotorRotationCount(OUT\_C));

posruedaIzquierda = double(MotorRotationCount(OUT\_A));

% calculamos las velocidades angulares (rad/s) de las ruedas // CONVERTIR A RADIANES

wdk= (posruedaDerecha - posruedaDerecha1)\*(pi/180)\*(1/Ts);

wik= (posruedaIzquierda- posruedaIzquierda1)\*(pi/180)\*(1/Ts);%derivada Euler

% calculamos las velocidades lineales (mm/s) de las ruedas: v = w\*radio

vdk= wdk \* radiorueda;

vik= wik \* radiorueda;

% calculamos la velocidad lineal del robot (mm/s)

vk =(vdk+vik)/2; %promedio

% calculamos la velocidad angular del robot (rad/s)

wk = (vdk-vik)/(2\*b);

% estimamos la posicion X-Y (mm) y la orientacion del robot (rad)

x=x + vk\*Ts\*cos(theta);

y=y+vk\*Ts\*sin(theta);

theta=theta+wk\*Ts;

% calculamos la velocidad del punto descentralizado a partir del control cinematico del robot (mm/s) %pag 18

velxp= velxref + kx \* (xref-(x+e\*cos(theta)));

velyp= velyref + ky \* (yref-(y+e\*sin(theta)));

% calculamos las velocidades lineales de la ruedas que debera aplicar el robot a partir del modelo cinematico inverso del robot (mm/s) %pag17

vi= (((e\*cos(theta)+b\*sin(theta))\*velxp)+((e\*sin(theta)-b\*cos(theta))\*velyp))\*(1/e);

vd= (((e\*cos(theta)-b\*sin(theta))\*velxp)+((e\*sin(theta)+b\*cos(theta))\*velyp))\*(1/e);

% calculamos las velocidades angulares de referencia para el control dinamico (rad/s)

wref\_d= vd/radiorueda;

wref\_i= vi/radiorueda;

% mandamos las acciones de control a aplicar a cada rueda (en % actuación)

OnFwd(OUT\_C,((100\*wref\_d)/18.32)); % motor derecha

OnFwd(OUT\_A,((100\*wref\_i)/18.32)); % motor izquierda

% almacenamos los valores de los encoder para la proxima iteracion

posruedaDerecha1 = posruedaDerecha;

posruedaIzquierda1 = posruedaIzquierda;

% iteración

i=i+1;

% calculo indice integral error cuadratico

ex= xref - x;

ey= yref - y;

errorcua= errorcua + (ex \* ex+ ey \* ey);

%Guardar datos

datos=[datos;xref yref x y errorcua];

t2=double(CurrentTick());

% Espera hasta el siguiente periodo

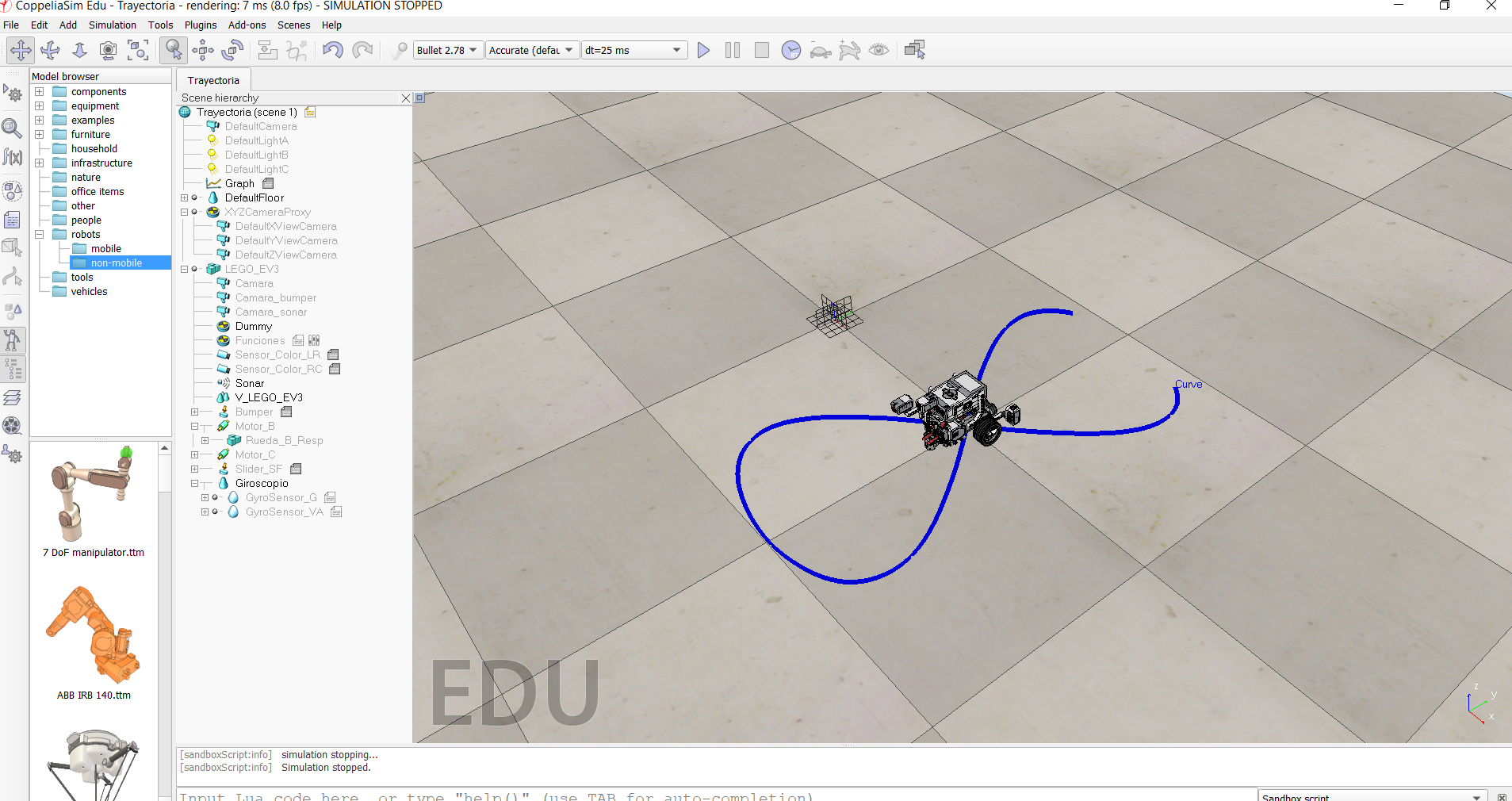
espera=(max(0,Ts\*1000-(t2-t1)));

Wait(espera);

### Simulación en CoppeliaSim

Simular en CoppeliaSim utilizando la escena Trayectoria.ttt y el código de comienzo StartSeguimientoTrayectoria.m

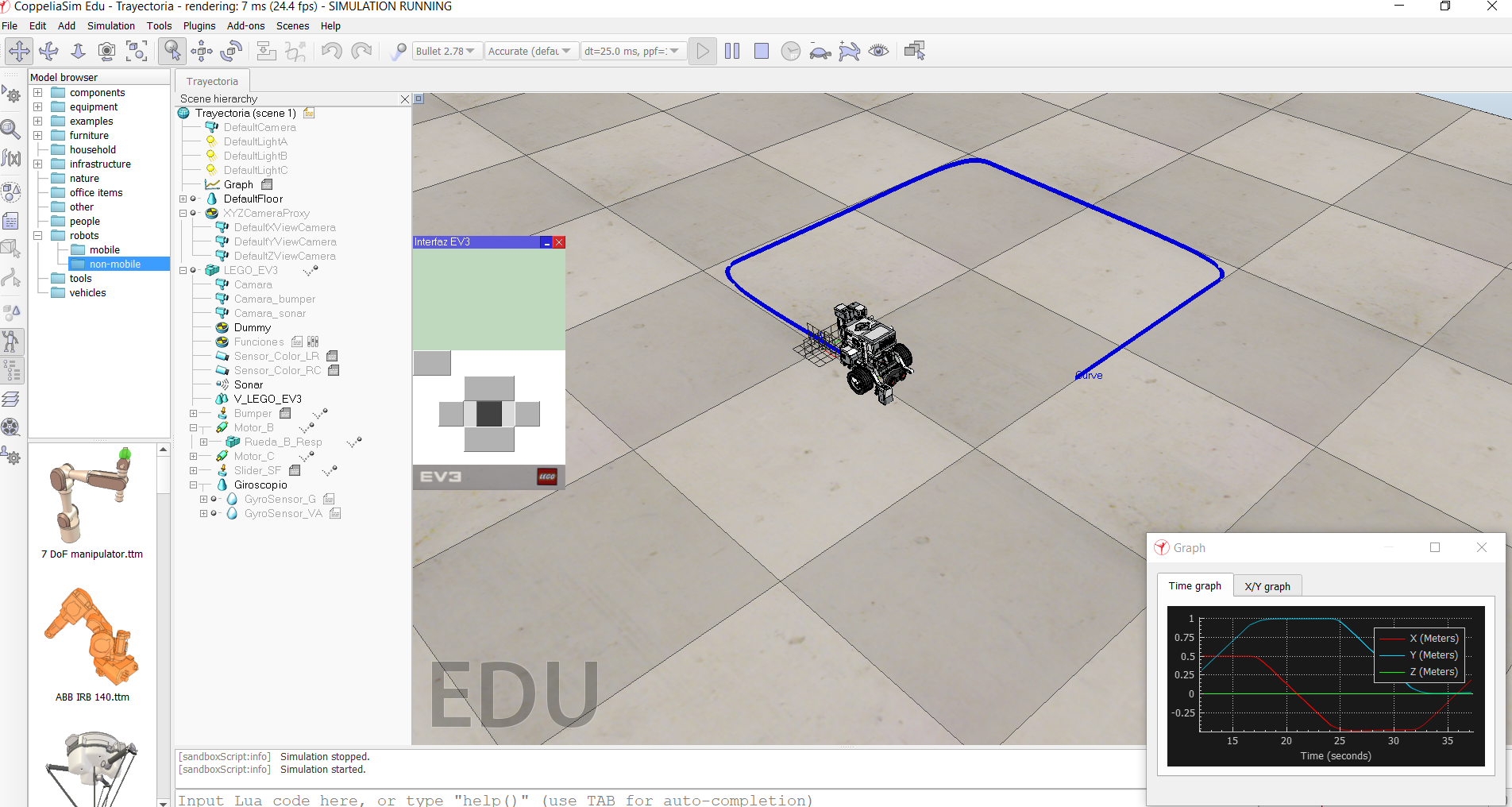
#### 2.1 Trayectoria Infinito



#### 2.2 Trayectoria Circular



#### 2.3 Trayectoria Cuadrada

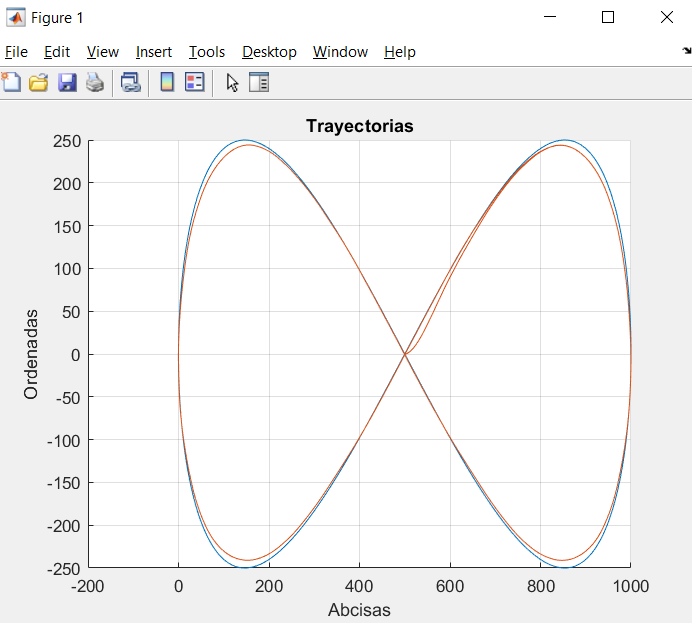


### Gráficos de Trayectorias

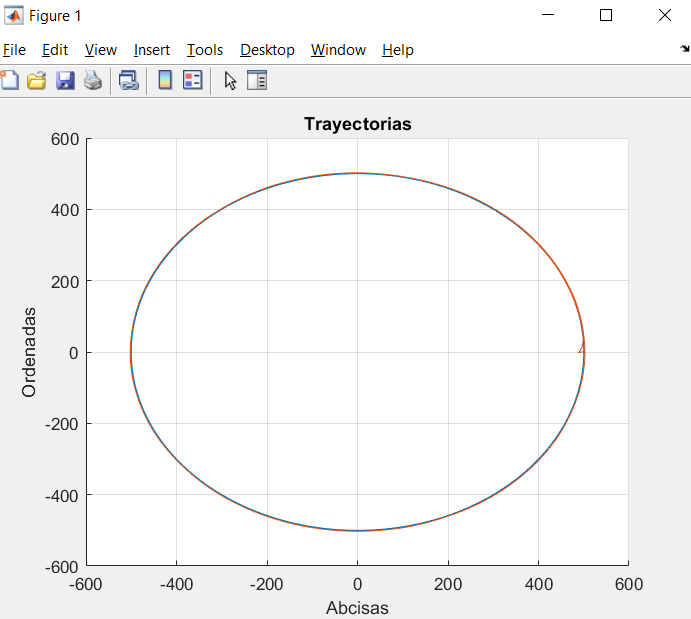
Graficar en la misma gráfica la trayectoria deseada del robot y la obtenida por odometría.

Se muestra en los gráficos la trayectoria de referencia representada con la línea roja, y con la azul la obtenida por el robot mediante odometría.

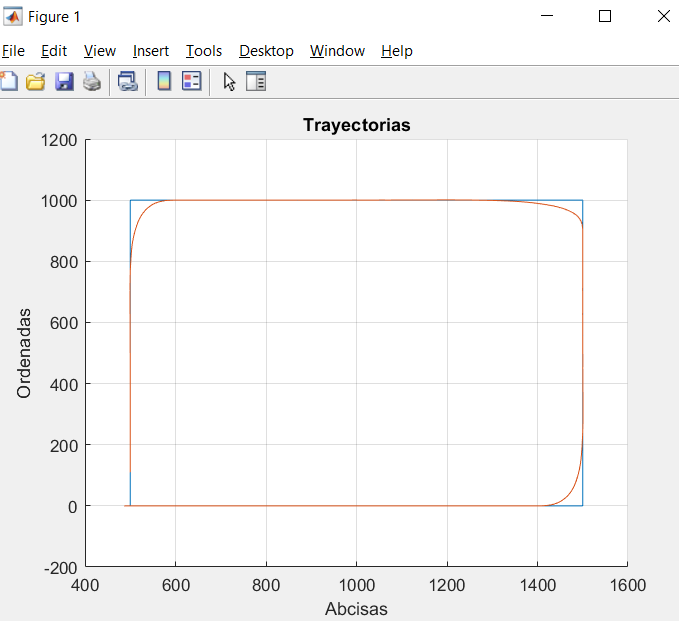
#### 3.1 Trayectoria de Infinito



#### 3.2 Trayectoria Circular



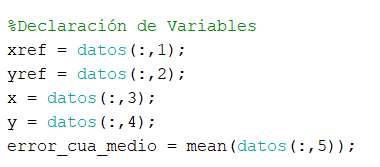
#### 3.3 Trayectoria Cuadrada



### Error Cuadrático Medio de Seguimiento

Calcular el error cuadrático medio del seguimiento de trayectoria obtenido

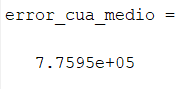
Al desarrollar la práctica se cálculo el error cuadrático medio que utiliza la función mean, indexando los datos recopilados de la matriz datos en la columna número cinco.



#### 4.1 Trayectoria de Infinito



#### 4.2 Trayectoria Circular



#### 4.3 Trayectoria Cuadrada

