تکلیف کامپیوتری پایانی شبکه های عصبی

نیم سال دوم ۱۴۰۲-۱۴۰۳

استاد درس : دکتر محمد فرخی

نویسنده : آرین حاجی زاده

شماره دانشجویی ۹۹۴۱۱۲۸۱

فهرست

چکیده :

در این مطالعه، هدف ما تخمین حرکت یک ربات چرخ‌دار غیرهولونومیک و کنترل آن با استفاده از شبکه توابع شعاعی پایه[۱] و به کار گیری آن در کنترل‌کننده عصبی است. برای دستیابی به این هدف، از ساختار شبکه آموزش داده‌شده در بخش اختیاری تکلیف اول کامپیوتری به‌عنوان مبنا استفاده شده و پس از اعمال تغییرات لازم، نتایج مورد بررسی قرار گرفته‌اند.

۱- الگوریتم توابع شعاعی پایه

همانطور که در بخش چکیده گفته شد، شبکه اصلی مورد بحث این بخش شبکه توابع پایه شعاعی یا همان RBF می‌باشد. تفاوت عمده این تکلیف با تکلیف پیشین آموزش برخط[۲] داده‌های تمرین، همچنین استفاده از شبکه عصبی جهت تخمین حرکت و نیز، کنترل حرکت ربات می‌باشد. در ادامه به توضیح عملکرد شبکه می‌پردازیم.

۱-۱- معماری شبکه[۳]

در این شبکه، معماری استفاده شده، الگوریتم ترکیبی بدون نظارت[۴] (k میانگین[۵]) برای لایه پنهان و الگورتیم با نظارت کمترین مربعات بازگشتی[۶] برای پیدا کردن وزن‌ها از لایه پنهان به لایه خروجی، می‌باشد. تابع استفاده شده به عنوان تابع گرین پایه، تابع گاوسی(رابطه ۱.۱) می‌باشد. همچنین برای الگورتیم کمترین مربعات بازگشتی از روابط (۱.۲) تا (۱.۶) استفاده شده است.

$$G = e^{-\frac{\|x-t_i\|}{2\sigma^2}} \qquad (۱.۱)$$

$$P(n) = R^{-1}(n) \qquad (۱.۲)$$

$$P(n) = P(n-1) - \frac{\left(P(n-1)\varphi(n)\varphi^T(n)P(n-1)\right)}{1+(\varphi^T(n)P(n-1)\varphi(n))} \qquad (۱.۳)$$

$$g(n) = \Phi(n)P(n) \qquad (۱.۴)$$

$$\alpha(n) = d(n) - \hat{w}^T(n-1)\Phi(n) \qquad (۱.۵)$$

---

[۱] Radial Basis Function

[۲] On-line

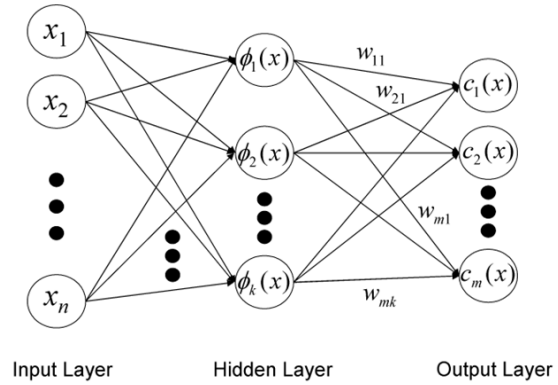
[۳] Network Architecture

[۴] Unsupervised

[۵] K-means

[۶] Recursive Least Squares

$$\hat{w}^T(n) = \hat{w}^T(n-1) + g(n)\alpha(n) \qquad (۱.۶)$$



شکل ۱.۱. ساختار کلی شبکه

لازم به ذکر است گزارشات لازم از نظر خطا و پارامترهای استفاده شده برای آموزش این شبکه در بنا به رعایت ترتیب مراحل انجام گرفته در بخش‌های بعدی مورد بررسی قرار گرفته‌اند.

۲- دینامیک سیستم

در چکیده گزارش به هدف این تکلیف اشاره شده است که کنترل گشتاور یک ربات چرخ‌دار غیرهولونومیک با استفاده از شبکه عصبی می‌باشد. برای این منظور، ابتدا با بهره‌گیری از داده‌های موجود در صورت سوال پروژه، دینامیک ربات با استفاده از روابط داده‌شده مدل‌سازی شد. برای انجام این کار تابعی با عنوان robot_dynamics در محیط نرم افزار پیاده سازی گردید.

$$M(q)\ddot{q} + V(q,\dot{q}) + G(q) = B(q)\tau + A(q) \qquad (۲.۱)$$

رابطه ۲.۱ به عنوان معادله دینامیکی ربات و معادله مرجع که در صورت سوال داده شده مورد استفاده قرار گرفته و از این رابطه دو رابطه پایه و بسیار مهم ۲.۲ و ۲.۳ بدست می‌آیند که برای پیش‌برد حل مساله نقش بسیار مهم و تعیین کننده دارند. از این جهت که تمامی ماتریس‌های به کار برده شده در رابطه بالا تعریف شده در صورت سوال می‌باشند از آوردن و تعریف دوباره آن‌ها در این گزارش خودداری شده است.

$$\ddot{q} = M(q)^{+}(B(q)\tau + A(q) - (V(q,\dot{q}) + G(q)) \qquad (۲.۲)$$

$$\tau = B(q)^{+}\,(M(q)\ddot{q} + V(q,\dot{q}) + G(q) - A(q)) \qquad (۲.۳)$$

در ادامه از روابط بالا برای حل مساله استفاده خواهد شد. اما همانطور که از روابط ۲.۲ و ۲.۳ پیداست از وارون جعلی[۱] در این روابط استفاده شده علت این امر احتمال تکین شدن ماتریس M برای رابطه ۲.۲ و برای رابطه ۲.۳، مربعی نبودن ماتریس میباشد. بنابراین دلایل از ماتریس وارون جعلی جهت رفع این مشکلات استفاده شده.

۳- ساختن شبکه

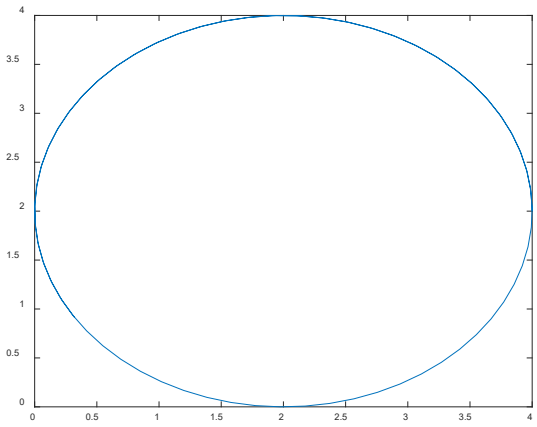
با دانستن مطالب گفته شده در بخش‌های پیشین، اکنون آماده سازی، آموزش و نیز نتایج ابتدایی در این شبکه مورد بررسی قرار می‌گیرد.

۳-۱- آماده سازی داده

برای آماده سازی داده‌های آموزش شبکه در ابتدا یک مسیر دایره شکل و پس از آن یک مسیر مربع شکل به عنوان مسیرهای مرجع مورد بررسی قرار گرفته‌اند. برای تعریف مسیر دایره شکل، از رابطه ۳.۱ استفاده شده است.

$$\begin{cases} x_r(t) = a + b\cos(\omega_r t) \\ y_r(t) = c + b\sin(\omega_r t) \\ \theta_r(t) = \omega_r t \end{cases} \qquad (۳.۱)$$

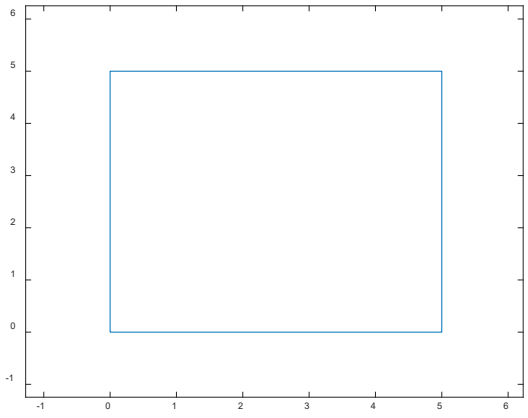
با تنظیم پارامترهای ورودی به صورت a = b = c = 2 و $\omega_r = 0.1$ شکل خروجی به صورت شکل ۳-۱ بدست می‌آید.



شکل ۳. ۱. مسیر مرجع دایره شکل

[۱] Pseudo Inverse

همچنین با تنظیم طول ضلع مربع برابر با ۵ خروجی شکل داده مورد نظر برابر با شکل ۳-۲ بدست می‌آید.



شکل ۳. 2 مسیر مرجع مربع شکل

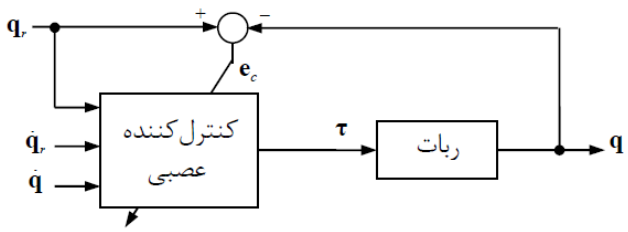
۳-۲- شبکه عصبی استفاده شده

این زیربخش، به نوعی در تکمیل مطالب بخش اول آمده و به این جهت در این قسمت آورده شده تا با آماده سازی و توضیحات انجام گرفته در بخش‌های پیشین شفافیت و رسایی بیشتری داشته باشد.

۳-۲-۱- آموزش شبکه

همانطور که در بخش یک گفته شد، شبکه به کار رفته در این پیاده سازی از نوع ترکیبی K میانگین و کمترین مربعات بازگشتی می‌باشد. در این پروژه، هدف آموزش شبکه توابع شعاعی پایه، به صورت برخط می‌باشد. بنابراین بایستی با اضافه شدن هر داده، شبکه آموزش ببیند. برای به انجام رسیدن این هدف، مقدار اولیه K برابر با ۱ در نظر گرفته شده و با افزایش تعداد داده‌ها، تعداد K نیز افزایش می‌یابد. این افزایش می‌تواند متناسب با اندازه داده ها و به مقدار آن‌ها بوده و یا در مقدار خاص تعیین شده متوقف شود. در پیاده سازی انجام گرفته مقدار نهایی K با استفاده از متغیر max_K محدود شده تا بیهوده مقدار K از اندازه متناسبی فراتر نرود. داده‌های ورودی بنابر شکل ۳-۳ تنظیم شده و به ورودی شبکه در هر تکرار[1] داده می‌شوند.

[1] iteration



شکل ۳.۳. دیاگرام بلوکی نشان دهنده ساختار و ورودی‌های شبکه

۳-۲-۲- خروجی شبکه

با نگاه به شکل ۳-۳ در می‌یابیم که با استفاده از این شبکه، بایستی مقادیر τ را به عنوان خروجی بدست آوریم. اما در صورت سوال خواسته شده که در این شبکه سعی نماییم، تابع هزینه آمده در رابطه (۳.۲) حداقل گردد. این مورد ممکن است در نگاه اول پیچیدگی ایجاد کند اما اگر مساله کمی دقیق بررسی شود واضح است که خروجی q نتیجه τ بوده و در واقع با حداقل کردن خطای حالت، خطای سیگنال کنترلی نیز به حداقل خواهد رسید.

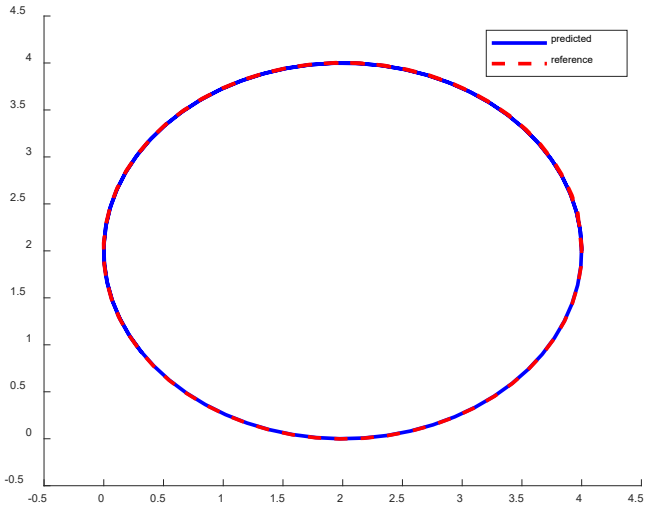
$$E_c = \frac{1}{2}e_c(k)^2 = \frac{1}{2}\big(q_r(k) - q(k)\big)^2 \qquad (۳.۲)$$

۳-۳- استفاده از نتایج شبکه عصبی

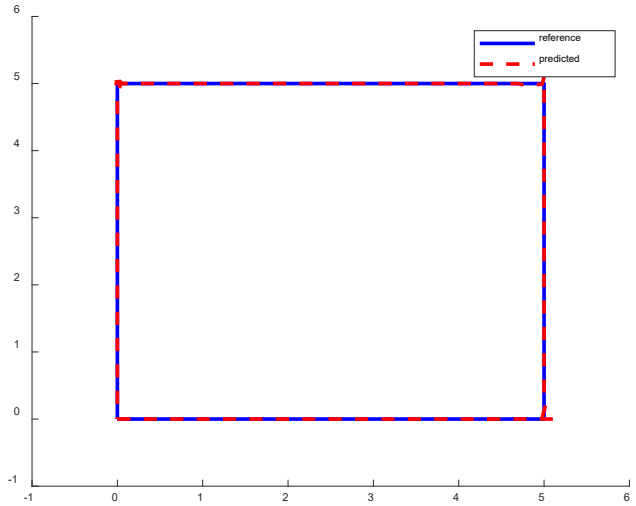
در بخش پیشین، خروجی به دست آمده به عنوان τ معرفی شد. برای دستیابی به خروجی مناسب، می‌توان با استفاده از معادله دینامیکی ربات که در رابطه (۲.۱) آمده است، و حل یک معادله دیفرانسیل با استفاده از داده‌های موجود، حالت ربات را در هر لحظه تخمین زد. در جعبه ابزار نرم‌افزار متلب، ابزاری به نام ode موجود است که انواع مختلفی دارد و دقت و نوع کاربرد آن‌ها در جدول ۳.۱ مشخص شده است. طبق داده‌های این جدول، از ابزار ode45 برای حل این معادله دیفرانسیلی استفاده شده است. در هر مرحله، از زمان نمونه‌برداری ضریب‌های i و i-1 به عنوان زمان مورد بررسی و حالت مرحله i ام برای شرایط اولیه استفاده می‌گردد. با حل این معادله، مقدار جدید q به دست می‌آید و در صورت آموزش صحیح شبکه، نتایج مطلوب حاصل خواهد شد. در ادامه نتایج و خروجی این شبکه در این مدل پیاده‌سازی برای دو مسیر مرجع پیشنهادی، در ساده ترین حالت و بدون اغتشاش خارجی و با تنظیم مناسب پارامترها آورده شده. این نتایج همانطور که گفته شد نتایج در ساده ترین حالت بوده و نتایج و بررسی‌های دقیق تر در بخش بعدی به طور مفصل مورد بررسی قرار خواهند گرفت.

جدول ۳.۱. انواع مدل‌های ode و کاربردهای آنها

Solver	Problem Type	Accuracy	When to Use
ode45	Nonstiff	Medium	Most of the time. ode45 should be the first solver you try.
ode23		Low	ode23 can be more efficient than ode45 at problems with crude tolerances, or in the presence of moderate stiffness.
ode113		Low to High	ode113 can be more efficient than ode45 at problems with stringent error tolerances, or when the ODE function is expensive to evaluate.
ode78		High	ode78 can be more efficient than ode45 at problems with smooth solutions that have high accuracy requirements.
ode89		High	ode89 can be more efficient than ode78 on very smooth problems, when integrating over long time intervals, or when tolerances are especially tight.
ode15s	Stiff	Low to Medium	Try ode15s when ode45 fails or is inefficient and you suspect that the problem is stiff. Also use ode15s when solving differential algebraic equations (DAEs).
ode23s		Low	ode23s can be more efficient than ode15s at problems with crude error tolerances. It can solve some stiff problems for which ode15s is not effective. ode23s computes the Jacobian in each step, so it is beneficial to provide the Jacobian via odeset to maximize efficiency and accuracy. If there is a mass matrix, it must be constant.
ode23t		Low	Use ode23t if the problem is only moderately stiff and you need a solution without numerical damping. ode23t can solve differential algebraic equations (DAEs).
ode23tb		Low	Like ode23s, the ode23tb solver might be more efficient than ode15s at problems with crude error tolerances.
ode15i	Fully implicit	Low	Use ode15i for fully implicit problems $f(t,y,y') = 0$ and for differential algebraic equations (DAEs) of index 1.



شکل ۳. ۴. نتیجه موفق در دنبال کردن مسیر دایره شکل



شکل ۳. ۵. نتیجه موفق در دنبال کردن مسیر مربع شکل

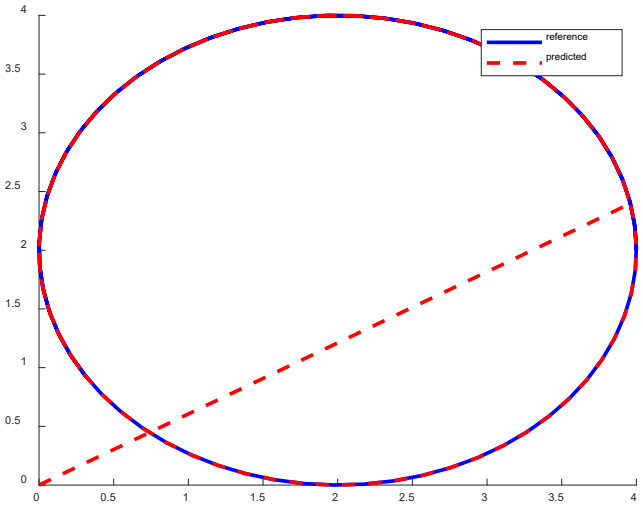
همانطور که از شکل ۳-۵ قابل مشاهده است در این مسیر مشکلی وجود دارد و این مشکل آمدن نقاط گوشه می‌باشد زیرا در این نقاط بر خلاف مسیر دایره شکل، نقاطی مانند نقاط گوشه با استفاده از نقاط پیشین قابل پیش بینی نیستند. اما مشاهده می‌شود، در این نقاط هم شبکه به سرعت وضعیت خود را جبران سازی می‌نماید.

۴- بررسی نتایج تحت شرایط مختلف

در این بخش با توجه به خواسته صورت مساله، هدف بررسی قوام شبکه تحت شرایط غیر ایده‌آل می‌باشد. برای این منظور در هر زیر بخش قسمتی از این اهداف را به انجام رسیده‌است.

۴-۱- شرایط اولیه نامطلوب

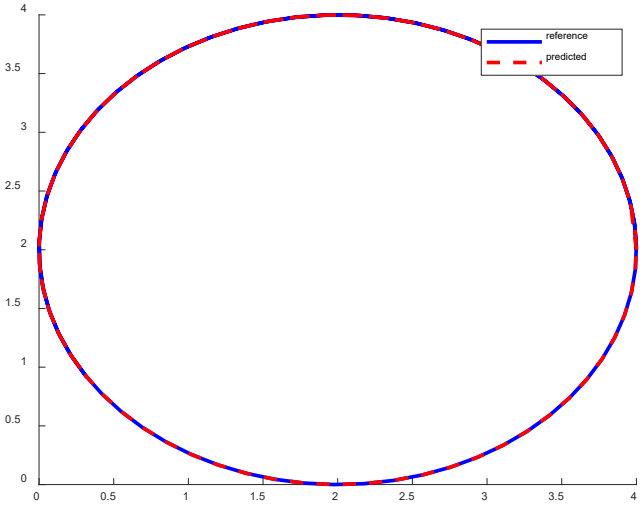
در نتایج بخش گذشته شرایط اوله با موقعیت اولیه ربات یکسان در نظر گرفته شد. در این قسمت بررسی می‌شود، در صورتی که شرایط اولیه ربات در مکان غیر از مکان مرجع اولیه باشد چه اتفاقی رخ می‌دهد.



شکل ۴. ۱. خروجی سیستم در صورت شروع از نقاط اولیه دور

۴-۲- تغییر پارامترهای سیستم

با تغییر پارامترهای موجود در سیستم در حالتی که هریک را به مقدار ۲ برابر افزایش می‌دهیم خروجی به صورت شکل ۴.۲ قابل مشاهده خواهد بود. پارامترهای جدید در شکل ۴.۳ مشاهده می‌شوند.



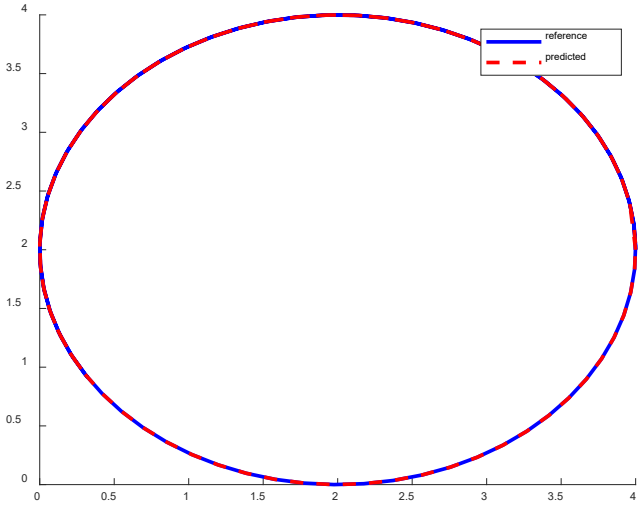
شکل ۴. ۲. خروجی سیستم در حالتی که پارامترهای سیستم تغییر کند.

```
%% Mobile Robot Parameters

m = 30; % kg
I = 10; % kg*m^2
R = 0.3; % m
r = 0.1; % m
d = 0.2; % m
```

شکل ۴. ۳. خروجی سیستم در حالتی که پارامترهای سیستم تغییر کند.

در رابطه با خروجی این مرحله می‌توان گفت که انتظار نمی‌رفت تغییر محسوسی در سیستم دیده شود. زیرا آموزش این سیستم به صورت برخط بوده و درصورتی که از ابتدا پارامتر ها را تغییر دهیم، نباید تغییری دیده شود. اما در صورتی که در میانه راه تغییری در پارامتر جرم سیستم ایجاد کنیم:

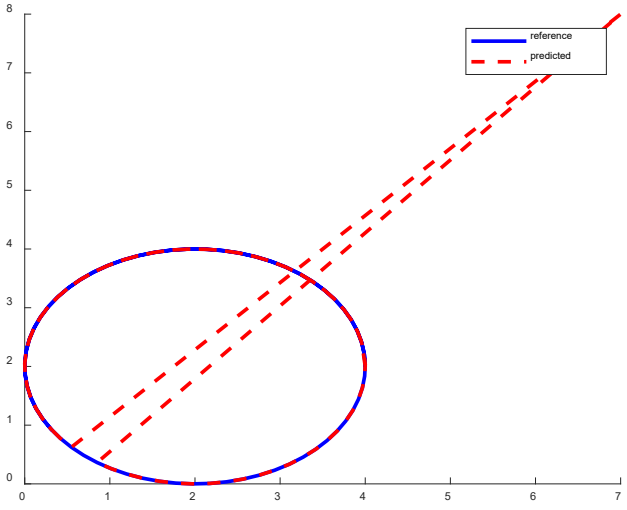


شکل ۴. ۴. عملکرد سیستم با تغییر پارامتر در وسط مسیر

همانطور که مشاهده می‌گردد با تغییر جرم در میانه مسیر نیز، تاثیری بر عملکرد ربات مشاهده نمی‌شود.

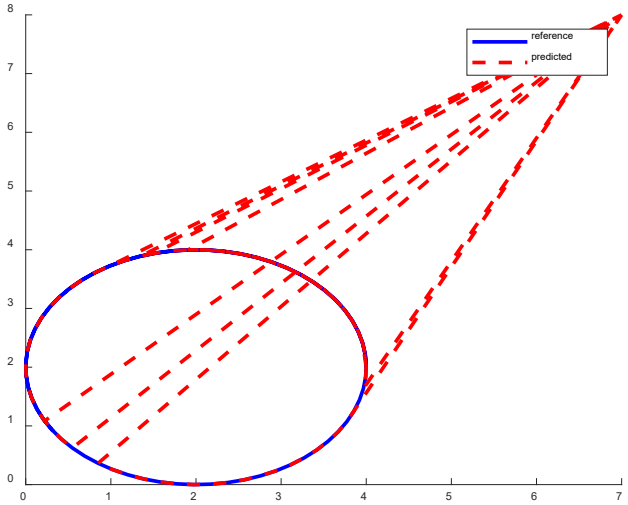
۳-۴- اغتشاش خارجی

در این قسمت در میانه مسیر حالت سیستم به نقطه‌ای خارج از مسیر مرجع انتقال می‌یابد. در شکل ۴.۵ عملکرد سیستم در مقابل این اغتشاش مشاهده می‌گردد.



شکل ۴. ۵. عملکرد سیستم نسبت به یک اغتشاش خارجی

همانطور که مشاهده می‌گردد سیستم به خوبی توانسته اثر این اغتشاش را خنثی نماید و پس از اعمال اغتشاش به سرعت به مسیر اصلی بازگشته. حال اثر چند اغتشاش همزمان مورد بررسی قرار می‌گیرد.

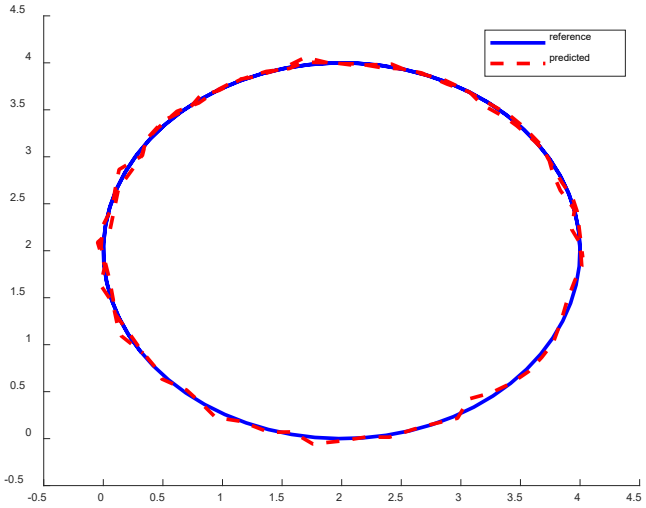


شکل ۴. ۶. خروجی سیستم برای حالت چندین اغتشاش

مشاهده می‌گردد که حتی با وجود اعمال اغتشاش زیاد به این سیستم همچنان موفق به بازگشت به مسیر اصلی می‌باشد. در این قسمت برای هر ۴۰ واحد از زمان نمونه برداری یک اغتشاش به سیستم وارد گردیده.

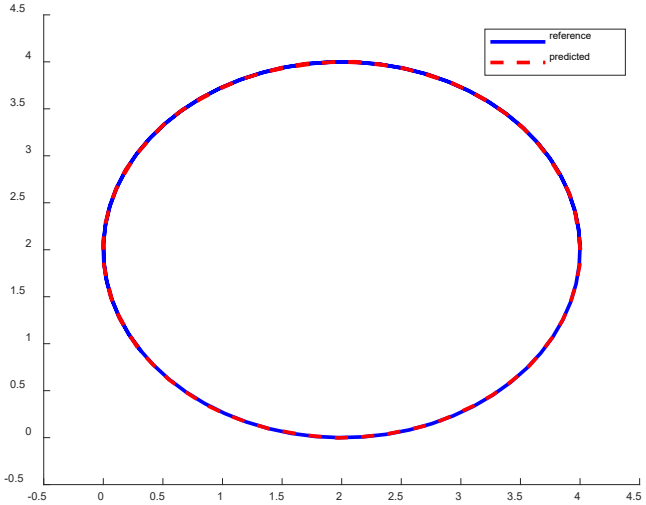
۴-۴- نویز اندازه گیری

در این بخش داده‌های اندازه گیری شده، با نویز جمع شونده گاوسی ترکیب شده و نتایج مشاهده می‌گردد. در این بخش سعی شده تا با نسبت سیگنال به نویز[۱] مختلفی مطالعه صورت گیرد.

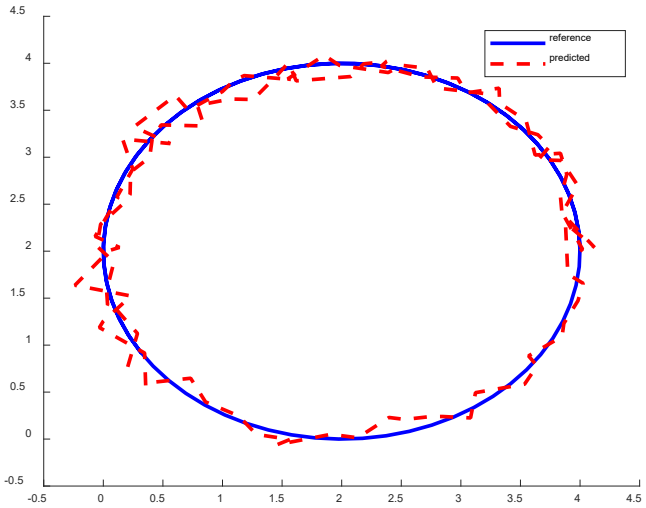


شکل ۴. ۷. خروجی سیستم با ۳۰ SNR

[۱] SNR



شکل ۴. ۸. خروجی سیستم با SNR ۵۰



شکل ۴. ۹. خروجی سیستم با SNR ۲۰

همانطور که مشاهده می‌گردد، تا زمانی که نسبت سیگنال به نویز مقدار بسیار بالایی مانند ۵۰ داشته باشد، اثر نویز به چشم نیم‌آید. اما با وارد کردن نویز با نسبت سیگنال به نویز پایین تر سیستم کمی دچار اختلال می‌شود اما همچنان قادر است مسیر را تا حدود خوبی دنبال نماید.

۵- قابلیت پیاده سازی بلادرنگ

به طور کلی از نظر محاسباتی پیچیدگی سنگینی برای این الگوریتم وجود ندارد. اما مسئله اصلی که باعث هزینه زمانی و هزینه محسباتی می‌گردد، آموزش برخط و نیز بالا بردن تعداد داده‌ها خواهد بود. در این مسئله برای محاسبات از تعداد ۱۰۰ الی ۲۰۰ داده استفاده گردیده و مشاهده شده زمان محسابات برای این حجم داده به صورت زیر است:

Elapsed time is 0.655510 seconds

در صورتی که تعداد داده‌ها را به ۳۰۰ افزایش دهیم زمان محاسبه به شکل زیر خواهد بود:

.Elapsed time is 2.805405 seconds

و در نهایت این زمان برای تعداد ۵۰۰ داده به صورت زیر خواهد بود:

.Elapsed time is 6.538451 seconds

مشاهده می‌گردد که با افزایش تعداد این داده‌ها، زمان محاسبه افزایش قابل توجهی می‌یابد. این مورد در صورتی که تمام داده‌ها را موجود بوده و آموزش برون خط باشد مشکل بزرگی نخواهد بود اما در صورت تمایل به استفاده از آموزش برخط این موضوع به مشکل بزرگی بدل خواهد شد و قابلیت استفاده به صورت بلادرنگ و همزمان از این سیستم گرفته می‌گردد و یا هزینه لازم برای در دست داشتن یک سیستم قدرتمند تر باید در نظر گرفته شود. برای استفاده‌های بلادرنگ استفاده از شبکه‌های سریع تر مانند ماشین بردار پشتیبان[۱] که از نظر محاسباتی کمی سبک تر بوده، پیشنهاد می‌گردد.

۶- نتیجه گیری

در این گزارش سعی گردید، با استفاده از آموزش برخط یک شبکه توابع شعاعی پایه برای یک ربات غیر هولونومیکی چرخ دار، مسیر ربات تخمین زده شده و کنترل کننده‌ای جهت کنترل گشتاور ورودی موتور ها لحاظ گردد. در این مطالعه با بررسی عوامل مختلفی که برروی شبکه تاثیرگزار بودند سعی گردید، قوام شبکه و نیز عملکرد آن در شرایط مختلف برای این ربات مورد بررسی قرار گیرد. نکته بسیار مهم درباره آموزش برخط این شبکه وجود مشکل زمان در به نتیجه رسیدن شبکه بود. باید در آموزش این شبکه به تعداد داده‌های آموزش و بالا نرفتن بیش از اندازه نویز توجه ویژه داشت. اما به طور کلی شبکه قوام بالا و مقاومت مناسبی در مقابل اثرات نا مطلوب از خود نشان داده.

[۱] Support Vector Machine (SVM)

۷- پیوست

۷-۱- مسیر دایره‌ای اصلی

```
%% Clearing Existing Data
clc
clear all
close all

%% Mobile Robot Parameters
R = 0.05; % Wheel radius
L = 0.1; % Distance between wheels
r = 0.05; % Radius of each wheel
m = 15;   % Robot mass
I = 5;    % Moment of inertia

%% Simulation parameters
sample_time = 1;       % Reducing the sample time can cause higher CPU usage
SimulationTime = 100;     % Change this if you want to change the simulation
duration
counter = 0;
time = 0:sample_time:SimulationTime;

%% Initial state
q = [0, 0, 0];  % Initial state [x_c, y_c, theta]
q_dot = [0, 0, 0];          % Initial linear velocity
omega = 0;      % Initial angular velocity

%% Parameters of the Circular Trajectory
a = 2;       % Circle Center in meters (X Axis)
c = 2;       % Circle Center in meters (Y Axis)
b = 2;       % Radius in meters
omega_r = 0.1; % Angular velocity for theta reference

%% Desired Trajectory (You can Change The Desired Trajectory)
x_ref = a + b * cos(omega_r * time);
y_ref = c + b * sin(omega_r * time);
theta_ref = omega_r * time;

q_ref = [x_ref; y_ref; theta_ref]';

x_dot_ref = -b * omega_r * sin(omega_r * time);
y_dot_ref =  b * omega_r * cos(omega_r * time);
omega_ref = omega_r * ones(size(time));

q_dot_ref = [x_dot_ref; y_dot_ref; omega_ref]';

x_ddot_ref = -b * omega_r^2 * cos(omega_r * time);
y_ddot_ref = -b * omega_r^2 * sin(omega_r * time);
alpha_ref  = omega_r^2 * ones(size(time));

q_ddot_ref = [x_ddot_ref; y_ddot_ref; alpha_ref]';

plot(q_ref(:,1),q_ref(:,2))
%Tobe or not To be
q = q_ref(1,:);
q_dot = q_dot_ref(1,:);
```

```
%% Improved Training Data Generation
K = 1;
iterations = 100;
lambda = 0.08;
max_K = 20;

for i = 1:size(q_ref,1)
    e = q_ref(i,:) - q;

    q_all(i,:) = q;
    q_dot_all(i,:) = q_dot;
    e_all(i,:) = e;
    X_train{i} = [q_ref(1:i,:),q_dot_ref(1:i,:),e_all(1:i,:),q_dot_all(1:i,:)];

    if(K > max_K)
        K = max_K;
    end

    centers = myKmeans(X_train{i}, K);

    max_d = 0;
    for temp = 1:K
        for temp2 = 1:K
            if(norm(centers(temp2)-centers(temp)) > max_d)
                max_d = norm(centers(temp2)-centers(temp));
            end
        end
    end
    % getting sigmas according to handouts and the book with K-means and RLS
    sigma = max_d*max(max(q_ref))/sqrt(2*K)  ;

    % this function is defined at the end of the file
    [out, Phi] = rbf_HL(X_train{i}, centers, sigma);

    % adding bias to the weights
    Phi = [Phi , ones(size(X_train{i},1),1)];



    P = lambda * eye(K + 1);
    w = zeros(K + 1, 2); % Initialize weight matrix for each output dimension

    for n = 1:i
        % Update P matrix
        P = P - (P * Phi(n,:)' * Phi(n,:) * P) / (1 + Phi(n,:) * P * Phi(n,:)');

        % Calculate gain vector
        g = P * Phi(n,:)';

        % Prediction error
        [M, V, G, B, A] = robot_dynamics(q_ref(i,:), q_dot_ref(i,:));
        tau_ref = pinv(B)*(M*q_ddot_ref(i,:)' + V + G - A);
        pre = tau_ref' - Phi(n,:) * w;

        % Update weights
        w = w + g * pre;
    end

    % Calculate mean squared error
```

```
    T = Phi * w;
    if(i>1)
        [t,x] = ode45(@(t,x) odefcn(t,x,T), sample_time*[i-1 i],
[q_ref(i,:)';q_dot_ref(i,:)']);
        q = ([x(length(t),1),x(length(t),2),x(length(t),3)]);
        q_dot = [x(length(t),4),x(length(t),5),x(length(t),6)];
    end

end
%% Plotting Output Data
figure;
hold on
plot(q_ref(1:size(q_all,1),1),q_ref(1:size(q_all,1),2),'b',LineWidth=2)
plot(q_all(:,1),q_all(:,2),"r--",LineWidth=2);
legend(["reference" "predicted"]);
hold off
%% function definitions
function [output, Phi] = rbf_HL(X, centers, sigma)
    num_data = size(X, 1);
    num_centers = size(centers, 1);
    Phi = zeros(num_data, num_centers);

    for i = 1:num_data
        for j = 1:num_centers
            Phi(i, j) = exp(-norm(X(i,:) - centers(j,:))^2 / (2 * sigma^2));
        end
    end

    output = Phi; % Just return Phi if we don't have target values
end

function [centers] = myKmeans(X, k)
    % Randomly initialize the cluster centers
    num_samples = size(X, 1);
    random_indices = randperm(num_samples, k);
    centers = X(random_indices, :);

    % Initialize variables
    cluster_assignment = zeros(num_samples, 1);
    max_iters = 100;
    iter = 0;

    while iter < max_iters
        iter = iter + 1;

        % Assign each sample to the nearest center
        for i = 1:num_samples
            distances = sum((X(i, :) - centers) .^ 2, 2);
            [~, min_index] = min(distances);
            cluster_assignment(i) = min_index;
        end

        % Update centers
        new_centers = zeros(size(centers));
        for j = 1:k
            cluster_points = X(cluster_assignment == j, :);
            if ~isempty(cluster_points)
                new_centers(j, :) = mean(cluster_points, 1);
            else
```

```
                % Reinitialize empty cluster
                new_centers(j, :) = X(randi(num_samples), :);
            end
        end

        % Check for convergence
        if all(new_centers == centers)
            break;
        end

        centers = new_centers;
    end
end

function [M, V, G, B, A] = robot_dynamics(q, q_dot)
    % Robot parameters
    m = 15; % kg
    I = 5; % kg*m^2
    R = 0.15; % m
    r = 0.05; % m
    d = 0.1; % m

    % State variables
    theta = q(3);
    theta_dot = q_dot(3);

    % Inertia matrix
    M = [m 0 m*d*sin(theta);...
        0 m -m*d*cos(theta);...
        m*d*sin(theta) -m*d*cos(theta) I];

    % Coriolis and centrifugal matrix
    V = [m*d*cos(theta)*theta_dot^2;...
         m*d*sin(theta)*theta_dot^2;...
         0];

    % Gravity vector
    G = [0; 0; 0]; % Assuming no gravity effects in x and y directions

    % Input matrix
    B = (1/r)*[cos(theta) cos(theta);...
        sin(theta) sin(theta) ;...
        R -R];

    A = [-m*sin(theta)*(q_dot(1)*cos(theta) + q_dot(2)*sin(theta))*theta_dot;...
        m*cos(theta)*(q_dot(1)*cos(theta) + q_dot(2)*sin(theta))*theta_dot;...
        -d*m*(q_dot(1)*cos(theta) + q_dot(2)*sin(theta))*theta_dot];

    S = [sin(theta) -d*cos(theta);cos(theta) -d*sin(theta);0 1];
end

function state = odefcn(t,x,T)

    state = zeros(6,1);

    % Robot parameters
    m = 15; % kg
    I = 5; % kg*m^2
    R = 0.15; % m
```

```
    r = 0.05; % m
    d = 0.1; % m

    % State variables
    theta = x(3);
    theta_dot = x(6);

    % Inertia matrix
    M = [m 0 m*d*sin(theta);...
        0 m -m*d*cos(theta);...
        m*d*sin(theta) -m*d*cos(theta) I];

    % Coriolis and centrifugal matrix
    V = [m*d*cos(theta)*theta_dot^2;...
         m*d*sin(theta)*theta_dot^2;...
         0];

    % Gravity vector
    G = [0; 0; 0]; % Assuming no gravity effects in x and y directions

    % Input matrix
    B = (1/r)*[cos(theta) cos(theta);...
        sin(theta) sin(theta) ;...
        R -R];

    A = [-m*sin(theta)*(x(4)*cos(theta) + x(5)*sin(theta))*theta_dot;...
        m*cos(theta)*(x(4)*cos(theta) + x(5)*sin(theta))*theta_dot;...
        -d*m*(x(4)*cos(theta) + x(5)*sin(theta))*theta_dot];


    eqn = pinv(M)*(B*T(end,:)' + A - G - V);

    state(1) = x(4);
    state(2) = x(5);
    state(3) = x(6);

    state(4) = eqn(1,1);
    state(5) = eqn(2,1);
    state(6) = eqn(3,1);

end
```

۲-۷- مسیر مربعی

```
%% Clearing Existing Data
clc
clear all
close all

%% Mobile Robot Parameters
R = 0.05; % Wheel radius
L = 0.1; % Distance between wheels
r = 0.05; % Radius of each wheel
m = 15;   % Robot mass
I = 5;    % Moment of inertia

%% Simulation parameters
sample_time = 1;       % Reducing the sample time can cause higher CPU usage
```

```
SimulationTime = 100;     % Change this if you want to change the simulation
duration
counter = 0;
time = 0:sample_time:SimulationTime;

%% Initial state
q = [0, 0, 0];  % Initial state [x_c, y_c, theta]
q_dot = [0, 0, 0];          % Initial linear velocity
omega = 0;      % Initial angular velocity

%% Parameters of the Circular Trajectory
a = 2;       % Circle Center in meters (X Axis)
c = 2;       % Circle Center in meters (Y Axis)
b = 2;       % Radius in meters
omega_r = 0.1; % Angular velocity for theta reference

%% Desired Trajectory (You can Change The Desired Trajectory)
% Define parameters for the square trajectory
side_length = 5;   % side length of the square
num_points = 200;  % number of points in the trajectory
points_per_side = num_points / 4;  % points per side

% Generate the x and y coordinates for each side of the square
x = [];
y = [];
theta = [];

% Bottom side (moving right)
x = [x, linspace(0, side_length, points_per_side)];
y = [y, zeros(1, points_per_side)];

% Right side (moving up)
x = [x, side_length * ones(1, points_per_side)];
y = [y, linspace(0, side_length, points_per_side)];

% Top side (moving left)
x = [x, linspace(side_length, 0, points_per_side)];
y = [y, side_length * ones(1, points_per_side)];


% Left side (moving down)
x = [x, zeros(1, points_per_side)];
y = [y, linspace(side_length, 0, points_per_side)];
for i = 1:size(x,2)
    if(x(i) == 0 && y(i) > 0)
        theta(1,i) = 0;
    elseif(x(i) == 0 && y(i) < 0)
        theta(1,i) = -pi;
    else
        theta(1,i) = atan(y/x);
    end
end

q_ref = [x', y', theta'];

q_dot_ref(1,:) = [0 0 0];
for i = 2:size(q_ref,1)
    q_dot_ref(i,1) = (q_ref(i,1) - q_ref(i-1,1))/sample_time;
    q_dot_ref(i,2) = (q_ref(i,2) - q_ref(i-1,2))/sample_time;
```

```
    q_dot_ref(i,3) = (q_ref(i,3) - q_ref(i-1,3))/sample_time;
end

q_ddot_ref(1,:) = [0 0 0];
for i = 2:size(q_ref,1)
    q_ddot_ref(i,1) = (q_dot_ref(i,1) - q_dot_ref(i-1,1))/sample_time;
    q_ddot_ref(i,2) = (q_dot_ref(i,2) - q_dot_ref(i-1,2))/sample_time;
    q_ddot_ref(i,3) = (q_dot_ref(i,3) - q_dot_ref(i-1,3))/sample_time;
end

plot(q_ref(:,1),q_ref(:,2))
%Tobe or not To be
q = q_ref(1,:);
q_dot = q_dot_ref(1,:);

%% Improved Training Data Generation
K = 1;
iterations = 100;
lambda = 0.08;
max_K = 20;
for i = 1:size(q_ref,1)
    e = q_ref(i,:) - q;

    q_all(i,:) = q;
    q_dot_all(i,:) = q_dot;
    e_all(i,:) = e;
    X_train{i} = [q_ref(1:i,:),q_dot_ref(1:i,:),e_all(1:i,:),q_dot_all(1:i,:)];

    if(K > max_K)
        K = max_K;
    end
    centers = myKmeans(X_train{i}, K);

    max_d = 0;
    for temp = 1:K
        for temp2 = 1:K
            if(norm(centers(temp2)-centers(temp)) > max_d)
                max_d = norm(centers(temp2)-centers(temp));
            end
        end
    end
    % getting sigmas according to handouts and the book with K-means and RLS
    sigma = max_d*max(max(q_ref))/sqrt(2*K)  ;

    % this function is defined at the end of the file
    [out, Phi] = rbf_HL(X_train{i}, centers, sigma);

    % adding bias to the weights
    Phi = [Phi , ones(size(X_train{i},1),1)];



    P = lambda * eye(K + 1);
    w = zeros(K + 1, 2); % Initialize weight matrix for each output dimension

    for n = 1:i
        % Update P matrix
        P = P - (P * Phi(n,:)' * Phi(n,:) * P) / (1 + Phi(n,:) * P * Phi(n,:)');
```

```
        % Calculate gain vector
        g = P * Phi(n,:)';

        % Prediction error
        [M, V, G, B, A] = robot_dynamics(q_ref(i,:), q_dot_ref(i,:));
        tau_ref = pinv(B)*(M*q_ddot_ref(i,:)' + V + G - A);
        pre = tau_ref' - Phi(n,:) * w;

        % Update weights
        w = w + g * pre;
    end

    % Calculate mean squared error
    T = Phi * w;
    if(i>1)
        [t,x] = ode45(@(t,x) odefcn(t,x,T), sample_time*[i-1 i], [q_ref(i-
1,:)';q_dot_ref(i-1,:)']);
        q = ([x(length(t),1),x(length(t),2),x(length(t),3)]);
        q_dot = [x(length(t),4),x(length(t),5),x(length(t),6)];
    end

end

%% Plotting Output Data
figure;
hold on
plot(q_ref(1:size(q_all,1),1),q_ref(1:size(q_all,1),2),'b',LineWidth=2)
plot(q_all(:,1),q_all(:,2),"r--",LineWidth=2);
legend(["reference" "predicted"]);
hold off

%% function definitions
function [output, Phi] = rbf_HL(X, centers, sigma)
    num_data = size(X, 1);
    num_centers = size(centers, 1);
    Phi = zeros(num_data, num_centers);

    for i = 1:num_data
        for j = 1:num_centers
            Phi(i, j) = exp(-norm(X(i,:) - centers(j,:))^2 / (2 * sigma^2));
        end
    end

    output = Phi; % Just return Phi if we don't have target values
end

function [centers] = myKmeans(X, k)
    % Randomly initialize the cluster centers
    num_samples = size(X, 1);
    random_indices = randperm(num_samples, k);
    centers = X(random_indices, :);

    % Initialize variables
    cluster_assignment = zeros(num_samples, 1);
    max_iters = 100;
    iter = 0;

    while iter < max_iters
        iter = iter + 1;
```

```
        % Assign each sample to the nearest center
        for i = 1:num_samples
            distances = sum((X(i, :) - centers) .^ 2, 2);
            [~, min_index] = min(distances);
            cluster_assignment(i) = min_index;
        end

        % Update centers
        new_centers = zeros(size(centers));
        for j = 1:k
            cluster_points = X(cluster_assignment == j, :);
            if ~isempty(cluster_points)
                new_centers(j, :) = mean(cluster_points, 1);
            else
                % Reinitialize empty cluster
                new_centers(j, :) = X(randi(num_samples), :);
            end
        end

        % Check for convergence
        if all(new_centers == centers)
            break;
        end

        centers = new_centers;
    end
end

function [M, V, G, B, A] = robot_dynamics(q, q_dot)
    % Robot parameters
    m = 15; % kg
    I = 5; % kg*m^2
    R = 0.15; % m
    r = 0.05; % m
    d = 0.1; % m

    % State variables
    theta = q(3);
    theta_dot = q_dot(3);

    % Inertia matrix
    M = [m 0 m*d*sin(theta);...
        0 m -m*d*cos(theta);...
        m*d*sin(theta) -m*d*cos(theta) I];

    % Coriolis and centrifugal matrix
    V = [m*d*cos(theta)*theta_dot^2;...
         m*d*sin(theta)*theta_dot^2;...
         0];

    % Gravity vector
    G = [0; 0; 0]; % Assuming no gravity effects in x and y directions

    % Input matrix
    B = (1/r)*[cos(theta) cos(theta);...
        sin(theta) sin(theta) ;...
        R -R];
```

```
    A = [-m*sin(theta)*(q_dot(1)*cos(theta) + q_dot(2)*sin(theta))*theta_dot;...
        m*cos(theta)*(q_dot(1)*cos(theta) + q_dot(2)*sin(theta))*theta_dot;...
        -d*m*(q_dot(1)*cos(theta) + q_dot(2)*sin(theta))*theta_dot];

    S = [sin(theta) -d*cos(theta);cos(theta) -d*sin(theta);0 1];
end

function state = odefcn(t,x,T)

    state = zeros(6,1);

    % Robot parameters
    m = 15; % kg
    I = 5; % kg*m^2
    R = 0.15; % m
    r = 0.05; % m
    d = 0.1; % m

    % State variables
    theta = x(3);
    theta_dot = x(6);

    % Inertia matrix
    M = [m 0 m*d*sin(theta);...
        0 m -m*d*cos(theta);...
        m*d*sin(theta) -m*d*cos(theta) I];

    % Coriolis and centrifugal matrix
    V = [m*d*cos(theta)*theta_dot^2;...
         m*d*sin(theta)*theta_dot^2;...
         0];

    % Gravity vector
    G = [0; 0; 0]; % Assuming no gravity effects in x and y directions

    % Input matrix
    B = (1/r)*[cos(theta) cos(theta);...
        sin(theta) sin(theta) ;...
        R -R];

    A = [-m*sin(theta)*(x(4)*cos(theta) + x(5)*sin(theta))*theta_dot;...
        m*cos(theta)*(x(4)*cos(theta) + x(5)*sin(theta))*theta_dot;...
        -d*m*(x(4)*cos(theta) + x(5)*sin(theta))*theta_dot];


    eqn = pinv(M)*(B*T(end,:)' + A - G - V);

    state(1) = x(4);
    state(2) = x(5);
    state(3) = x(6);

    state(4) = eqn(1,1);
    state(5) = eqn(2,1);
    state(6) = eqn(3,1);

end
```

۳-۷- مسیر دایره‌ای با در نظر گرفتن عوامل مختلف

```
%% Clearing Existing Data
clc
clear all
close all

%% Mobile Robot Parameters

    m = 30; % kg
    I = 10; % kg*m^2
    R = 0.3; % m
    r = 0.1; % m
    d = 0.2; % m

%% Simulation parameters
sample_time = 1;       % Reducing the sample time can cause higher CPU usage
SimulationTime = 150;     % Change this if you want to change the simulation
duration
counter = 0;
time = 0:sample_time:SimulationTime;


%% Parameters of the Circular Trajectory
a = 2;       % Circle Center in meters (X Axis)
c = 2;       % Circle Center in meters (Y Axis)
b = 2;       % Radius in meters
omega_r = 0.1; % Angular velocity for theta reference

%% Desired Trajectory (You can Change The Desired Trajectory)
x_ref = a + b * cos(omega_r * time);
y_ref = c + b * sin(omega_r * time);
theta_ref = omega_r * time;

q_ref = [x_ref; y_ref; theta_ref]';

x_dot_ref = -b * omega_r * sin(omega_r * time);
y_dot_ref =  b * omega_r * cos(omega_r * time);
omega_ref = omega_r * ones(size(time));

q_dot_ref = [x_dot_ref; y_dot_ref; omega_ref]';

x_ddot_ref = -b * omega_r^2 * cos(omega_r * time);
y_ddot_ref = -b * omega_r^2 * sin(omega_r * time);
alpha_ref  = zeros(size(time));

q_ddot_ref = [x_ddot_ref; y_ddot_ref; alpha_ref]';


%Tobe or not To be
q = q_ref(1,:);
q_dot = q_dot_ref(1,:);

%% Improved Training Data Generation
K = 1;
iterations = 100;
lambda = 0.08;
max_K = 20;
```

```
tau_max = 0;
coeff = 1;
tic
for i = 1:size(q_ref,1)
    e = q_ref(i,:) - q;

    q_all(i,:) = q;
    q_dot_all(i,:) = q_dot;
    e_all(i,:) = e;
    X_train{i} = [q_ref(1:i,:),q_dot_ref(1:i,:),e_all(1:i,:),q_dot_all(1:i,:)];

    if(K > max_K)
        K = max_K;
        % uncomment the line below for experiment
        % coeff = 2;
    end

    centers = myKmeans(X_train{i}, K);

    max_d = 0;
    for temp = 1:K
        for temp2 = 1:K
            if(norm(centers(temp2)-centers(temp)) > max_d)
                max_d = norm(centers(temp2)-centers(temp));
            end
        end
    end
    % getting sigmas according to handouts and the book with K-means and RLS
    sigma = max_d*max(max(q_ref))/sqrt(2*K)  ;

    % this function is defined at the end of the file
    [out, Phi] = rbf_HL(X_train{i}, centers, sigma);

    % adding bias to the weights
    Phi = [Phi , ones(size(X_train{i},1),1)];



    P = lambda * eye(K + 1);
    w = zeros(K + 1, 2); % Initialize weight matrix for each output dimension

    for n = 1:i
        % Update P matrix
        P = P - (P * Phi(n,:)' * Phi(n,:) * P) / (1 + Phi(n,:) * P * Phi(n,:)');

        % Calculate gain vector
        g = P * Phi(n,:)';

        % Prediction error
        [M, V, G, B, A] = robot_dynamics(q_ref(i,:), q_dot_ref(i,:),coeff);

        tau_ref = pinv(B)*(M*q_ddot_ref(i,:)' + V + G - A);
        pre = tau_ref' - Phi(n,:) * w;
        % Update weights
        w = w + g * pre;
    end

    % Calculate mean squared error
```

```
    tau_ref_all(:,i) = tau_ref;
    T = Phi * w;
    T_all(:,i) = T(end,:)';
    if(i>1)
        [t,x] = ode45(@(t,x) odefcn(t,x,tau_ref,coeff), sample_time*[i-1 i],
[q_ref(i,:)';q_dot_ref(i,:)']);
        q = ([x(length(t),1),x(length(t),2),x(length(t),3)]);
        q_dot = [x(length(t),4),x(length(t),5),x(length(t),6)];
    end

    %uncomment below lines for experiment
    % if mod(i,20) == 0
    %     q = [7 8 0];
    % end
    %
    % q = awgn(q,20);

end

toc
%% Plotting Output Data
figure;
hold on
plot(q_ref(1:size(q_all,1),1),q_ref(1:size(q_all,1),2),'b',LineWidth=2)
plot(q_all(:,1),q_all(:,2),"r--",LineWidth=2);
legend(["reference" "predicted"]);
hold off

%% function definitions
function [output, Phi] = rbf_HL(X, centers, sigma)
    num_data = size(X, 1);
    num_centers = size(centers, 1);
    Phi = zeros(num_data, num_centers);

    for i = 1:num_data
        for j = 1:num_centers
            Phi(i, j) = exp(-norm(X(i,:) - centers(j,:))^2 / (2 * sigma^2));
        end
    end

    output = Phi; % Just return Phi if we don't have target values
end

function [centers] = myKmeans(X, k)
    % Randomly initialize the cluster centers
    num_samples = size(X, 1);
    random_indices = randperm(num_samples, k);
    centers = X(random_indices, :);

    % Initialize variables
    cluster_assignment = zeros(num_samples, 1);
    max_iters = 100;
    iter = 0;

    while iter < max_iters
        iter = iter + 1;

        % Assign each sample to the nearest center
        for i = 1:num_samples
```

```
            distances = sum((X(i, :) - centers) .^ 2, 2);
            [~, min_index] = min(distances);
            cluster_assignment(i) = min_index;
        end

        % Update centers
        new_centers = zeros(size(centers));
        for j = 1:k
            cluster_points = X(cluster_assignment == j, :);
            if ~isempty(cluster_points)
                new_centers(j, :) = mean(cluster_points, 1);
            else
                % Reinitialize empty cluster
                new_centers(j, :) = X(randi(num_samples), :);
            end
        end

        % Check for convergence
        if all(new_centers == centers)
            break;
        end

        centers = new_centers;
    end
end

function [M, V, G, B, A] = robot_dynamics(q, q_dot,coeff)

% Robot parameters

    m = coeff*30; % kg
    I = 10; % kg*m^2
    R = 0.3; % m
    r = 0.1; % m
    d = 0.2; % m

    % State variables
    theta = q(3);
    theta_dot = q_dot(3);

    % Inertia matrix
    M = [m 0 m*d*sin(theta);...
        0 m -m*d*cos(theta);...
        m*d*sin(theta) -m*d*cos(theta) I];

    % Coriolis and centrifugal matrix
    V = [m*d*cos(theta)*theta_dot^2;...
         m*d*sin(theta)*theta_dot^2;...
         0];

    % Gravity vector
    G = [0; 0; 0]; % Assuming no gravity effects in x and y directions

    % Input matrix
    B = (1/r)*[cos(theta) cos(theta);...
        sin(theta) sin(theta) ;...
        R -R];

    A = [-m*sin(theta)*(q_dot(1)*cos(theta) + q_dot(2)*sin(theta))*theta_dot;...
```

```
        m*cos(theta)*(q_dot(1)*cos(theta) + q_dot(2)*sin(theta))*theta_dot;...
        -d*m*(q_dot(1)*cos(theta) + q_dot(2)*sin(theta))*theta_dot];

end

function state = odefcn(t,x,T,coeff)

    state = zeros(6,1);

    % Robot parameters
    m = coeff*30; % kg
    I = 10; % kg*m^2
    R = 0.3; % m
    r = 0.1; % m
    d = 0.2; % m

    % State variables
    theta = x(3);
    theta_dot = x(6);

    % Inertia matrix
    M = [m 0 m*d*sin(theta);...
        0 m -m*d*cos(theta);...
        m*d*sin(theta) -m*d*cos(theta) I];

    % Coriolis and centrifugal matrix
    V = [m*d*cos(theta)*theta_dot^2;...
         m*d*sin(theta)*theta_dot^2;...
         0];

    % Gravity vector
    G = [0; 0; 0]; % Assuming no gravity effects in x and y directions

    % Input matrix
    B = (1/r)*[cos(theta) cos(theta);...
        sin(theta) sin(theta) ;...
        R -R];

    A = [-m*sin(theta)*(x(4)*cos(theta) + x(5)*sin(theta))*theta_dot;...
        m*cos(theta)*(x(4)*cos(theta) + x(5)*sin(theta))*theta_dot;...
        -d*m*(x(4)*cos(theta) + x(5)*sin(theta))*theta_dot];


    eqn = pinv(M)*(B*T + A - G - V);

    state(1) = x(4);
    state(2) = x(5);
    state(3) = x(6);

    state(4) = eqn(1,1);
    state(5) = eqn(2,1);
    state(6) = eqn(3,1);

end
```