# Memoria de la asignatura de Robótica 2021/2022

Logotipo

Descripción generada automáticamente

Luis Gutiérrez Jerez

## Movimiento de la cabeza y primeros contactos

En la primera clase, realizamos un acercamiento al lenguaje de programación Matlab y algunas de sus funciones útiles que serán detalladas a continuación:

* Plot(var1,var2): Crea una representación gráfica (en este caso en 2 dimensiones) de los datos en var2(eje Y) frente a los valores de var1 (eje X).
* Tic: Guarda un identificador temporal para su posterior toc
* Toc(var1): Devuelve el tiempo que ha traspasado tras el tiempo determinado mediante el identificador temporal (que deberá de estar en var1)

La explicación se basó en la importancia de las smooth function (funciones continuas derivables a un punto) en los ámbitos de la robótica para evitar movimientos bruscos mecánicos.

Hemos realizado el proyecto de mostrar gráficamente la función sen(X) usando una función auxiliar, a continuación el código:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | k=**1**;  **for** s=**0**:**0.01**:**2**\*pi    t(k) = s;  y(k) = signal\_v0(t(k));  k=k+**1**;    **end**  %plot(t,y)  tstart = tic;  segundos = **10**;  t(k) = **0**;  k = **1**;  **while** (t(k) < segundos)  k=k+**1**;  t(k) = toc(tstart);  y(k) = signal\_v0(t(k));    **end**  plot(t,y) |

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | **function** salida = **signal\_v0**(t)  %UNTITLED2 Summary of this function goes here  % Detailed explanation goes here  salida = sin(t);  **end** |

Para la siguiente clase también seguimos con el mismo propósito de la anterior.

A partir de la tercera clase se comenzó a trabajar con el objetivo de animar la gráfica sen(x) para simular como se comportaría la cabeza del robot, tras un intento fallido en dicha clase, se consiguió animar en la clase 4 junto a una modificación de la función auxiliar anterior( ahora pasaremos a llamarle signal\_vf), adjunto el código fuente:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39 | clear all  clc  delay = **4**;  periodo = **6**;  amplitud = **1**;  k = **1**;  **for** s=**0**:**0.01**:**4**\*pi    t(k)=s;  y(k)=signal\_vf(t(k),delay,periodo,amplitud);    k = k+**1**;  **end**  k = **1**;  tstart = tic;  segundos = **12**;  tiempo(k) = **0**;  **while** tiempo(k) < segundos  y2(k) = signal\_vf(tiempo(k),delay,periodo,amplitud);  cla  plot(t,y)  hold on    plot(tiempo(k),y2(k),'Or')  k = k + **1**;  tiempo(k) = toc(tstart);    axis([**0** **12** -**1** **1**])    drawnow    **end**  plot(t,y) |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | **function** salida = **signal\_vf**(t, delay,periodo,amplitud)  %UNTITLED Summary of this function goes here  % Detailed explanation goes here  salida = amplitud \* sin((**2**\*pi/periodo)\* (t-delay));  **if** t < delay  salida = **0**;  **else** **if** t > delay+periodo  salida = **0**  **end**    **end** |

En la clase 6 se nos dio un acercamiento a las funciones de Matlab que interactúan directamente con el robot, con el objetivo de realizar movimientos en las distintas partes de este. Algunas de estas funciones son:

* Legoev3(‘USB’): Permite conectar el robot y sincronizarlo con el script de Matlab.
* touchSensor(NombreConexion,NPuerto): Devuelve un objeto del sensor de toque conectado en el puerto NPuerto del robot NombreConexion (Tenemos que tener en cuenta que puede haber distintos robots en un mismo sistema, por esto es necesario tener una identificación de cada uno de ellos).
* sonicSensor(NombreRobot): Devuelve un objeto del sensor de proximidad situado en la cabeza del robot, notar que, no es necesario identificar el nº del puerto debido a que al solo haber un sensor de este tipo conectado en el robot, este lo identifica automáticamente.
* Motor(NombreConexion,NPuerto): Nos devuelve un objeto con el que podremos accionar los motores correspondiente al NPuerto indicado configurando posteriormente la velocidad.
* objetoMotor.Speed: Nos permite configurar el valor de la velocidad de giro del motor.
* objetoMotor.start(): Nos permite accionar el motor.
* readDistance(objetoSonicSensor): Devuelve la distancia medida por el objeto SonicSensor
* readTouch(objetoTouchSensor): Devuelve un booleano que será true si el pulsador ha sido accionado.

A continuación se muestra el código fuente:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | WallE=legoev3('USB');  PulsadorArranque = touchSensor(WallE,**4**);  Sonar = sonicSensor(WallE);  motorDerecho = motor(WallE,'D');  motorIzquierdo = motor(WallE,'A');  start(motorDerecho);  motorDerecho.Speed = **50**;  distancia = readDistance(Sonar);  Pulsacion = readTouch(PulsadorArranque); |
|  |  |

Para la sexta clase, realizamos un sistema de representación tridimensional para representar la cabeza del robot, antes de mostrar el código es necesario algunas explicaciones conceptuales y matemáticas:

configuracion de objetos rigidos: magnitudes que si las conozco me permiten establecer o

%conocer las coordenadas de cada uno de los puntos que componen el cuerpo

%respecto un eje de referencia

DOF : degree of freedom : tanta libertad como variables de configuracion

%puedo cambiar

La lógica matemática que vamos a usar van a ser las matrices y las matrices de transformación (o cambio de base)

Calse teórica 04/11/2021



El sonar nos proporciona una medida directa de lo que hay en frente(d)



Tenemos que saber donde está el objeto detectado con respecto el eje coordenadas global del sistema



Conozco la orientación del robot y sus coordenadas con respecto a R, junto a la posición de la cabeza respecto a las coordenadas de R y conoce el angulo alfa



Tenemos la matriz de transformación



Y

La Xc coincide con el haz sonico que permite medir la distancia, por tanto la coordenada del punto del objeto detectado y el punto de referencia de la cabeza (cXo,xYo)=[d 0 0 1]

(la x es la distancia, y e z son 0 y siempre tenemos que poner la ultima columna en 1].

Multiplico la rTc por la matriz [d 0 0 1]’ , esto representa la rXo,rYo,rZo, si multiplicamos esto por la matriz de transformación de gTr quedando gTr\*rTc\*[d 0 0 1]’ = [gXo,gYo,gZo,1]’

Clase teórica nº 5:

En esta clase teórica vimos como representar en un prisma en 3 dimensiones con el objetivo de posteriores implementaciones para la visualización del robot.

Primero voy a introducir los conceptos teóricos necesarios:

Partimos del siguiente problema: Teniendo unas coordenadas globales , en las cuales se encuentra nuestra robot, necesitamos controlar a nuestro robot, el cual, en sí mismo, tiene otro sistema de referencia diferente a nuestras coordenadas globales.

El objetivo de esto es encontrar la configuración de los objetos rígidos(conjunto de magnitudes que si son conocidas, podemos establecer o conocer las coordenadas de cada uno de los puntos que componen el cuerpo respecto a un sistema de referencia).



Donde G es el sistema de representación global , Omega es el ángulo y L el sistema de representación propio del robot ( tenemos que tener en cuenta que, en caso de nuestro robot, la propia cabeza del robot sería otro sistema de representación, cosa que resolveremos en clases posteriores).

Otro concepto interesante es el DOF (degree of freedom) que se define como la cantidad de variables de configuración modificables para un cuerpo.

En el ejemplo anterior, la matriz de cambio que resolvería el problema anterior sería:



Nótese que el último termino de ambas P es 1 debido a la dimensión de T(4\*4) por lo cual es necesario que P sea 4\*1

Vamos a resolver un problema similar pero con dos estructuras como ejemplo:

Se nos muestra un punto de una figura A (P1) donde es necesario pasarlo al sistema de referencia global del sistema para hallar las coordenadas del punto.



A continuación nos piden que realicemos la siguiente actividad: Representar un punto en el punto central superior de un prisma.

El prisma creado por la función pinta\_bloque en este caso creará un prisma de las siguientes dimensiones, donde tendremos que dibujar un punto en el centro de la cara superior hallando sus coordenadas:



A continuación, vamos a crear un nuevo prisma, rotarlo 180º en el eje Y y 45º en el eje Z.

Para ello definimos una matriz de transformación que contenga, a su vez, distintas matrices con cada uno de las transformaciones deseadas, primeramente, crearemos una matriz de translacción que mueva el objeto a otra posición (para evitar que se solapen), después, las siguientes dos matrices serán respectivamente, cada uno de las rotaciones de los ejes de coordenadas.

matrizFinal**=**makehgtform**(**'translate'**,**5**,**5**,**5**)\***makehgtform**(**'xrotate'**,**pi**/**2**)\***makehgtform**(**'zrotate'**,**pi**/**4**);**

El resultado de esto nos queda:

Gráfico

Descripción generada automáticamente

A continuación, se nos anima a dibujar los puntos de los vértices superiores de este nuevo prisma

La mejor forma de hacer esta tarea es definir las coordenadas locales del objeto, y tras esto, aplicar la misma matriz de transformación que la usada previamente para rotar dicho prisma.

Pcentral\_G **=** **[**0 0 6 1**]'** %vector columna

Pcentral\_L **=** matrizFinal **\*** Pcentral\_G

plot3**(**5**,-**1**,**5**,**'\*b'**)**

Aquí observamos el punto central de dicho prisma rotado, el resto de puntos de las aristas se realizan de la misma forma, por lo que por motivos de limpieza, voy a obviar, a continuación se muestra el código:

pinta\_bloque**(**eye**(**4**,**4**),**'r'**)**

%Nos permite representar en 3d una matriz.

%configuracion de objetos rigidos: conjunto de magnitudes que si las

%conozco me permite establecer o conocer las coordenadas de cada uno de los

%puntos que componen el cuerpo respecto a un sistema de referencia.

plot3**(**1.5**,**1.5**,**6**,**'\*g'**)**

plot3**(-**1.5**,**1.5**,**6**,**'\*g'**)**

plot3**(**1.5**,-**1.5**,**6**,**'\*g'**)**

plot3**(-**1.5**,-**1.5**,**6**,**'\*g'**)**

%makehgtform me permite hacer una matriz de transformación

pinta\_bloque**(**makehgtform**(**'translate'**,**5**,**5**,**5**)** **\*** makehgtform**(**'xrotate'**,**pi**/**2**)** **\*** makehgtform**(**'zrotate'**,**pi**/**4**),**'r'**)**

%Me Con este comando he movido y rotado el bloque

%Ahora quiero hayar los punto sdel objeto girado:

matriz\_\_final **=** makehgtform**(**'translate'**,**5**,**5**,**5**)** **\*** makehgtform**(**'xrotate'**,**pi**/**2**)** **\*** makehgtform**(**'zrotate'**,**pi**/**4**);**

p\_central\_L **=** **[**0 0 6 1**]';**

p\_central\_G**=**matriz\_\_final **\***p\_central\_L

P1 **=** **[**1.5 1.5 6 1**]'**

P1\_L **=** matriz\_\_final **\*** P1

plot3**(**5**,-**1**,**7.1213**,**'\*b'**)**

P2 **=** **[-**1.5**,** 1.5**,** 6 **,**1**]'**

P2\_L **=** matriz\_\_final **\*** P2

plot3**(**2.8787**,-**1**,**5**,**'\*b'**)**

P3 **=** **[-**1.5**,** **-**1.5**,** 6 **,**1**]'**

P3\_L **=** matriz\_\_final **\*** P3

plot3**(**5**,-**1**,**2.8787**,**'\*b'**)**

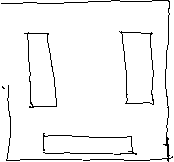
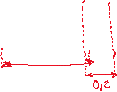
P4 **=** **[**1.5**,** **-**1.5**,** 6 **,**1**]'**

P4\_L **=** matriz\_\_final **\*** P4

plot3**(**7.1213**,-**1**,**5**,**'\*b'**)**

Tras esta clase, se nos asignó ver y realizar una actividad online la cual consta de realizar el script de pinta\_robot versión 2, donde pintaremos una representación en dos dimensiones para simular el comportamiento del robot.

A continuación se muestran las medidas de la representación (estas serán alteradas en un futuro para aplicarle medidas reales).



A continuación, voy a explicar los diferentes comandos usados para realizar la representación:

* makehgtform-> Nos permite realizar operaciones y representar matrices de transformación, en el primer parámetro, ajustamos la operación que deseamos realizar, en el segundo, introducimos una matriz con el tamaño del rectángulo deseado siendo [coord.PuntoIzquierdoX coord.PuntoIzquierdoY longitudAlto longitudAncho] .
* rectangle -> Nos permite crear rectángulos en una coordenada.

**function** pinta\_robot\_v1\_2**(**x**,**y**,**theta**,**alfa**,**SR\_robot**,**SR\_cabeza**,**distancia**)**

%planteamos un rectangulo en un espacio concreto

R **=** rectangle**(**'Position'**,[-**1.5 **-**1.5 3 3**],** 'Parent'**,**SR\_robot**);**

%posicionamos al robot

M **=** makehgtform**(**'translate'**,** **[**x y 0**],** 'zrotate'**,** theta**);**

SR\_robot**.**Matrix **=** M**;**

%definamos las ruedas del robot

rueda\_size **=** **[-**0.5 **-**0.1 1 0.2**];**

%rueda derecha

SR\_rueda\_derecha **=** hgtransform**(**'Parent'**,** SR\_robot**);**

rueda\_derecha **=** rectangle**(**'Position'**,** rueda\_size**,** 'Parent'**,**SR\_rueda\_derecha**);**

M **=** makehgtform**(**'translate'**,[**0 **-**1 0**]);**

SR\_rueda\_derecha**.**Matrix **=** M**;**

%rueda izquierda

SR\_rueda\_izquierda **=** hgtransform**(**'Parent'**,**SR\_robot**);**

rueda\_izquierda **=** rectangle**(**'Position'**,**rueda\_size**,**'Parent'**,**SR\_rueda\_izquierda**);**

M **=** makehgtform**(**'translate'**,[**0 1 0**]);**

SR\_rueda\_izquierda**.**Matrix **=** M**;**

cabeza\_size **=** **[-**0.25 **-**0.5 0.5 1**];**

cabeza **=** rectangle**(**'Position'**,** cabeza\_size**,**'Parent'**,** SR\_cabeza**);**

M **=** makehgtform**(**'translate'**,** **[**1 0 0**],** 'zrotate'**,** alfa**);**

SR\_cabeza**.**Matrix **=** M**;**

axis**([-**10 10 **-**10 10**])**

Mt **=** SR\_robot**.**Matrix **\*** SR\_cabeza**.**Matrix**;**

punto **=** Mt**\*[**distancia 0 0 1**]';**

animatedline**(**punto**(**1**),** punto**(**2**),** 'Marker'**,** '\*'**,** 'LineStyle'**,** 'none'**);**

**end**

El funcionamiento es simple, definimos el robot como un rectángulo (el rectángulo R), al cual le aplicamos una translación y rotación mediante la matriz de transformación M, tras ser asignada, obtenemos el rectángulo del contorno del robot girado y colocado según los parámetros x, y y theta.

Tras esto, los elementos siguen todos el mismo esquema, primero, se crea el objeto con respecto al eje de coordenadas local y se colocan las medidas, tras esto, se aplica la matriz de transformación con la opción ‘Parent’ la cual permite ser hija de la matriz del robot, tras esto, solo tenemos que definir el rectángulo colocando en los parámetros las opciones anteriormente mencionadas.

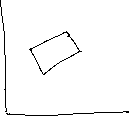
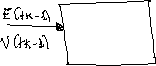
En la clase 7 vamos a usar lo aprendido anteriormente para simular y predecir al robot para posteriormente aplicarlo a los robots de laboratorio.

Existen dos modelos posibles:

* Dinámico: Para la simulación es necesario considerar el momento y masa del cuerpo a partir de la fuerza, el par, el momento de inercia y masa, determinando la aceleración y velocidad
* Cinemático: Es un modelo simplificado donde sencillamente podemos describir la velocidad directamente sin necesidad de describir todas las magnitudes antes descritas para el modelo dinámico

Por las características de esta asignatura, elegiremos el modelo cinemático por su simpleza.

El modelo deseado es el siguiente:



Queremos un modelo donde introduciendo el estado en instante k (E(tk-1)), el estado en el instante k (Estad(tk-1)) y la salida será X(tk). El estado será el conjunto de variables que me permitan representar el sistema globalmente.

En el robot, deberíamos conocer las posiciones del robot (x,y), el angulo que forma con el eje X del sistema global (K) y el ángulo de la cabeza.

Las variables de entrada serán la velocidad y la velocidad de giro deseadas.

Para describir el movimiento del robot respecto al tiempo, es necesario explicar las derivadas necesarias para describir esto.

Considerando que:

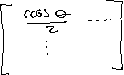


Buscamos una función cuya entrada sea la situación actual del estado (lo llamaremos x) y el valor de entrada (lo llamaremos e) y nos permita predecir los datos a futuro de tal forma que si estamos en un tiempo T(k-1) nos prediga T(k).



El método de integración de Euler es un método de primer orden, por lo cual el error local es proporcional al cuadrado del tamaño del tiempo, por lo cual, es efectivo siempre y cuando dt tenga tamaños reducidos.

Existen otros métodos como el método trapeziodal o Runge-Kutta que reduce bastante el error en comparación con el método de Euler.



De esto podemos sacar que:



clear all

clc

tstart=tic;

i=1;

t(i)=toc(tstart);

%condiciones inciales

encoder1(i)=0;

encoder2(i)=0;

x(i)=0;

y(i)=0;

theta(i)=0;

mapa=[];

SR\_robot = hgtransform;

SR\_cabeza = hgtransform('Parent',SR\_robot);

global radio\_rueda

global l %distancia entre centro y rueda

radio\_rueda=3;

l=1;

while t(i)<5

i=i+1;

t(i)=toc(tstart);

%velocidades de las ruedas.

w1=1;

w2=0;

%------------------------------------------------

%Los angulos de los encoders tenen que estar en radianes

%----------------------------------------------------

%simulamos la evoluciÃ³n de los encoders

encoder1(i)=w1\*t(i);

encoder2(i)=w2\*t(i);

%simulamos la odometrÃ­a

[x(i) y(i) theta(i)]=calculo\_odometria(encoder1,encoder2,x,y,theta,i);

%Dibujamos la simulaciÃ³n

pinta\_robot\_v1\_2(x(i),y(i),theta(i),0,SR\_robot,SR\_cabeza,15);

drawnow

end

plot(t,encoder1,t,encoder2)



La función calculo\_odometría usa las formulas anteriormente descritas para que, introduciendo los datos de giro, ángulo, posición etc. nos devuelva los valores del modelo cinemático.

global radio\_rueda

global l %distancia entre ruedas

%calculo del incremento del angulo de rodado

dalfa1=giro\_A(i)-giro\_A(i-1);

dalfa2=giro\_C(i)-giro\_C(i-1);

%Cálculo del ángulo

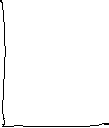
theta\_new=theta(i-1)+radio\_rueda\*(dalfa1-dalfa2)/(2\*l);

x\_new=x(i-1)+((radio\_rueda\*(dalfa1+dalfa2)/2)\*cos((theta\_new+theta(i-1))/2));

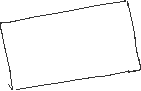
y\_new=y(i-1)+((radio\_rueda\*(dalfa1+dalfa2)/2)\*sin((theta\_new+theta(i-1))/2));

Actividad Propuesta:

Debemos modificar el anterior fichero de pinta\_robot introduciendo las medidas reales del robot para su correcta simulación y visión.



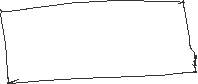
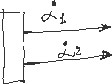
En la clase 8 se sentó las bases teóricas para la simulación del movimiento cinemático del robot móvil



Existen dos tipos de modelos posibles para un robot, el método Directo, y el método Indirecto.

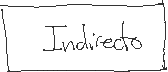
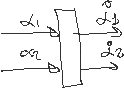
Método Directo:

Se basa en la premisa de que integrando es posible conseguir las posiciones futuras y el ángulo futuro.



Método Indirecto:

El método indirecto se basa en definir las velocidades y los ángulos para calcular la velocidad a la que deben ir las ruedas.



A continuación, vamos a demostrar la formula del modelo indirecto.

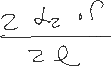
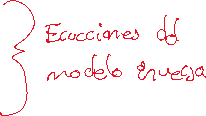
Como sabemos, las fórmula de V es : y de W :



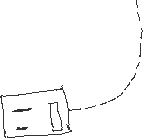
Como también sabemos que la formula de cada V puede ser expresada de la siguiente forma:



Juntando estas dos expresiones, podemos obtener:



Supongamos el ejercicio de un robot se quiere desplazar siguiendo una curva de radio R:



En cada punto tendremos una velocidad tangencial que define un radio R, siendo W la velocidad angular cuya formula es : podemos definir trayectorias curvas calculando dicha R.



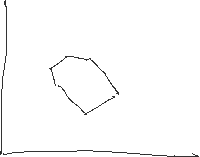
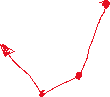
Tenemos que tener en cuenta que en dicha formula, si estamos en una línea recta( R = infinito) W tenderá a ser 0. Por lo que necesitamos algo adicional.

Siendo Ro(P) curvatura igual a 1/R podemos juntar las ecuaciones para conseguir W = P \* V, lo cual nos servirá para relacionar trayectorias.

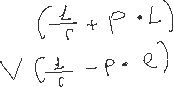
Clase 9:

En la clase se nos propuso construir un controlador con el modelo Inverso y odometría.

La premisa se basa en que dado dos puntos siempre se puede encontrar la circunferencia que los une, por lo que tenemos que configurar al robot para seguir dicha trayectoria.



Usando las ecuaciones de la clase anterior (del modelo inverso) podemos hayar la ecuación de la curva.



Clase 10:

En esta clase el objetivo es el seguimiento de un path (path tracking) usando los conocimientos de las dos clases anteriores.

Gráfico, Gráfico de líneas

Descripción generada automáticamente

Existen dos alternativas para cumplir con este objetivo

* El seguimiento de camino, que se basa en una función que hace un seguimiento de puntos.
* El seguimiento de trayectoria, que se basa en que el camino tiene asignado un perfil cinemático.

Se seguirá la estrategia del pure pursuit, el cual nos permite que en trayectorias suaves permite asegurar bajo error en la posición y orientación.

El pure pursuit funciona obteniendo en cada instante el camino más cercano a la posición (x,y) del vehículo, dicho punto objetivo se calculo escogiendo el punto del camino que dista un valor look\_ahead. Se aplica un algoritmo de control para converger al punto objetivo y se introduce en el bucle de control hasta que el punto final sea recorrido.

Clase 11:

Ya con el modelo de path tracking podemos seguir un camino determinado, pero existe un problema, nuestro robot seguirá **exactamente** la ruta definida, aunque esta contenga ángulos imposibles por la estructura física del robot.

Para evitar esto, se nos propone el objetivo de conseguir que el robot una curva aproximando el seguimiento para que sea posible de realizar en físico.

Gráfico, Gráfico de líneas

Descripción generada automáticamente

Para conseguir este objetivo se han añadido las siguientes modificaciones al ejercicio de la clase anterior(todos estos ejercicios son accesibles en los ficheros que se adjuntarán con esta memoria).

Texto, Carta

Descripción generada automáticamente

En este segmento de código declaramos el look\_ahead (podemos definirlo como la cantidad de puntos que separa los puntos a coger para la trayectoria).

Texto

Descripción generada automáticamente

Con este código mediremos la distancia al punto final del objetivo para reducir la velocidad y evitar un freno demasiado brusco.

Clase 12:

En esta clase programamos el control reactivo del robot para localizar objetos y desviar su trayectoria.

El esquema del robot sería el siguiente:



Necesitamos programar un control borroso ( controlador que sigue un conjunto de reglas prescritas).

Podemos actuar de dos formas, o bien el robot puede evitar la pared mirándola y manteniendo una distancia de seguridad con ella o avanzar hacia el lado sin pared.

A continuación se realiza los cálculos para reaccionar a obstáculos en un camino mediante un haz de lasers que detecten las distancias de estos.

