# Anexo A: Código MATLAB

## Crear variables globales

```
functioncotrolbrazo_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to cotrolbrazo (see VARARGIN)
% Choose default command line output for cotrolbrazo
handles.output = hObject;
% Update handles structure
guidata(hObject, handles);
global q1 q2 q3 q4 q5 rob a a2 a3 a4 a5 puntocin
a=0;q1=0;q2=0;q3=0;q4=0;q5=0;
a2=0;
a3=0;
a4=0;a5=0;
```

## Botón de inicio

Se toma los datos ingresados en los cuadros de texto que pertenecen a los parámetros D-H, se convierte los valores de ángulos ingresados a grados, se crea la matriz del robot y se procede a graficar en la posición inicial.

```
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global rob
an1= str2double(get(handles.a1, 'String'));
```

```
an2= str2double(get(handles.a2, 'String'));
an3= str2double(get(handles.a3, 'String'));
an4= str2double(get(handles.a4, 'String'));
an5= str2double(get(handles.a5, 'String'));
de1= str2double(get(handles.d1, 'String'));
de2= str2double(get(handles.d2, 'String'));
de3= str2double(get(handles.d3, 'String'));
de4= str2double(get(handles.d4, 'String'));
de5= str2double(get(handles.d5, 'String'));
alf1= str2double(get(handles.b1, 'String'));
conv1=(alf1*pi)/180;
alf2= str2double(get(handles.b2, 'String'));
conv2=(alf2*pi)/180;
alf3= str2double(get(handles.b3, 'String'));
conv3=(alf3*pi)/180;
alf4= str2double(get(handles.b4, 'String'));
conv4=(alf4*pi)/180;
alf5= str2double(get(handles.b5, 'String'));
conv5=(alf5*pi)/180;
L1 = Link('d', de1, 'a', an1, 'alpha', conv1);
L2 = Link('d', de2, 'a', an2, 'alpha', conv2);
L3 = Link('d', de3, 'a', an3, 'alpha', conv3);
L4 = Link('d', de4, 'a', an4, 'alpha', conv4);
L5 = Link('d', de5, 'a', an5, 'alpha', conv5);
rob = SerialLink([L1 L2 L3 L4 L5], 'name', 'andresbot');
rob.plot([0 0 0 0 0]);
```

**Función slider #1**

Se inicia las variables a utilizar, procesa los datos obtenidos del slider 1, uso del comando para calculo cinemática directa, visualización de los valores de coordenadas cartesianas.

Para los demás sliders el cálculo es el mismo.

```
function slider1_Callback(hObject, eventdata, handles)

% hObject    handle to slider1 (see GCBO)

% eventdata  reserved - to be defined in a future version of MATLAB

% handles    structure with handles and user data (see GUIDATA)

global q1 rob a a2 a3 a4 a5 punto x y z

punto=[a a2 a3 a4 a5];

handles.slider1=get(hObject,'Value');

q1=handles.slider1;

set(handles.angulo,'String',fix(handles.slider1));

rob.plot([a a2 a3 a4 a5])

cin=rob.fkine(punto);

x=cin(1,4);

y=cin(2,4);

z=cin(3,4);

set(handles.px,'String',fix(x));

set(handles.py,'String',fix(y));

set(handles.pz,'String',fix(z));
```

**Función slider #2**

```
function slider2_Callback(hObject, eventdata, handles)

% hObject    handle to slider2 (see GCBO)

% eventdata  reserved - to be defined in a future version of MATLAB

% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider

%        get(hObject,'Min') and get(hObject,'Max') to determine range of
slider

globalrob q2 a a2 a3 a4 a5 punto cin x y z

punto=[a a2 a3 a4 a5];

handles.slider2=get(hObject,'Value');

q2=handles.slider2;
```

```
set(handles.angulo2,'String',fix(handles.slider2));
rob.plot([a a2 a3 a4 a5])
cin=rob.fkine(punto);
x=cin(1,4);
y=cin(2,4);
z=cin(3,4);
set(handles.px,'String',fix(x));
set(handles.py,'String',fix(y));
set(handles.pz,'String',fix(z));
```

**Función slider #3**

```
function slider3_Callback(hObject, eventdata, handles)
% hObject    handle to slider3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'Value') returns position of slider
%get(hObject,'Min') and get(hObject,'Max') to determine range of slider
globalrob q3 a a2 a3 a4 a5 punto cin x y z
punto=[a a2 a3 a4 a5];
handles.slider3=get(hObject,'Value');
q3=handles.slider3;
set(handles.angulo3,'String',fix(handles.slider3));
rob.plot([a a2 a3 a4 a5]);
cin=rob.fkine(punto);
x=cin(1,4);
y=cin(2,4);
z=cin(3,4);
set(handles.px,'String',fix(x));
set(handles.py,'String',fix(y));
set(handles.pz,'String',fix(z));
```

**Función slider #4**

```
function slider4_Callback(hObject, eventdata, handles)
% hObject    handle to slider4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'Value') returns position of slider
%get(hObject,'Min') and get(hObject,'Max') to determine range of slider
globalrob q4 a a2 a3 a4 a5 punto cin x y z
punto=[a a2 a3 a4 a5];
handles.slider4=get(hObject,'Value');
q4=handles.slider4;
set(handles.angulo4,'String',fix(handles.slider4));
rob.plot([a a2 a3 a4 a5]);
cin=rob.fkine(punto);
x=cin(1,4);
y=cin(2,4);
z=cin(3,4);
set(handles.px,'String',fix(x));
set(handles.py,'String',fix(y));
set(handles.pz,'String',fix(z));
```

### Función slider #5

```
function slider5_Callback(hObject, eventdata, handles)
% hObject    handle to slider5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'Value') returns position of slider
% get(hObject,'Min') and get(hObject,'Max') to determine range of slider
globalrob q5 a a2 a3 a4 a5 punto cin x y z
punto=[a a2 a3 a4 a5];
handles.slider5=get(hObject,'Value');
q5=handles.slider5;
```

```
set(handles.angulo5,'String',fix(handles.slider5));
rob.plot([a a2 a3 a4 a5]);
cin=rob.fkine(punto);
x=cin(1,4);
y=cin(2,4);
z=cin(3,4);
set(handles.px,'String',fix(x));
set(handles.py,'String',fix(y));
set(handles.pz,'String',fix(z));
```

**Botón cinemática inversa**

En esta función se inicia en cero los sliders, se toma los datos de posición y rotación para calcular los ángulos que deben tomar para posicionar al robot, se grafica la posición obtenida.

```
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
globalrob
%Se piden las coordenadas del efector final
handles.slider1=0;
handles.slider2=0;
handles.slider3=0;
handles.slider4=0;
handles.slider5=0;
x = str2double(get(handles.punx, 'String'));
y = str2double(get(handles.puny, 'String'));
z = str2double(get(handles.punz, 'String'));
ax = str2double(get(handles.ax, 'String'));
by = str2double(get(handles.ay, 'String'));
cz = str2double(get(handles.az, 'String'));
```

```
p=(ax*pi)/180;
th=(by*pi)/180;
s=(cz*pi)/180;
T = transl(x, y, z);
R = eul2tr(p, th, s);
c=T*R;
i = rob.ikine(c, [0 0 0 0 0], [1 1 1 1 1 0]);
rob.plot(i);
```

### Botón trayectoria

Se ingresa la posición inicial y final que se desea obtener, con el comando de trayectoria se obtiene el movimiento que se desea alcanzar en un tiempo determinado.

```
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global rob
qf0=[0 0 0 0 0];
qf1=[pi/2 pi/2 pi/2 pi/2 0];
T0= rob.fkine(qf0);
T1=rob.fkine(qf1);
tempo=0:0.4:10;
q=jtraj(qf0, qf1, tempo);
rob.plot(q)
q=jtraj(qf1, qf0, tempo);
rob.plot(q)
pause(1)
```

### Comando PLOT

Los datos ingresados con la ayuda del comando Link, se puede representar gráficamente el robot, graficando el tamaño y posición de cada eslabón, la ubicación de las articulaciones, también crea una sombra en la base y delimita el espacio de trabajo.

```
function h = create_robot (robot, opt)
%disp('crea un Nuevo robot');
links = robot.links;
    s = opt.scale;
% crea un eje
ish = ishold();
if ~ishold
% ajuste de dimensiones de eje
axis(opt.workspace);
holdon
end
N = robot.n;
% crea la base
ifopt.base
bt = transl(robot.base);
bt = [bt'; bt'];
bt(1,3) = opt.floorlevel;
        line(bt(:,1), bt(:,2), bt(:,3), 'LineWidth', opt.basewidth,
'Color', opt.basecolor);
end
% agrega el nombre del robot
if opt.name
        b = transl(robot.base);
bz = 0;
ifopt.base
bz = 0.5*opt.floorlevel;
end
```

```
text(b(1), b(2)-s, bz, [' ' robot.name], 'FontAngle', 'italic',
'FontWeight', 'bold')

end

group = hggroup('Tag', robot.name);

h.group = group;

% grafica la articulación y eslabón del robot

for L=1:N

ifopt.debug

fprintf('create graphics for joint %d\n', L);

end

% grafica la unión de articulación de junta con el enlace

h.link(L) = hgtransform('Tag', sprintf('link%d', L), 'Parent', group);

% crea un cilindro de union

ifopt.joints

% crea el cuerpo de la articulación

if links(L).isrevolute

cyl('z', 2*s, opt.jointdiam/2*s*[-1 1], opt.jointcolor, [], 'Parent',
h.link(L));

else

% create an additional hgtransform for positioning and scaling the
prismatic

% element.  The element is created with unit length.

h.pjoint(L) = hgtransform('Tag', 'prismatic', 'Parent', h.link(L));

if links(L).mdh

% make the box extend in negative z-dir because scaling factor in animate

% must be positive

box('z', s, [0 -1], opt.jointcolor, [], 'Parent', h.pjoint(L));

else

box('z', s, [0 1], opt.jointcolor, [], 'Parent', h.pjoint(L));

end

end

end
```

```
% crea el cuerpo de enlace
% crea elementos que representan la traslación entre articulaciones
% This is drawn to resemble orthogonal plumbing.
ifrobot.mdh
% modified DH convention
if L < N
                A = links(L+1).A(0);
                t = transl(A);
if t(1) ~= 0
cyl('x', s, [0 t(1)], opt.linkcolor, [], 'Parent', h.link(L));
end
if t(2) ~= 0
cyl('y', s, [0 t(2)], opt.linkcolor, [t(1) 0 0], 'Parent', h.link(L));
end
if t(3) ~= 0
cyl('z', s, [0 t(3)], opt.linkcolor, [t(1) t(2) 0], 'Parent', h.link(L));
end
end
else
% standard DH convention
if L > 1
Ainv = inv(links(L-1).A(0));
                t = transl(Ainv);
if t(1) ~= 0
cyl('x', s, [0 t(1)], opt.linkcolor, [], 'Parent', h.link(L));
end
if t(2) ~= 0
cyl('y', s, [s t(2)], opt.linkcolor, [t(1) 0 0], 'Parent', h.link(L));
end
if t(3) ~= 0
```

```
cyl('z', s, [s t(3)], opt.linkcolor, [t(1) t(2) 0], 'Parent', h.link(L));
end
%line([0 t(1)]', [0 t(2)]', [0 t(3)]', 'Parent', h.link(L));
end
end
ifopt.jaxes&&opt.jvec
error('RTB:plot:badopt', 'Can''t specify ''jaxes'' and ''jvec''')
end
% create the joint axis line
ifopt.jaxes
line('XData', [0 0], ...
'YData', [0 0], ...
'ZData', 12*s*[-1 1], ...
'LineStyle', ':', 'Parent', h.link(L));

% create the joint axis label
text(0, 0, 12*s, sprintf('q%d', L), 'Parent', h.link(L))
end
% create the joint axis vector
ifopt.jvec
daspect([1 1 1]);
ha = arrow3([0 0 -12*s], [0 0 20*s]);
set(ha, 'Parent', h.link(L));
% create the joint axis label
text(0, 0, -12*s, sprintf('q%d', L), 'Parent', h.link(L))
end
end
ifopt.debug
fprintf('create graphics for tool\n');
end
```

```
% display the tool transform if it exists
h.link(N+1) = hgtransform('Tag', sprintf('link%d', N+1), 'Parent', group);
tool = eye(4,4);
if ~robot.mdh
tool = links(L).A(0);
end
if ~isempty(robot.tool)
tool = tool * robot.tool;
end
    t = transl(inv(tool));
if t(1) ~= 0
cyl('x', s, [0 t(1)], 'r', [], 'Parent', h.link(N+1));
end
if t(2) ~= 0
cyl('y', s, [s t(2)], 'r', [t(1) 0 0], 'Parent', h.link(N+1));
end
if t(3) ~= 0
cyl('z', s, [s t(3)], 'r', [t(1) t(2) 0], 'Parent', h.link(N+1));
end

% display the wrist coordinate frame
ifopt.wrist
ifopt.arrow
h.wrist = trplot(eye(4,4), 'labels', upper(opt.wristlabel), ...
'arrow', 'rgb', 'length', 15*s);
else
h.wrist = trplot(eye(4,4), 'labels', upper(opt.wristlabel), ...
'rgb', 'length', 15*s);
end
else
```

```
h.wrist = [];
end
% display a shadow on the floor
ifopt.shadow
% create the polyline which is the shadow on the floor
h.shadow = line('LineWidth', opt.shadowwidth, 'Color', opt.shadowcolor);
end
ifopt.trail
h.trail = plot(0, 0, opt.trail);
robot.trail = [];
end
% deal with some display options
ifopt.shading
lightinggouraud
light('position', opt.lightpos)
end
xlabel('X')
ylabel('Y')
zlabel('Z')
gridon
% restore hold setting
if ~ish
holdoff
end
h.floorlevel = opt.floorlevel;
h.robot = robot;
h.opt = opt;
% attach the handle structure to the top graphical element
set(group, 'UserData', h);
end
```

```
% draw a cylinder of radius r in the direction specified by ax, with an
% extent from extent(1) to extent(2)
functioncyl(ax, r, extent, color, offset, varargin)
if abs(extent(1) - extent(2)) < eps
return
end
ifisempty(offset)
offset = [0 0 0];
end
%fprintf('   cyl: %s, r=%f, extent=[%g, %g]\n', ax, r, extent);
n = 20;
r = [r;r];
theta = (0:n)/n*2*pi;
sintheta = sin(theta); sintheta(n+1) = 0;
switch ax
case'x'
y = r * cos(theta) + offset(2);
z = r * sintheta + offset(3);
            x = extent(:) * ones(1,n+1) + offset(1);
case'y'
            x = r * cos(theta) + offset(1);
            z = r * sintheta + offset(3);
            y = extent(:) * ones(1,n+1) + offset(2);
case'z'
x = r * cos(theta) + offset(1);
            y = r * sintheta + offset(2);
z = extent(:) * ones(1,n+1) + offset(3);
end
% walls of the cylinder
surf(x,y,z, 'FaceColor', color, 'EdgeColor', 'none', varargin{:})
```

```
% put the ends on
patch(x', y', z', color, 'EdgeColor', 'none', varargin{:});
end
% draw a cylinder of radius r in the direction specified by ax, with an
% extent from extent(1) to extent(2)
function box(ax, r, extent, color, offset, varargin)
if abs(extent(1) - extent(2)) < eps
return
end
%fprintf('   box: %s, r=%f, extent=[%g, %g]\n', ax, r, extent);
n = 4;
r = [r;r];
theta = (0:n)/n*2*pi;
sintheta = sin(theta); sintheta(n+1) = 0;
switch ax
case'x'
            y = r * cos(theta);
z = r * sintheta;
x = extent(:) * ones(1,n+1);
case'y'
            x = r * cos(theta);
z = r * sintheta;
            y = extent(:) * ones(1,n+1);
case'z'
y = r * cos(theta);
x = r * sintheta;
z = extent(:) * ones(1,n+1);
end
% walls of the cylinder
surf(x,y,z, 'FaceColor', color, 'EdgeColor', 'none', varargin{:})
```

```
% put the ends on
patch(x', y', z', color, 'EdgeColor', 'none', varargin{:});
end
% draw a tiled floor in the current axes
functioncreate_tiled_floor(opt)
xmin = opt.workspace(1);
xmax = opt.workspace(2);
ymin = opt.workspace(3);
ymax = opt.workspace(4);
% create a colored tiled floor
xt = xmin:opt.tilesize:xmax;
yt = ymin:opt.tilesize:ymax;
    Z = opt.floorlevel*ones(numel(yt), numel(xt));
    C = zeros(size(Z));
    [r,c] = ind2sub(size(C), 1:numel(C));
C = bitand(r+c,1);
    C = reshape(C, size(Z));
    C = cat(3, opt.tile1color(1)*C+opt.tile2color(1)*(1-C), ...
opt.tile1color(2)*C+opt.tile2color(2)*(1-C), ...
opt.tile1color(3)*C+opt.tile2color(3)*(1-C));
    [X,Y] = meshgrid(xt, yt);
surface(X, Y, Z, C, ...
'FaceColor','texturemap',...
'EdgeColor','none',...
'CDataMapping','direct');
end
% process a cell array of options and return a struct
% define all possible options and their default values
function opt = plot_options(robot, optin)
% timing/looping
```

```
opt.delay = 0.1;
opt.fps = [];
opt.loop = false;
opt.raise = false;
% general appearance
opt.scale = 1;
opt.zoom = 1;
opt.trail = [];
opt.workspace = [];
    opt.name = true;
opt.projection = {'ortho', 'perspective'};
opt.view = [];
opt.top = false;
% 3D rendering
opt.shading = true;
opt.lightpos = [0 0 20];
% tiled floor
opt.tiles = true;
    opt.tile1color = [0.5 1 0.5];  % light green
    opt.tile2color = [1 1 1];  % white
opt.floorlevel = [];
opt.tilesize = 0.2;
% shadow on the floor
opt.shadow = true;
opt.shadowcolor = [0.5 0.5 0.5];
opt.shadowwidth = 6;
% the base or pedestal
opt.base = true;
opt.basewidth = 3;
opt.basecolor = 'k';
```

```
% wrist
opt.wrist = true;
opt.wristlabel = {'xyz', 'noa'};
opt.arrow = true;
% joint rotation axes
opt.jaxes = false;
opt.jvec = false;
% joint cylinders
opt.joints = true;
opt.jointdiam = 5;
opt.jointcolor = [0.7 0 0];
% links
opt.linkcolor = 'b';
opt.toolcolor = 'r';
% misc
opt.movie = [];
% construir una lista de opciones de las fuentes
%   1.the M-file plotbotopt if it exists
%   2.robot.plotopt
%   3.command line arguments
if exist('plotbotopt', 'file') == 2
options = [plotbotoptrobot.plotoptoptin];
else
options = [robot.plotoptoptin];
end
% analizar las opciones
    [opt,args] = tb_optparse(opt, options);
if ~isempty(args)
error(['unknown option: 'args{1}]);
end
```

```
ifopt.top
opt.view = 'top';
end
if ~isempty(opt.projection)
opt.projection = 'ortho';
end
% muestra el tamaño de la figura
ifisempty(opt.workspace)
%
% heurística simple para calcular el alcance máximo del robot
%
L = robot.links;
if any(L.isprismatic)
error('Prismatic joint(s) present: requires the ''workspace'' option');
end
reach = 0;
fori=1:robot.n
reach = reach + abs(L(i).a) + abs(L(i).d);
end
reach = reach + sum(abs(transl(robot.tool)));
reach = reach/opt.zoom;

% if we have a floor, quantize the reach to a tile size
ifopt.tiles
reach = opt.tilesize * ceil(reach/opt.tilesize);
end
% creaunvolumen 3d
opt.workspace = [-reach reach -reach reach -reach reach];
% Si se ha dado un nivel de base, mejorar el volumen 3d
if ~isempty(opt.floorlevel)
```

```
opt.workspace(5) = opt.floorlevel;

else

opt.floorlevel = -reach;

end

else

reach = min(abs(diff(reshape(opt.workspace, [2 3]))));

ifopt.tiles

% establece limites de xy

opt.workspace(1:4) = opt.tilesize * round(opt.workspace(1:4)/opt.tilesize);

opt.floorlevel = opt.workspace(5);

end

end

% actualizar el facto de escala (dado por el usuario como multiplicador)
por una longitud derivada de

% la dimensión total del espacio

%  necesario al momento de crear el robot

opt.scale = opt.scale * reach/40;

if ~isempty(opt.fps)

opt.delay = 1/opt.fps;

end

end
```