دانشگاه صنعتی شریف

دانشکده مهندسی مکانیک

عنوان:

تمرینات سری سوم

محاسبه ماتریس های ژاکوبی و کاربرد آن ها

نگارش

محمدسعید صافی زاده

استاد راهنما

دکتر بهزادی پور

آبان ماه ۱۴۰۳

فهرست مطالب

۱ صورت سوالات

۱.۱ سوال اول

در نرم افزار متلب تابعی بسازید که حل سینماتیک مستقیم H یک ربات فضایی را در قالب جدول دناویت–هارتنبرگ و بردار متغیرهای مفصل Q به صورت سمبولیک دریافت کرده و ماتریس های ژاکوبی انتقالی Jv و دورانی Jw را به صورت سمبولیک بازگرداند، بکمک این تابع، ژاکوبی ربات تمرین ۳ را تولید و به همراه گزارشی از فرآیند حل در کد متلب تهیه کنید.

۲.۱ سوال دوم

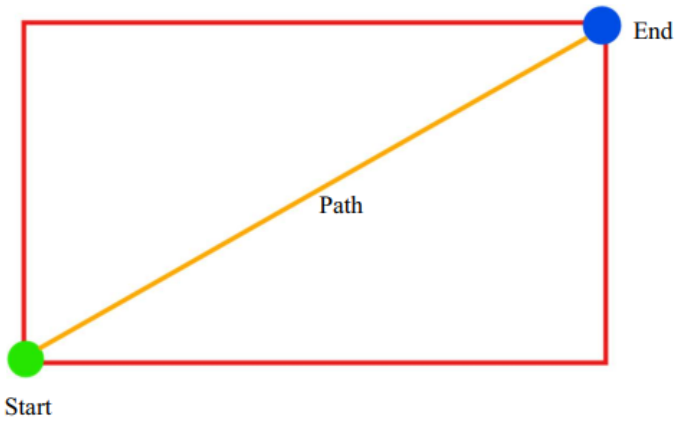
نقاط تکین را در صورت وجود به صورت پارامتریک در ربات تمرین ۳ بدست آورید و نتایج را به همراه گزارش ارائه نمایید.

۳.۱ سوال سوم

فرض کنید عملگر نهایی ربات تمرین ۳باید قادر باشد در فضای کاری خود، بردار سرعت $\vec{v} = (150, 100, 50)\frac{mm}{s}$ را در تمام نقاط ایجاد نماید. فضای کاری ربات را اینگونه در نظر بگیرید که مفصل اول θ_1 از ۰ تا ۸۰ درجه، مفصل دوم از -۴۵ تا ۴۵ درجه و مفصل سوم از d ۱۰۰ میلی متر تا ۲۰۰ میلی متر حرکت می کند. حداکثر سرعت مورد نیاز برای هریک از موتورهای سه گانه ربات را بدست آورید. گزارشی خلاصه از روند حل به همراه برنامه متلب و نتایج نهایی را ارائه نمایید.

٤.۱ سوال چهارم

فرض کنید عملگرنهایی ربات تمرین۳، یک قطعه را از نقطه شروع به پایان ببرد و میخواهیم مطابق شکل ۱در مسیر قطر یک مستطیل به طول ۲۴سانتی متر و عرض ۱۰ سانتی متر با سرعت ثابت ۵ سانتی متر بر ثانیه حرکت کند، امکان پذیری این حرکت را بررسی کرده و سرعت موتورهای این ربات را در طول حرکت ثبت نمایید. نتایج سوال را به همراه گزارش تئوری کامل و کد متلب مربوطه ارائه نمایید. (امتیازی ۲۰ درصد)



شکل ۱: شکل صورت سوال چهارم

۲ پاسخ سوال اول

۱.۲ تعریف تابعی برای دریافت جدول D-H از کاربر

برای دریافت جدول از کاربر می توان به روش های متفاوتی مانند دریافت جدول در اکسل و یا به صورت یک ماتریس استفاده کرد. در این جا سعی بر آن داشتم تا یک محیط گرافیکی برای تعامل بهتر کاربر ایجاد به کمک کد های موجود در اینترنت و تغییر برای رسیدن به حالت مطلوب خودمان ایجاد کنم. در ادامه کد را مشاهده می کنید که خط به خط به توضیح عملکرد آن می پردازم.

```
function DH_Table = DHInputGUI()
% Create the figure for GUI
fig = figure('Position', [100, 100, 600, 400], 'Name', 'D-H Parameter Table Input', ...
'CloseRequestFcn', @closeGUI);

% Create label and text box for number of rows input
uicontrol('Style', 'text', 'Position', [50, 350, 150, 20], 'String', ...
 'Enter number of joints:');
numRowsInput = uicontrol('Style', 'edit', 'Position', [200, 350, 100, 25]);

% Create button to generate the table
uicontrol('Style', 'pushbutton', 'Position', [320, 350, 100, 25], 'String', ...
'Generate Table','Callback', @(src, event) generateTableCallback());

% Table to display data with specified column names
dataTable = uitable('Parent', fig, 'Position', [50, 50, 500, 260], ...
'ColumnName', {'theta', 'd', 'l', 'alpha'});

% Button to retrieve data
uicontrol('Style', 'pushbutton', 'Position', [450, 350, 100, 25], 'String', ...
 'Get Data', 'Callback', @(src, event) getDataCallback());

% Wait for the figure to be closed
uiwait(fig);

% Nested function to generate the table based on user input for rows
function generateTableCallback()
numRows = str2double(numRowsInput.String);

if isnan(numRows) || numRows <= 0
errordlg('Please enter a valid number of joints.', 'Invalid Input');
return;
end

% Initialize table with empty strings (for text entry)
data = repmat({''}, numRows, 4);


% Set data and column properties of the table
dataTable.Data = data;
```

```
dataTable.ColumnEditable = true(1, 4); % Make all columns editable
end

% Callback function to retrieve data from the table
function getDataCallback()
data = dataTable.Data;
DH_Table = data;
% Display the input data in the command window
%disp('User Input Data:');
%disp(data);

% Assign the data to a variable in the base workspace
disp('Data has been saved to the workspace variable "DH_Table".');
end

% Function to handle GUI closure
function closeGUI(~, ~)
% Resume execution when the GUI is closed
uiresume(fig);
delete(fig);
end
end
```

در تابع DHInputGUI ابتدا یک figure با همین دستور درست می کنیم. attribute های آن شامل محل قرار گیری figure بر روی صفحه نمایش(position) و نام figure (name) می باشد. هم چنین برای بسته شدن این figure یک تابع به نام CloseGUI تعریف می کنیم.در بخش بعد یک المان برای دریافت تعداد مفصل های ورودی ایجاد می کنیم. به این صورت که با دستور uicontrol یک المان با attribute های style ، position و String ایجاد می کنیم که به ترتیب برای ایجاد textbox ، موقعیت قرار گیری این باکس و عنوانی که باید به کاربر نمایش دهد هستند. در ادامه از یک دکمه برای ساختن table استفاده می کنیم به این صورت که یک المان دقیقا با attribute های textbox قبل ایجاد می کنیم با این تفاوت که style آن از نوع pushbutton خواهد بود و سپس با تابعی به نام generateTableCallback عمل این دکمه را تعریف می کنیم که در ادامه به طور کامل به آن خواهیم پرداخت. سپس با دستور uitable یک جدول خالی با نام ستون های θ,d,l و α ایجاد می کنیم. در اینجا باید دیتا های ورودی توسط کاربر در جدول را به برنامه منتقل کنیم برای این کار، دکمه ای تعریف می کنیم مانند دکمه مرحله قبل اما در آن از تابعی با نام getDataCallback استفاده می کنیم که مقادیر جدول را ذخیره می کند که جلو تر به آن می پردازم. نهایتاً برای اینکه باقی کد ، تا زمان دریافت اطلاعات و ذخیره آن ها اجرا نشود؛ از دستور uiwait استفاده کردم. در اینجا به دو تابعی که در بالا استفاده شده بود می پردازیم. تابع اول برای تولید جدول بود به این نحو که با دریافت تعداد مفاصل، سطر های جدید برای وارد کردن عناصر جدول DH ایجاد می کرد. این تابع ابتدا ورودی textbox ای که در بالا ایجاد کرده بودیم را دریافت می کند و به فرم عدد تبدیل می کند. سپس چک می کند که این عدد مقداری صحیح و مثبت باشد. در ادامه با دستور repmat یک جدول خالی ایجاد می کنیم و با دستور ColumnEditable ، مقادیر این جدول را قابل ادیت می کنیم.
تابع ذخیره دیتا برای دکمه getdata نیز بدین شرح است. ابتدا مقادیر جدول را با دستور dataTable.Data درون متغیر data و سپس متغیر DH-Table می ریزم. با انجام این عمل برای متوجه شدن، پیغام ذخیره شدن دیتا نمایش داده می شود. نهایتاً برای اجرا شدن برنامه از دستور uiresume و برای بسته شدن figure از دستور delete(fig) استفاده می کنیم.

۲.۲ تعریف تابعی برای دریافت بردار متغیر های مفصلی از کاربر

در این بخش بر اساس تعداد مفصل هایی که در ابتدای برنامه بخش قبل از کاربر گرفته بودیم یک figure ایجاد می کنیم تا کاربر متغیر های مفصلی را در آن وارد کند. کد آن به صورت زیر می باشد.

```
function JointVariables = JointVariablesVectorInput(vectorSize)
```

```
% Create the figure for GUI
fig = figure('Position', [100, 100, 400, 300], 'Name',...
'Joint Variables Vector Input','MenuBar', 'none', 'CloseRequestFcn', @closeGUI);

% Create label for vector size
uicontrol('Style', 'text', 'Position', [50, 220, 150, 20], 'String',...
 ['Number of joint Variables: ' num2str(vectorSize)]);

% Create input fields for each element of the vector immediately
inputFields = gobjects(vectorSize, 1); % Preallocate graphics object array
for i = 1:vectorSize
uicontrol('Style', 'text', 'Position', [50, 220 - 30*i, 80, 20], 'String',...
['Joint ' num2str(i) ':']);
inputFields(i) = uicontrol('Style', 'edit', 'Position', [140, 220 - 30*i, 200, 25]);
end

% Create button to retrieve vector data
uicontrol('Style', 'pushbutton', 'Position', [250, 20, 100, 25], 'String', ...
'Get Vector','Callback', @getVectorCallback);

% Wait for the figure to be closed
uiwait(fig);

% Callback function to retrieve the vector data
function getVectorCallback(~, ~)
vector = cell(length(inputFields), 1); % Preallocate cell array for string vector

for i = 1:length(inputFields)
vector{i} = inputFields(i).String; % Get the string from each input field
end

% Display the vector in the command window
%disp('User Input Vector:');
%disp(vector);
% Save the vector to the base workspace
JointVariables = vector;
disp('Data has been saved to the workspace variable "JointVariables".');
end

% Function to handle GUI closure
function closeGUI(~, ~)
% Resume execution when the GUI is closed
uiresume(fig);
delete(fig);
end
end
```

در این تابع دقیقاً مشابه با قسمت قبل ابتدا یک figure با دستور figure ایجاد می کنیم که برای موقعیت، نام و بسته شدن از attribute های position, Name و تابع CloseRequestFcn استفاده کردیم. سپس برای نمایش تعداد مفاصلی که داریم یک باکس با دستور uicontrol به کاربر نمایش می دهیم. در ادامه برای دریافت متغیرهای مفصلی نیاز به باکس ورودی یا همان inputField داریم که بر اساس تعداد متغیر های مفصلی این کار را با دستور gobjects انجام می دهیم. حلقه for را در ادامه برای نام متغیر ها استفاده کردم و در این حلقه، موقعیت و قابل ادیت بودن این فیلد ها تعریف شده است.
با استفاده از دستور uicontrol یک المان و با attribure استایل آن را به دکمه تبدیل می کنیم و نهایتاً بعد از تعریف نام این دکمه، عمل آن را با تابع getVectorCallback تعریف می کنیم. برای جلوگیری از اجرا شدن ادامه برنامه ی اصلی زمانی که در ان تابع است از دستور uiwait استفاده کردم. برای ذخیره بردار ها که با فشرده شدن دکمه get vector با استفاده از حلقه for و پیمایش بر روی هر فیلد ورودی داده آن را در متغیر vector و سپس در متغیر JointVariables که خروجی تابع می باشد ذخیره می کنیم. و پیام ذخیره شدن آن را نمایش می دهیم.در آخر نیز با دستور uiresume برنامه اصلی را اجرا می کنیم و با دستور delete(fig)، figure را می بندیم.

۳.۲ تعریف تابع برای دریافت جدول دناویت هارتنبرگ و متغیر های مفصلی از کاربر و ساختن ماتریس همگن

در بخش قبل به طور مفصل به نحوه دریافت متغیر های مفصلی و جدول دناویت هارتنبرک توسط کاربر پرداختیم. حال در این بخش می خواهیم با استفاده از دو تابعی که در دو بخش قبل نوشتیم، جدول دناویت هارتنبرگ و متغیر های مفصلی را از کاربر دریافت کنیم و ماترس های ژاکوبین سرعت انتقالی و سرعت دورانی را با روش تحلیلی(مشتق گیری) بیابیم.

```
function [JointVar, Jv, Jw] = Jacobian()
DH_Table = DHInputGUI();
s = size(DH_Table);
s = s(1);
DH_Table_corrected_type = cell(s,4);
for i=1:s
for j=1:4
if isnan(str2double(DH_Table{i,j}))
DH_Table_corrected_type{i,j} = str2sym(DH_Table{i,j});
else
DH_Table_corrected_type{i,j} = str2double(DH_Table{i,j});
end
end
end

JointNumbers = s;
JointVariables = JointVariablesVectorInput(JointNumbers);
JointVar = cell(s,1);
for i=1:s
JointVar{i,1} = str2sym(JointVariables{i,1});
end

H = 1;
for i=1:JointNumbers
H = H * Rot('z', DH_Table_corrected_type{i,1})* ...
Trans('z', DH_Table_corrected_type{i,2})*Trans('x', DH_Table_corrected_type{i,3})* ...
Rot('x', DH_Table_corrected_type{i,4});
```

```
end
%disp(H);
```

خروجی های تابع را ، متغیر های مفصلی، ماتریس های ژاکوبین سرعت انتقالی و سرعت دورانی تعریف می کنیم . برای دریافت جدول دناویت هارتنبرگ از کاربر از تابع DHInputGUI استفاده می کنیم. تمامی عناصر دریافت شده از کاربر در متغیر DH-Table به صورت string ذخیره شده اند. برای اینکه بتوانیم تبدیل های استاندارد را پیاده سازی کنیم باید آن را به قسمت های پارامتری و عددی بشکنیم؛ بدین منظور، ابتدا تعداد سطر های جدول دناویت هارتنبرگ را پیدا کرده و یک cell برای ذخیره دیتا ها با توجه به تایپ آن ها می سازیم که این توضیحات با این سه خط کد زیر انجام می شوند.

```
s = size(DH_Table);
s = s(1);
DH_Table_corrected_type = cell(s,4);
```

در ادامه با حلقه for زیر بر روی هر درایه از سلول(cell) ورودی از کاربر حرکت می کنیم. سعی می کنیم هر درایه را به عدد تبدیل کنیم. در صورتی که این امر امکان پذیر باشد آن را در درایه متناظر، در سلول جدید با تایپ درست(Double) متغیر ذخیره می کنیم. در غیر این صورت متوجه می شویم که تایپ آن پارامتر است، بنابراین آن را به متغیر سیمبولیک تبدیل می کنیم و در سلول جدید می ریزیم. کد مربو به این توضیحات را در ادامه مشاهده می کنید.

```
for i=1:s
for j=1:4
if isnan(str2double(DH_Table{i,j}))
DH_Table_corrected_type{i,j} = str2sym(DH_Table{i,j});
else
DH_Table_corrected_type{i,j} = str2double(DH_Table{i,j});
end
end
end
```

برای متغیر های مفصلی و تبدیل آن ها از تایپ string به symbolic نیز مانند شیوه استفاده شده در بالا برای جدول ثناویت هارتنبرگ به صورت زیر عمل می کنیم.

```
JointNumbers = s;
JointVariables = JointVariablesVectorInput(JointNumbers);
JointVar = cell(s,1);
for i=1:s
JointVar{i,1} = str2sym(JointVariables{i,1});
end
```

حال همه شرایط مهیا است. کافی است از چهار تبدیل استاندارد برای هر سطر جدول دناویت هارتنبرگ برای یافتن تبدیل همگن همان سطر استفاده کنیم و سپس با ضرب کردن ماتریس های همگن همه سطر ها به ماتریس همگن کلی برسیم که این کار با کد زیر قابل دستیابی است.

```
H = 1;
for i=1:JointNumbers
```

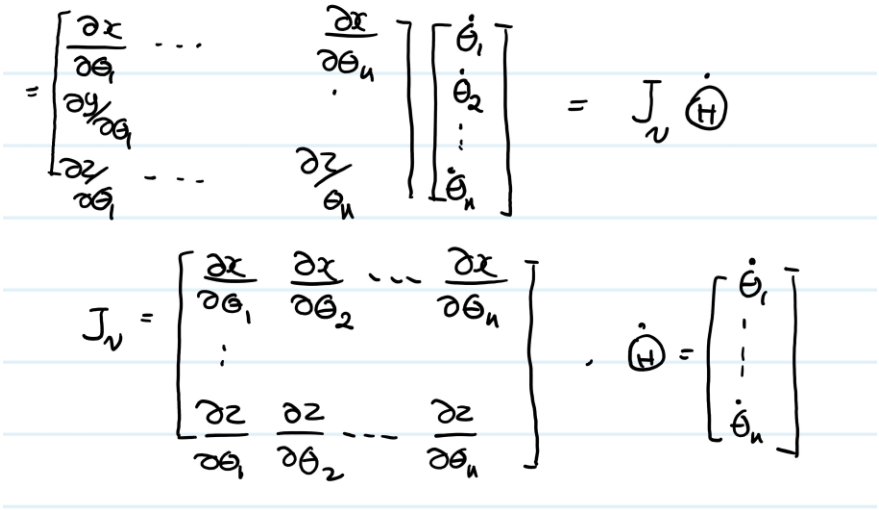
```
H = H * Rot('z', DH_Table_corrected_type{i,1})*...
Trans('z', DH_Table_corrected_type{i,2})*...
Trans('x', DH_Table_corrected_type{i,3})*Rot('x', DH_Table_corrected_type{i,4});
end
```

۴.۲ ساختن ماتریس ژاکوبی سرعت

در ادامه تابع بخش قبل، برای ساختن ماتریس ژاکوبی سرعت از روش تحلیلی کافی است مطابق اثبات جزوه در هر ستون ماتریس ژاکوبی از مولفه سرعت نسبت به هر متغیر مفصلی، مشتق بگیریم.

```
Jv = cell(3,s);
for i=1:3
for j=1:s
Jv{i,j} = simplify(diff(H(i,4), JointVar{j,1}));
end
end
```

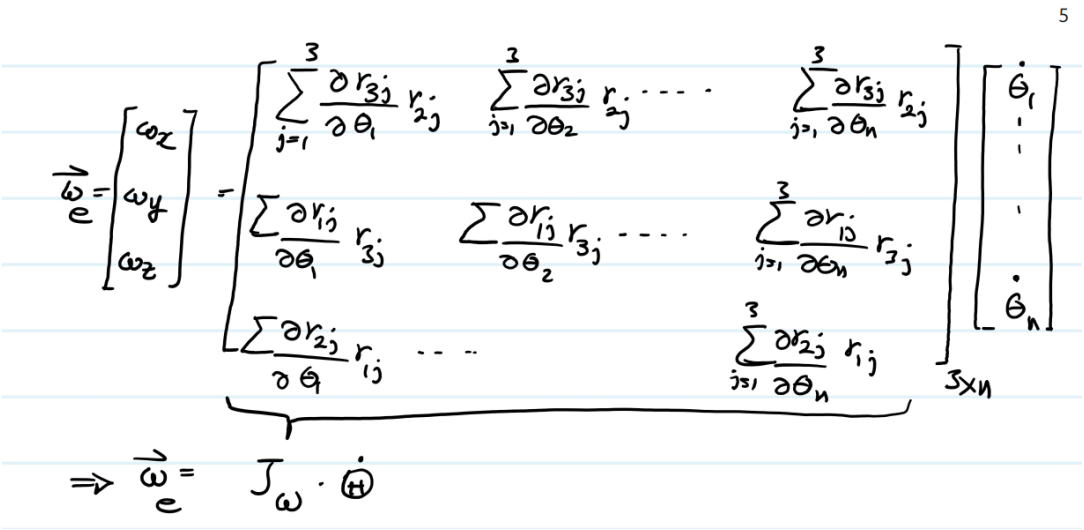
در این کد ابتدا یک سلول سه در n که n نماینده تعداد متغیر های مفصلی است ایجاد کردم. سپس از مولفه اول موقعیت(مولفه ی راستای x در بردار $\vec{O_0O_n}^{0}$) در ماتریس همگن نسبت به متغیر های مفصلی مشتق می گیریم و در سطر اول سلول Jv می ریزیم. همین کار را برای دو مولفه دیگر موقعیت انجام می دهیم و این امر به کمک دو حلقه for امکان پذیر خواهد بود. فرم دقیق تر و نوشتنی آن مطابق جزوه و شکل ۲ می باشد.



شکل ۲: فرم ریاضی ماتریس ژاکوبی سرعت انتقالی بدست آمده از روش تحلیلی

۵.۲ ساختن ماتریس ژاکوبی سرعت دورانی

برای ساختن ماتریس ژاکوبی سرعت دورانی نیز از روش تحلیلی استفاده کرده ام. همانگونه که در کلاس اثبات شد، این ماتریس به فرم نهایی ۳ در می آید که می توان آن را به صورت کد زیر نوشت.



شکل ۳: فرم ریاضی ماتریس ژاکوبی سرعت دورانی بدست آمده از روش تحلیلی

```
%%% Jw procedure
R = H(1:3,1:3);
Jw = cell(3,s);
for i=1:3
for j=1:s
if i == 1
sigma = 0;
for k=1:3
sigma = sigma + (diff(R(3,k), JointVar{j,1}) * R(2,k));
end
end

if i == 2
sigma = 0;
for k=1:3
sigma = sigma + (diff(R(1,k), JointVar{j,1}) * R(3,k));
end
end

if i == 3
sigma = 0;
for k=1:3
sigma = sigma + (diff(R(2,k), JointVar{j,1}) * R(1,k));
end
end

Jw{i,j} = simplify(sigma);
end
end
```

۶.۲ کد نهایی تابع ساخت ماتریس های ژاکوبی سرعت و سرعت دورانی

کد نهایی و جمع بندی شده در پایین آورده شده است :

```
function [JointVar, Jv, Jw] = Jacobian()
DH_Table = DHInputGUI();
s = size(DH_Table);
s = s(1);
DH_Table_corrected_type = cell(s,4);
for i=1:s
for j=1:4
if isnan(str2double(DH_Table{i,j}))
DH_Table_corrected_type{i,j} = str2sym(DH_Table{i,j});
else
DH_Table_corrected_type{i,j} = str2double(DH_Table{i,j});
end
end
end

JointNumbers = s;
JointVariables = JointVariablesVectorInput(JointNumbers);
JointVar = cell(s,1);
for i=1:s
JointVar{i,1} = str2sym(JointVariables{i,1});
end

H = 1;
for i=1:JointNumbers
H = H * Rot('z', DH_Table_corrected_type{i,1})*...
Trans('z', DH_Table_corrected_type{i,2})*Trans('x', DH_Table_corrected_type{i,3})*...
Rot('x', DH_Table_corrected_type{i,4});
end
%disp(H);

%%% Jv procedure
Jv = cell(3,s);
for i=1:3
for j=1:s
Jv{i,j} = simplify(diff(H(i,4), JointVar{j,1}));
end
end

%%% Jw procedure
R = H(1:3,1:3);
Jw = cell(3,s);
for i=1:3
for j=1:s
if i == 1
```

```
sigma = 0;
for k=1:3
sigma = sigma + (diff(R(3,k), JointVar{j,1}) * R(2,k));
end
end

if i == 2
sigma = 0;
for k=1:3
sigma = sigma + (diff(R(1,k), JointVar{j,1}) * R(3,k));
end
end

if i == 3
sigma = 0;
for k=1:3
sigma = sigma + (diff(R(2,k), JointVar{j,1}) * R(1,k));
end
end

Jw{i,j} = simplify(sigma);
end
end

end
```

۷.۲ تست و یافتن ماتریس ژاکوبی سرعت و سرعت دورانی مثال ربات اسکارا در جزوه برای صحه گذاری بر کدها و ماتریس های ژاکوبی بدست آمده

برای تست صحت و نحوه کار با این کد، از مثال جزوه برای ربات اسکارا برای بررسی آن استفاده می کنیم.
ماتریس دناویت هارتنبرگ این ربات به صورت ۱ و شکل آن به صورت شکل ۴می باشد.

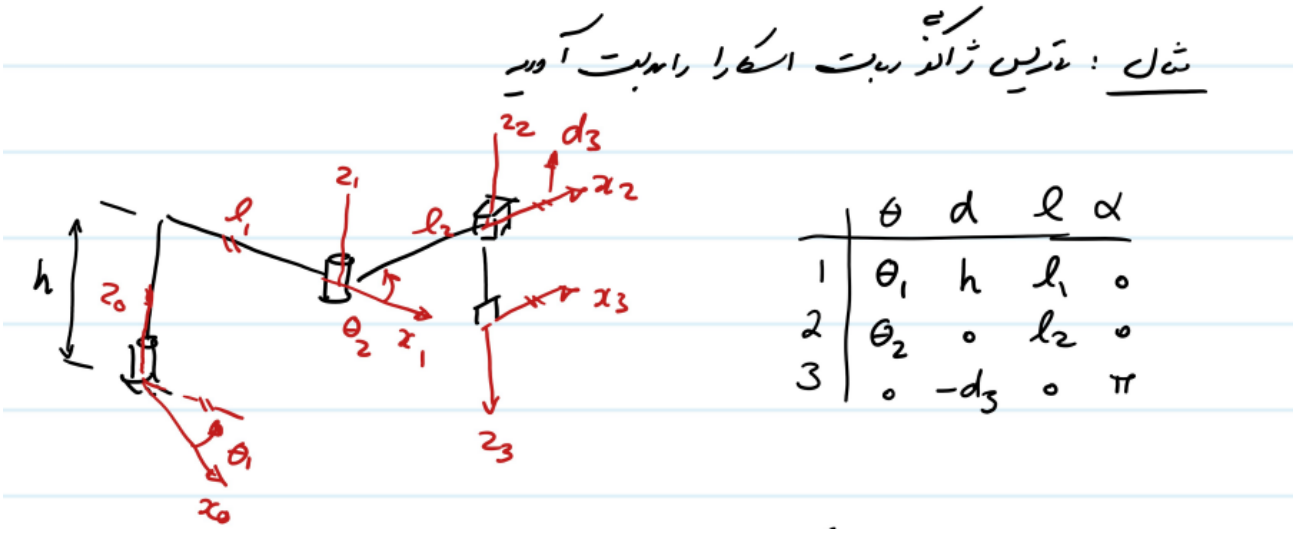
	θ	d	l	α
1	$\theta_1^\star$	h	$l1$	0
2	$\theta_2^\star$	0	$l2$	0
3	0	$-d_3^\star$	0	pi

جدول ۱: جدول دناویت هارتنبرگ برای ربات اسکارا در شکل ۴

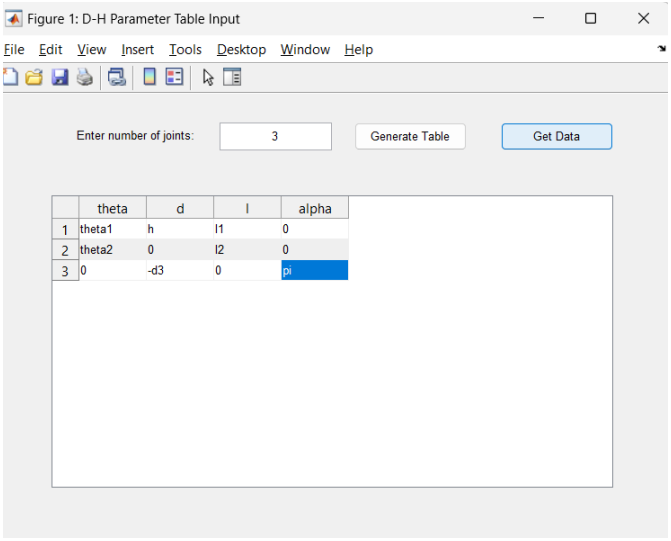
حال قطعه کد زیر را اجرا می کنیم:

```
[JointVar, Jv, Jw] = Jacobian();
```

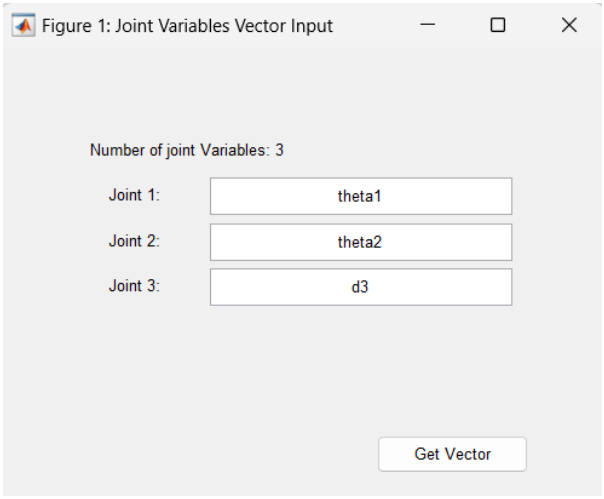
در پنجره نمایش شده ابتدا تعداد مفاصل را وارد می کنیم و سپس بر روی دکمه Generate Table کلیک می کنیم تا جدول ایجاد شود و مطابق با شکل ۵(آ) جدول دناویت هارتنبرگ را وارد می کنیم و دکمه Get Data را کلیک می کنیم. بعد از بستن این پنجره، یک پنجره دیگر برای دریافت متغیر های مفصلی نمایش داده می شود. در فیلد های مشخص شده متغیر های مفصلی را مطابق با شکل ۵(ب) وارد و بر روی دکمه



شکل ۴: ربات اسکارا



(آ) نحوه گرفتن ورودی جدول دناویت هارتنبرگ تست در برنامه



(ب) نحوه گرفتن ورودی متغیر های مفصلی تست در برنامه

شکل ۵: ورودی های برنامه

Get Vector کلیک کنید. نهایتاً با بستن این پنجره، ماتریس های ژاکوبی سرعت انتقالی ، سرعت دورانی و متغیر های مفصلی در workspace ذخیره می شوند.
نهایتاً ماتریس های Jv و Jw به صورت شکل ۶ و ۷ در می آیند.

```
>> cell2sym(Jv)

ans =

[- l2*sin(theta1 + theta2) - l1*sin(theta1), -l2*sin(theta1 + theta2),  0]
[  l2*cos(theta1 + theta2) + l1*cos(theta1),  l2*cos(theta1 + theta2),  0]
[                                         0,                         0, -1]
```

شکل ۶: ماتریس ژاکوبی سرعت انتقالی ربات اسکارا

```
>> cell2sym(Jw)

ans =

[0, 0, 0]
[0, 0, 0]
[1, 1, 0]

>>
```

شکل ۷: ماتریس ژاکوبی سرعت دورانی ربات اسکارا

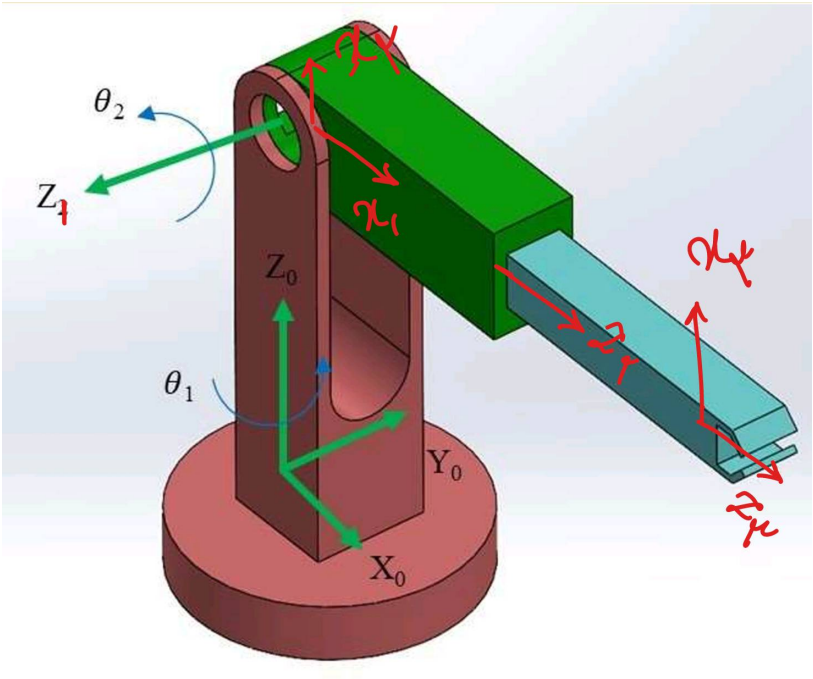
۸.۲ ساخت ماتریس ژاکوبی ربات تمرین سوم

مطابق با شکل ۸ جدول دناویت هارتنبرگ برای برای ربات تمرین سوم بدست می آوریم که مطابق با جدول ۲ خواهد شد. همچنین $l_1 = 35cm$ و $l_2 = 20cm$ می باشد.

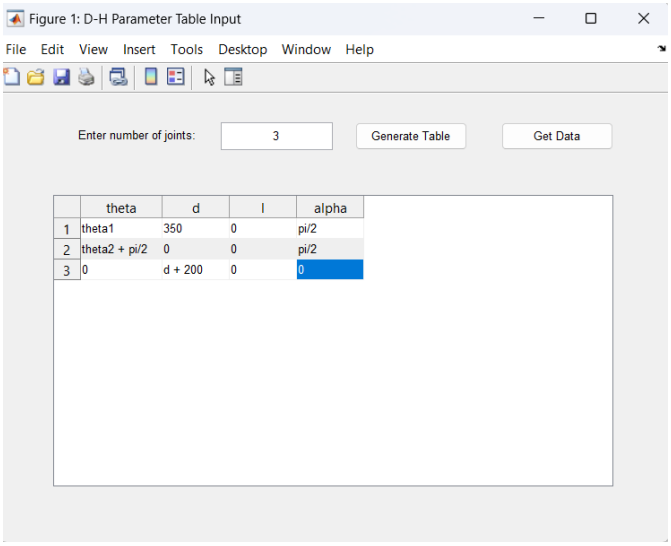
	θ	d	l	α
1	$\theta_1^\star$	l_1	0	$\frac{\pi}{2}$
2	$\theta_2^\star + \frac{\pi}{2}$	0	0	$\frac{\pi}{2}$
3	0	$d^\star + l_2$	0	0

جدول ۲: $\star$ نمایش دهنده متغیرها است

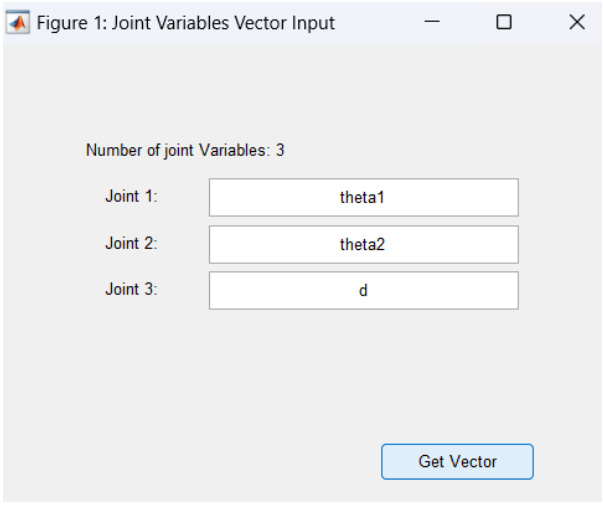
همین جدول را در برنامه وارد می کنیم و متغیر های مفصلی را به صورت θ_1 ، θ_2 و d تعریف می کنیم. مطابق شکل ۹(آ) و ۹(ب) نهایتاً مقادیر Jv و Jw به صورت شکل ۱۰ و ۱۱ در خواهند آمد.



شکل ۸: نمایش دستگاه های تعریف شده برای یافتن جدول DH



(آ) نحوه گرفتن ورودی جدول دناویت هارتنبرگ ربات تمرین ۳ در برنامه



(ب) نحوه گرفتن ورودی متغیر های مفصلی ربات تمرین ۳ در برنامه

شکل ۹: ورودی های برنامه برای ربات تمرین سوم

۳ پاسخ سوال دوم

۱.۳ توضیحات نقطه تکین

نقطه تکین یا singular نقطه ای می باشد که در آن ماتریس های ژاکوبی وارون پذیر نباشند. به این معنا که نتوان به ازای یک سرعت، و یک موقعیت مشخص برای ربات، ماتریس های ژاکوبی را یافت. برای یافتن نقطه تکین، متناسب با عملکرد ربات عمل می کنیم. برای مثال اگر نقاط تکین یک wrist را بخواهیم کافی است جایی که ماتریس ژاکوبی Jw وارون پذیر نیست را بیابیم و یا اگر ربات برای رفتن به موقعیت خاصی

```
>> cell2sym(Jv)

ans =

[-cos(theta2)*sin(theta1)*(d + 200), -cos(theta1)*sin(theta2)*(d + 200), cos(theta1)*cos(theta2)]
[ cos(theta1)*cos(theta2)*(d + 200), -sin(theta1)*sin(theta2)*(d + 200), cos(theta2)*sin(theta1)]
[                                 0,                cos(theta2)*(d + 200),                sin(theta2)]

>>
```

شکل ۱۰: ماتریس Jv برای ربات تمرین سوم

```
>> cell2sym(Jw)

ans =

[0,  sin(theta1), 0]
[0, -cos(theta1), 0]
[1,            0, 0]
```

شکل ۱۱: ماتریس Jw برای ربات تمرین سوم

(arm) طراحی شده است کافی است، نقاطی که Jv در آن وارون پذیر نیستند را بیابیم.
برای تعبیر وارون ناپذیری می توان از رنک و یا دترمینان استفاده کرد. به این معنا که زمانی یک ماتریس وارون ناپذیر خواهد بود که دترمینان آن صفر باشد. از همین موضوع در بخش بعد استفاده می کنیم تا نقاط تکین را بیابیم.

۲.۳ محاسبه نقاط تکین ربات تمرین سوم به صورت پارامتریک

برای محاسبه نقاط تکین ربات تمرین سوم، با توجه به اینکه عملکرد آن برای موقعیت دهی به end effector است کافی است جایی که Jv وارون پذیر نیست را بیابیم. یعنی دترمینان Jv را به ازای هر متغیر مفصلی برابر صفر قرار می دهیم تا نقاط تکین حاصل شوند. کد این توضیحات به شکل زیر خواهد شد.

```
[JointVar, Jv, Jw] = Jacobian();

eqn1 = det(cell2sym(Jv)) == 0;

sing_jv1 = solve(eqn1,JointVar{1});
sing_jv2 = solve(eqn1,JointVar{2});
sing_jv3= solve(eqn1,JointVar{3});
```

در خط اول تابعی که در پاسخ به سوال اول نوشته بودیم فراخوانی شده است و در ادامه یک معادله برای صفر قرار دادن دترمینان Jv تعریف شده است. دلیل استفاده از تابع cell2sym برای تعریف پارامتری درایه های Jv است. در ادامه با دستور solve سه معادله را برای یافتن حالت تکین حل می کنیم. خروجی این کد به صورت زیر خواهد بود.

```
>> sing_jv1
```

```
sing_jv1 =
Empty sym: 0-by-1

>> sing_jv2
sing_jv2 =
0

>> sing_jv3
sing_jv3 =
-200
-200
```

برای متغیر مفصلی θ_1 خروجی کد به صورت Empty شده است؛ به این معنا که به ازای هیچ مقداری از θ_1 این ربات تکین نخواهد شد. برای θ_2 ، مقدار صفر در آمده است یعنی به ازای هر اندازه ای در دیگر متغیر های مفصلی و $\theta_2 = 0$ ربات در وضعیت تکین است و هم چنین زمانی که $d = -200mm$ نیز وضعیت تکین داریم چرا که end efector نمی تواند سرعتی در راستای مفصل اول بگیرد که البته این حالت اصلاً با قیود فیزیک مسئله نمی سازد. دلیل یافتن دو جواب برای دترمینانِ صفر Jv3 داشتن ریشه مضاعف در آن می باشد.

٤ پاسخ سوال سوم

۱.٤ راه حل اول

در این راه حل با استفاده از سه حلقه ی for ، محدوده های داده شده را در ماتریس Jv جایگذاری می کنیم و با داشتن سرعت عملگر نهایی، مقادیر سرعت مفصل ها را در این حالات می یابیم(با کمک رابطه ی $\dot{\Theta} = J_v^{-1}V_e$)، در نهایت با ماکزیمم گرفتن بین این اعداد به مقدار ماکزیمم سرعت برای هر مفصل خواهیم رسید. برای اینکه تعیین گام در این حلقه های for می توان از استقلال در شبکه استفاده کرد.

```
answer = [];
syms theta1 theta2 d;
Jv_prime = rewrite(cell2sym(Jv), 'sin');
Jv_prime = rewrite((Jv_prime), 'cos');
Jv_prime = simplify(Jv_prime);

for t1i=0:10:80
for t2j =-45:15:45
for dk=100:10:200
answer(1:3,end+1) = subs(inv(Jv_prime),...
[theta1,theta2,d],[t1i*pi/180, t2j*pi/180, dk])*[150;100;50];
end
end
end
```

جواب های این کد به صورت زیر هستند:

```
>> max(abs(answer(1,:)))
ans =
0.6145
```

```
>> max(abs(answer(2,:)))
ans =
0.5419

>> max(abs(answer(3,:)))
ans =
186.7147
```

با تغییر گام و تبدیل کد بالا به کد زیر داریم:

```
answer = [];
syms theta1 theta2 d;
Jv_prime = rewrite(cell2sym(Jv), 'sin');
Jv_prime = rewrite((Jv_prime), 'cos');
Jv_prime = simplify(Jv_prime);

for t1i=0:5:80
for t2j =-45:5:45
for dk=100:5:200
answer(1:3,end+1) = subs(inv(Jv_prime), [theta1,theta2,d],...
[t1i*pi/180, t2j*pi/180, dk])*[150;100;50];
end
end
end
```

خروجی خواهد شد :

```
>> max(abs(answer(1,:)))
ans =
0.6145

>> max(abs(answer(2,:)))
ans =
0.5427

>> max(abs(answer(3,:)))
ans =
187.0302
```

عدد اول ثابت شده است. دوباره گام های حلقه for اول و دوم و سوم را دوباره کاهش می دهیم مطابق با کد زیر:

```
answer = [];
syms theta1 theta2 d;
Jv_prime = rewrite(cell2sym(Jv), 'sin');
Jv_prime = rewrite((Jv_prime), 'cos');
Jv_prime = simplify(Jv_prime);
```

```
for t1i=0:2:80
for t2j =-45:2:45
for dk=100:2:200
answer(1:3,end+1) = subs(inv(Jv_prime), [theta1,theta2,d],...
[t1i*pi/180, t2j*pi/180, dk])*[150;100;50];
end
end
end
```

آنگاه جواب خواهد شد :

```
>> max(abs(answer(1,:)))
ans =
0.6145

>> max(abs(answer(2,:)))
ans =
0.5428

>> max(abs(answer(3,:)))
ans =
187.0732
```

اعداد با مرحله قبل نهایتا در صدم متفاوت هستند، پس می توان گفت به دقت خوبی رسیده ایم. بنابراین جواب نهایی ما خواهد بود: $\dot{\theta}_1 = 0.6145\frac{rad}{s}$ ، $\dot{\theta}_2 = 0.5428\frac{rad}{s}$ و $\dot{d} = 187.0732\frac{mm}{s}$

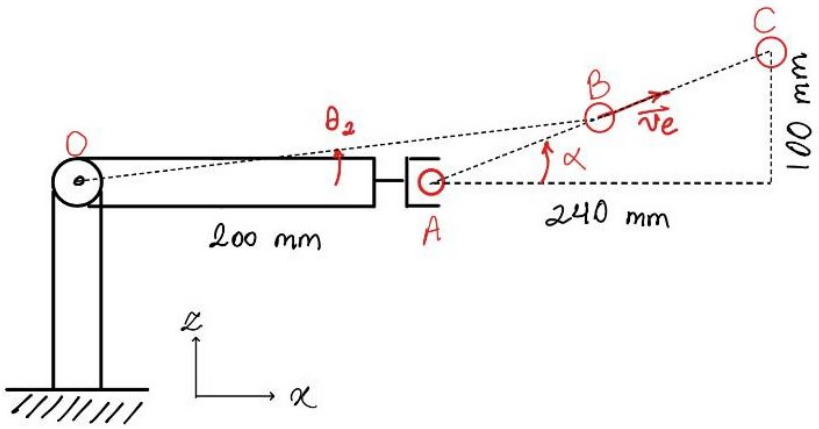
۲.٤ راه حل دوم

در این راه حل سرعت های مفاصل را به صورت پارامتریک برحسب متغیرهای مفصلی به دست می آوریم.یعنی مطابق با رابطه ی $\dot{\Theta} = Jv^{-1}V_e$ برای $\dot{\theta}_1$ از سمت راست معادله سه بار نسبت به θ_1 ، θ_2 و d مشتق می گیریم و برابر صفر قرار می دهیم. جواب های بدست آمده لزوماً اکسترمم نیستند و باید ابتدا و انتهای بازه های موجود را نیز بررسی کنیم. بنابراین با جایگذاری اعداد به دست آمده و ابتدا و انتهای بازه ها و ماکزیمم گرفتن از آن ها، $\dot{\theta}_1$ بدست می آید. به همین شیوه برای دو مولفه دیگر عمل می کنیم. این روش بار محاسباتی کمتر ولی پیچیدگی های خودش را هم دارد. برای مثال در حل ممکن است عدد مختلط برگرداند که نشان می دهد در آن بازه اکسترمم نداریم که همه این موارد باید در حل لحاظ شود. این راه حل تنها به عنوان ایده دوم آورده شده است.

۵ پاسخ سوال چهارم

قبل از توضیح پاسخ لازم به ذکر است این مسئله با هماهنگی با جناب آقای حسینی، دستیار درس در صفحه X_0Z_0 حل شده است.
در این سوال به دنبال یافتن سرعت موتور ها در مفاصل برای یک حرکت با سرعت مشخص در یک مسیر مشخص هستیم. برای این کار می توانیم این مسیر را به بخش های کوچک بر اساس زمان تقسیم کنیم. در این صورت در هر یک از این بخش ها موقعیت های مفصلی را داریم پس می توانیم ماتریس ژاکوبی سرعت را بیابیم. هم چنین اندازه و جهت سرعت را با توجه به مسیر حرکت داریم. پس می توانیم با استفاده از رابطه $\dot{\Theta} = Jv^{-1} * Ve$ در هر لحظه سرعت مورد نیاز موتور ها در مفاصل را بیابیم. در این جا ابتدا فرض می کنیم این حرکت در صفحه X_0Z_0 مطابق دستگاه های مختصات شکل ۸رخ می دهد. بنابراین متغیر مفصلی θ_1 تاثیری نخواهد داشت.
حال به توجه به هندسه مسئله در هر زمان متغیر های مفصلی θ_2 و d را پیدا می کنیم. می دانیم در شکل ۱۲ طول AB از رابطه ی Vt که V سرعت برابر ۵۰ میلی متر بر ثانیه و t زمان مد نظر است بدست می آید. همچنین می توانیم دو رابطه ی زیر را برای یافتن متغیر های مفصلی θ_2 و d بنویسیم.

$$(d+200)*sin(\theta_2) = 50*sin(\alpha)$$
$$(d+200)*cos(\theta_2) = 200+50*cos(\alpha)$$



شکل ۱۲: تصویر ربات تمرین سوم در صفحه X_0Z_0

که در این جا $\alpha = \arctan\frac{100}{240} = 22.62\,\text{deg}$ است. سرعت عملگر نهایی با توجه به α خواهد شد :

$$V_e = [50cos(\theta_2), 0, 50sin(\theta_2)]^T$$

و هم چنین زمان حرکت برای کل مسیر برابر با $t = \frac{x}{v} = \frac{(240^2+100^2)^{\frac{1}{2}}}{50} = 5.2s$ می باشد. بنابراین کد زیر از زمان 0 تا 5.2 ثانیه با گام های 0.01 ثانیه مقادیر θ_2 و d را پیدا کرده و در ماتریس Jv می گذارد و نهایتا با ضرب این ماتریس در V_e مقدار سرعت موتور ها را می یابد.

```
[JointVar, Jv, Jw] = Jacobian();
Jv_prime = rewrite(cell2sym(Jv), 'sin');
Jv_prime = rewrite((Jv_prime), 'cos');
Jv_prime = simplify(Jv_prime);

Ve = [50*cosd(22.61986);0;50*sind(22.61986)];

syms theta1 theta2 d
Tdot = [];
for i=0:0.01:5.2
eqn1 = (200+d)*sin(theta2) == 50 * i * sind(22.61986);
eqn2 = (200+d)*cos(theta2) == 200 + 50 * i * cosd(22.61986);
s = vpasolve([eqn1,eqn2]);
Tdot(1:3,end+1) = (subs(inv(Jv_prime), [theta1,theta2,d],[0, s.theta2, s.d])*Ve);
end
```

نحوه عملکرد کد در بالا توضیح داده شد، فقط در اینجا ذکر چند نکته الزامی است. اول اینکه از دستور vpasolve استفاده کردم تا یکی فقط از پاسخ های مثبت دستگاه دو معادله دو مجهول را بدست بیاورم، دوم اینکه این مقادیر را در ماتریسی سه در n می ریزم که n نماینده تعداد گام های

طی شده است. جواب ها به صورت ماتریس در کد متلب موجود هستند. در اینجا به ذکر بازه تقریبی آن را می پردازم. سرعت مفصل اول تاثیری ندارد. سرعت موتور مفصل دوم از مقدار 0.0189 تا مقدار 0.0962 رادیان بر ثانیه می باشد. سرعت موتور سوم نیز در محدوده ی 46.1538 میلی نتر بر ثانیه و 49.2681 میلی متر بر ثانیه می باشد.