Walking Robot Gait Optimization Script

This script optimizes the walking gait of a robot using a genetic algorithm (GA).

In this example, a gait is defined as trajectory of waypoints for the reference hip, knee, and ankle joint angles. These waypoints are evenly spaced in time over a predefined gait period and repeated for the duration of the simulation.

Note that there are other approaches for using optimization on walking gaits. For example, you can optimize

- Foot placement position and orientation, and solve inverse kinematics (IK) to get joint angles
- Parameters (gains, thresholds, etc.) of a closed-loop controller or policy

Set initial parameters

Open the optimization model

```
mdlName = 'walkingRobotOptim'; % Main model
open_system(mdlName);
```

Flags to speed up simulation (strongly recommended unless you are testing)

```
accelFlag = true;     % Compile the model to run a single simulation faster
parallelFlag = false;  % Use parallel computing on multiple cores on your machine
or on a cluster or cloud, if available
```

Joint actuator type for optimization: 1 = motion | 2 = torque | 3 = motor

```
actuatorType = 1;
```

Define the number of trajectory points and total gait time (each point is evenly spaced within this time)

```
numPoints = 6;      % Number of joint angle points
gait_period = 1.5;  % Period of walking gait [s]
```

To reduce the search space, scale the angle waypoints and solve the optimization algorithm with integer parameters. This scaling factor is from degrees to an integer.

```
scalingFactor = 2.5;
```

Create initial conditions to seed the initial population for optimization. Alternatively you can load from one of the presaved MAT-files provided in the `SavedResults` folder, as long as you use the scaling factor below to convert the gait waypoints.

```
p0 = zeros(1,numPoints*3);  % Create zero motion initial conditions
```

Set optimization options

The options for the genetic algorithm are defined using the `optimoptions` function.

```
opts = optimoptions('ga');
opts.Display = 'iter';
opts.MaxGenerations = 100;
opts.PopulationSize = 100;
opts.InitialPopulationMatrix = repmat(p0,[5 1]); % Add copies of initial gait
opts.PlotFcn = @gaplotbestf; % Add progress plot of fitness function
opts.UseParallel = parallelFlag;
```

Set joint angle bounds and constraints

```
upperBnd = [45*ones(1,numPoints), ... % Hip limits
            90*ones(1,numPoints), ... % Knee limits
            45*ones(1,numPoints)] ... % Ankle limits
            /scalingFactor;
lowerBnd = [-45*ones(1,numPoints), ... % Hip limits
              0*ones(1,numPoints), ... % Knee limits
            -45*ones(1,numPoints)] ... % Ankle limits
            /scalingFactor;
```

Run commands to set up parallel/accelerated simulation

```
doSpeedupTasks;
```

Run optimization

Here we use the `ga` function from Global Optimization Toolbox to optimize the walking gait, with `simulateWalkingRobot` as the fitness function.

```
costFcn =
@(p)simulateWalkingRobot(p,mdlName,scalingFactor,gait_period,actuatorType);
disp(['Running optimization. Population: ' num2str(opts.PopulationSize) ...
x      ', Max Generations: ' num2str(opts.MaxGenerations)])
```

```
Running optimization. Population: 100, Max Generations: 100
```

```
[pFinal,reward] = ga(costFcn,numPoints*3,[],[],[],[], ...
                     lowerBnd,upperBnd,[],1:numPoints*3,opts);
```

```
Single objective optimization:
18 Variables
18 Integer variables

Options:
CreationFcn:       @gacreationuniformint
CrossoverFcn:      @crossoverlaplace
SelectionFcn:      @selectiontournament
MutationFcn:       @mutationpower

                                  Best           Mean         Stall
Generation        Func-count     Penalty        Penalty    Generations
    1                200          -1.802       -0.03314        0
    2                295          -1.802        -0.1436        1
```

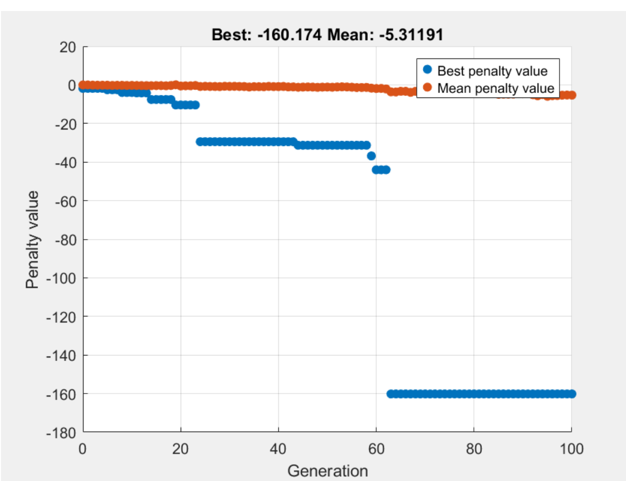
3	390	-1.802	-0.1786	2
4	485	-1.802	-0.1775	3
5	580	-2.533	-0.1741	0
6	675	-2.533	-0.2885	1
7	770	-2.533	-0.2262	2
8	865	-3.942	-0.2442	0
9	960	-3.942	-0.3143	1
10	1055	-3.942	-0.2623	2
11	1150	-4.168	-0.3474	0
12	1245	-4.168	-0.3639	1
13	1340	-4.168	-0.3984	2
14	1435	-7.581	-0.4012	0
15	1530	-7.581	-0.3778	1
16	1625	-7.581	-0.3935	2
17	1720	-7.581	-0.4519	3
18	1815	-7.581	-0.3224	4
19	1910	-10.42	0.0395	0
20	2005	-10.42	-0.5633	1
21	2100	-10.42	-0.4938	2
22	2195	-10.42	-0.474	3
23	2290	-10.42	-0.3193	4
24	2385	-29.45	-0.8315	0
25	2480	-29.45	-0.7372	1
26	2575	-29.45	-0.8162	2
27	2670	-29.45	-0.8374	3
28	2765	-29.45	-0.8926	4
29	2860	-29.45	-0.7327	5

Generation	Func-count	Best Penalty	Mean Penalty	Stall Generations
30	2955	-29.45	-0.7919	6
31	3050	-29.45	-0.7423	7
32	3145	-29.45	-0.8328	8
33	3240	-29.45	-0.8542	9
34	3335	-29.45	-1.102	10
35	3430	-29.45	-0.9211	11
36	3525	-29.45	-0.9338	12
37	3620	-29.45	-0.9714	13
38	3715	-29.45	-0.7938	14
39	3810	-29.45	-0.8871	15
40	3905	-29.45	-0.8902	16
41	4000	-29.45	-0.8328	17
42	4095	-29.45	-1.071	18
43	4190	-29.45	-1.039	19
44	4285	-31.29	-1.247	0
45	4380	-31.29	-1.149	1
46	4475	-31.29	-1.096	2
47	4570	-31.29	-1.156	3
48	4665	-31.29	-1.283	4
49	4760	-31.29	-1.168	5
50	4855	-31.29	-1.133	6
51	4950	-31.29	-1.222	7
52	5045	-31.29	-1.085	8
53	5140	-31.29	-1.018	9
54	5235	-31.29	-1.077	10
55	5330	-31.29	-1.207	11
56	5425	-31.29	-1.378	12
57	5520	-31.29	-1.364	13
58	5615	-31.29	-1.301	14
59	5710	-36.82	-1.712	0

Generation	Func-count	Best Penalty	Mean Penalty	Stall Generations
60	5805	-44.02	-1.902	0

61	5900	-44.02	-1.848	1
62	5995	-44.02	-2.098	2
63	6090	-160.2	-3.615	0
64	6185	-160.2	-3.678	1
65	6280	-160.2	-3.331	2
66	6375	-160.2	-3.33	3
67	6470	-160.2	-3.858	4
68	6565	-160.2	-3.38	5
69	6660	-160.2	-3.369	6
70	6755	-160.2	-3.545	7
71	6850	-160.2	-3.413	8
72	6945	-160.2	-3.391	9
73	7040	-160.2	-3.702	10
74	7135	-160.2	-3.631	11
75	7230	-160.2	-3.668	12
76	7325	-160.2	-3.454	13
77	7420	-160.2	-3.394	14
78	7515	-160.2	-3.861	15
79	7610	-160.2	-3.803	16
80	7705	-160.2	-3.455	17
81	7800	-160.2	-3.439	18
82	7895	-160.2	-3.41	19
83	7990	-160.2	-3.53	20
84	8085	-160.2	-3.598	21
85	8180	-160.2	-4.91	22
86	8275	-160.2	-4.815	23
87	8370	-160.2	-4.807	24
88	8465	-160.2	-4.781	25
89	8560	-160.2	-4.339	26

Generation	Func-count	Best Penalty	Mean Penalty	Stall Generations
90	8655	-160.2	-4.547	27
91	8750	-160.2	-4.534	28
92	8845	-160.2	-5.096	29
93	8940	-160.2	-5.66	30
94	9035	-160.2	-5.035	31
95	9130	-160.2	-5.924	32
96	9225	-160.2	-5.608	33
97	9320	-160.2	-5.537	34
98	9415	-160.2	-5.365	35
99	9510	-160.2	-5.269	36
100	9605	-160.2	-5.312	37



ga stopped because it exceeded options.MaxGenerations.

```
disp(['Final reward function value: ' num2str(-reward)])
```

```
Final reward function value: 160.1742
```

Analyze and save results

Convert from optimization integer search space to trajectories in radians

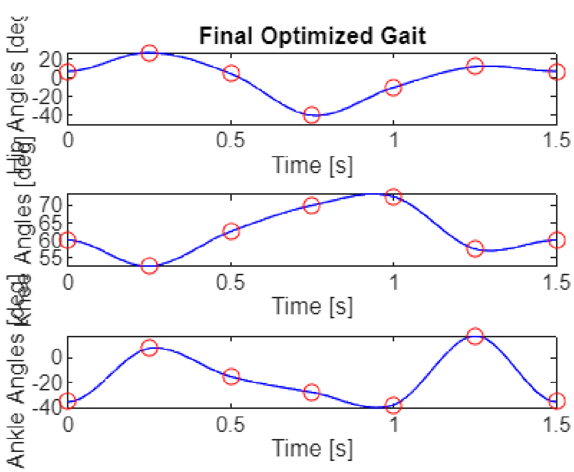
```
pScaled = scalingFactor*pFinal;
traj_times = linspace(0,gait_period,numPoints+1);
hip_motion = deg2rad([pScaled(1:numPoints) pScaled(1)]);
knee_motion = deg2rad([pScaled(numPoints+1:2*numPoints) pScaled(numPoints+1)]);
ankle_motion = deg2rad([pScaled(2*numPoints+1:3*numPoints) pScaled(2*numPoints+1)]);
```

Evaluate the trajectory at a few points for visualization

```
numTrajPoints = 101;
evalTimes = linspace(0,gait_period,numTrajPoints);
[q,hip_der,knee_der,ankle_der] =
createSmoothTrajectory(hip_motion,knee_motion,ankle_motion,gait_period,evalTimes);
```

Plot the resulting trajectory

```
figure
subplot(3,1,1)
plot(evalTimes,rad2deg(q(1,:)),'b-',traj_times,rad2deg(hip_motion),'ro');
title('Final Optimized Gait')
xlabel('Time [s]');
ylabel('Hip Angles [deg]');
subplot(3,1,2)
plot(evalTimes,rad2deg(q(2,:)),'b-',traj_times,rad2deg(knee_motion),'ro');
xlabel('Time [s]');
ylabel('Knee Angles [deg]');
subplot(3,1,3)
plot(evalTimes,rad2deg(q(3,:)),'b-',traj_times,rad2deg(ankle_motion),'ro');
xlabel('Time [s]');
ylabel('Ankle Angles [deg]');
```



Save results to a timestamped MAT-file

```
outFileName = ['optimizedData_' datestr(now,'ddmmmyy_HHMM')];
save(outFileName,'reward','gait_period','traj_times','hip_motion','knee_motion','ankle_motion');
```

Cleanup

Close the model and, if a parallel pool was created, delete it.

```
doCleanup = true;
if doCleanup
    bdclose(mdlName);
    if parallelFlag
        delete(gcp('nocreate'));
    end
end
```