Purdue

AAE 55000

Multidisciplinary Design Optimization

Final Project

Professor Crossley

Enrique Babio

Table of Contents

Aim and Scope

The aim of this document is to present the approach, methods and results applied to the optimization of a trajectory definition problem. The goal is not to describe the trajectory optimization formulation process, but the methods used to solve the minimum once the problem is formulated into a standard optimization approach.

Because of that this report is structured as follows:

The first section will consist of the optimization problem description. It states the optimization problem and discusses any insight of the problem from the mathematical and engineering point of view that may be useful for optimization. In this section we will finally describe the intended approach that will be followed for the optimization process.

The next section will focus on the discussion of the optimization process. The chosen methods implementations and solutions will be described. The preliminary results and its effect on modifying the problem formulation will be discussed. Also the problems of initial conditions and global minimality determination will be shown in detail. Finally, and taking all the previous the facts into account the optimization solution chosen is introduced.

Following, we will cover the most relevant facts about the solution. The optimal solution will be presented and discussed. Besides, some insightful results are provided to gain a better understanding on the optimization process and the obtained results.

Finally, some conclusions will be drawn on the solution optimality and the results obtained from the optimization process. The suitability of the optimization algorithms and approach followed for the particular problem will be also discussed.

The three annexes will cover all the details left by the main report. The engineering problem and the optimization formulation for it will be introduced and discussed. All the software used will be presented, and finally a more extended collection of results will be included.

Problem Description

The engineering problem consists in finding the smoothest trajectory for an airplane travelling through a 2D region with some No-Fly areas. The trajectory is defined through discrete-time states. The smoothness is measured through the jerk of the trajectory, which is the objective function to be minimized. The constraints for the trajectory are imposed by the No Fly zones as inequality constraints, and the dynamics of the airplane as equality constraints. Finally, the maneuverability limits are incorporated into the bounds. Expressed in the optimization formulation they are:

Design vector:

$$x = \phi_1, \psi_1, x_1, y_1, r_1, \phi_2, \psi_2, x_2, y_2, r_2, \dots, \phi_n, \psi_n, x_n, y_n, r_n$$

Objective function:

$$f(x) = \frac{1}{n}\sum_{i=0}^{n}\left(\frac{g}{1+\phi_i^2} r_i\right)^2$$

Inequality constraints:

$$g_{n(k-1)+i}(x) = r_k^2 - ((x_i - x_k)^2 + (y_i - y_k)^2) \leq 0 \qquad for\ i = 1, \dots, n-1 \quad k = 1, \dots, m$$

Equality constraints:

(Boundary conditions)

$$h_1(x) = \phi_1 = 0$$

$$h_2(x) = \psi_1 = 0$$

$$h_3(x) = x_1 = x_0$$

$$h_4(x) = y_1 = y_0$$

$$h_5(x) = r_1 = 0$$

$$h_6(x) = \phi_n = 0$$

$$h_7(x) = \psi_n = 0$$

$$h_8(x) = y_n = y_f$$

$$h_9(x) = r_n = 0$$

(Dynamics constraints)

$$h_{9+i}(x) = \phi_{i+1} - (\phi_i + \cos(\phi_i)\, r_i\, dt) = 0 \qquad for\ i = 1, \dots, n-1$$

$$h_{9+(n-1)+i}(x) = \psi_{i+1} - \left(\psi_i + \frac{g}{v}\tan(\phi_i)\, dt\right) = 0 \qquad for\ i = 1, \dots, n-1$$

$$h_{9+2(n-1)+i}(x) = x_{i+1} - (x_i + v\cos(\psi_i)\, dt) = 0 \qquad for\ i = 1, \dots, n-1$$

$$h_{9+3(n-1)+i}(x) = y_{i+1} - (y_i + v\sin(\psi_i)\, dt) = 0 \qquad for\ i = 1, \dots, n-1$$

Bounds:

$$-r_{max} \leq \ r_i \leq r_{max}$$

$$-\phi_{max} \leq \ \phi_i \leq \phi_{max}$$

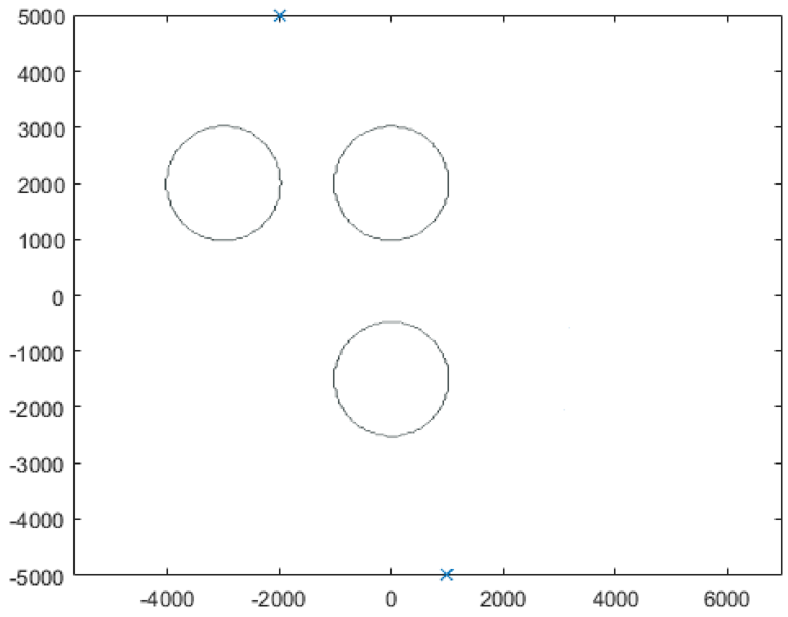
$$-\pi \ \leq \ \phi_i \leq \pi$$

$$-8 \cdot 10^3 \leq x_i \leq 8 \cdot 10^3$$

$$-8 \cdot 10^3 \leq y_i \leq 8 \cdot 10^3$$

For simplicity the parameters are expressed as symbolic variables, the actual values used in the problem can be checked at Annex A. The number of constraints is dependent on the choice of n and m: n is the number of points used to discretize the problem and m is the number of No Fly areas. For this problem is has been chosen to be $n = 25$ and $m = 3$. This means the number of inequality constraints is $nk = 75$ and the equality constraints add up to $9 + 4(n-1) = 105$.

This formulation will look into the possible trajectories joining from the initial, the lower blue point, to a final point in the vertical of the upper plotted point while avoiding the No Fly Zones, the plotted circles.



This means that there will be several local optimal trajectories depending on the way No Fly zones are avoided in terms of leaving the No Fly zones to the left or right.

From we have seen so far we have two important characteristics of the problem:

- There will be several local optima, meaning a global search capability is required
- The design space is highly constrained, meaning most random changes will not feasible. Derivative information will be highly appreciated to take into account the usability of changes.

No single method covered in class offers both characteristics so we will use a global non-smooth method and a direct constrained method. Because of the implementation complexity of the problem MATLAB will be used, so we will choose Genetic Algorithm as SQP as the methods from both categories as the priori candidate optimizers.

Optimization Discussion

Both algorithms were implemented using the methods covered in the homework. SQP was used by selecting the algorithm in MATLAB's fmincon, GA was setup using the GA_550.m file. The problem formulation used was the one above which was coded into a library of function and visualization tools, the implementation in detail is covered in Annex B. Here we will cover the steps followed to define convergence and global optimality.

The first goal is to determine the convergence of the algorithm. We start by covering the setup of the problem for SQP, we will describe the problem description in terms of derivatives, linearity of constraints and initial conditions to achieve convergence.

The first decision is to choose what kind of derivatives we will use. We have seen that, while any single function is not particularly complex, there is a huge number of them involved. Also, the number of functions involved changes with the size of the number of points used, this number was not known a priori (the initial guess was 10 points and it grew up to 20 as a compromise between expense and trajectory granularity). Therefore, numerical derivatives were used: they should slow down the process because of the high number of function evaluations, but it was considered more convenient than to develop a gradient formulation that adapted to changing sizes.

MATLAB's SQP allows for the user to differentiate between linear constraints and nonlinear constraints. A closer look at the constraints reveals that there are no linear inequality constraints, and in the equality constraints only the boundary conditions are linear. This means that only a small number of constraints are linear, so it has been considered that setting up the problem to use this few derivatives on a problem comprising 90 variables will not be really worth the gain.

Finally, the initial conditions are quite important. Since SQP linearize the constraints and most of the equations are nonlinear very likely convergence will only be achieved from initial trajectories that resemble feasible trajectories. The tentative initial trajectory was a straight line from the initial point to the tentative final point, this trajectory does not meet boundary conditions and steps over a No Fly zone. The initial results were positive, the algorithm converged and a possible optimal trajectory was returned.

Now we will cover our attempt to make Genetic Algorithm. Problem setup in this case is simpler because it does not take initial conditions and choices on constraints. All constraints were implemented as penalty functions on the trajectory function. The preliminary encoding chosen was: x_i, y_i with 15 bits (.5m resolution), ψ_i to 12 bits (0.1 deg), ϕ_i to 10 bits (0.1deg) and r to r_i to 10 bits (0.05deg/s). For 20 points this adds up to a chromosome length of 1240 bits. All other parameters were chosen following the guidelines used in class.

The initial tests were run with a limit of 100 generations to assess. The optimizer worked much more slowly than SQP just to perform this 100 initial generations taking over a minute to reach the limit. Several runs were performed and no patter was seen to be emerging, instead trajectories looked completely random. This could happen because of the huge number of constraints, so that no random point satisfies all of them (including 80 equality constraints) so the function evaluation is driven by penalties instead of performance. Increasing resolution by a factor 10 did not improve results and neither did reducing the penalties. Therefore, this method is considered to be not appropriate.

The preliminary runs show that SQP is capable of converging, but this is not possible for Genetic Algorithm. Also the conclusions obtained on Genetic Algorithm non-convergence would apply to any other Global method, the method we use needs to take into consideration the constraints or at least its derivatives to be able to fit all of them simultaneously, this will not be achieved by a heuristic method. This leaves with no other option to develop a through global search strategy that uses SQP as an optimizer.

After some preliminary tests a random initial trajectory description was found. A 1D random Fourier series was setup, then it was rotated and scaled such that it fits the initial point and the tentative final point. All ϕ_i and r_i were set to 0 and ψ_i to the heading of line passing through both points. Hundreds of runs were performed programmatically and new optimal trajectories were detected using the value of the objective function, if a new run differed in more than 2% from previous optima it was considered to be a new optimum. This was done to prevent numerical errors or stopping conditions based on tolerances to count as new optima.

Results

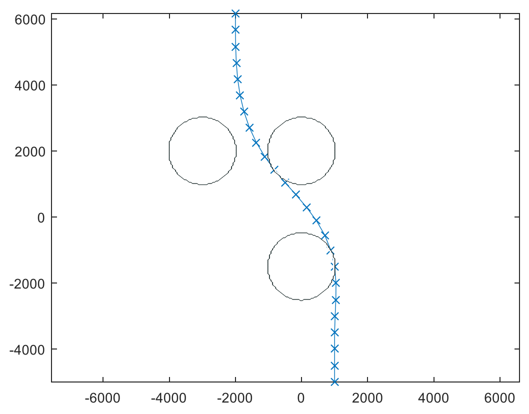
As it was foreseen several local minima were found. Because of the way the No Fly zones are laid out there are several groups of trajectories in terms of how the zones are left on the left or on the right. However, besides this combinatorial-like problem that defines families of trajectories, for each family several optimal trajectories, this means that the problem formulation also has small local minima.

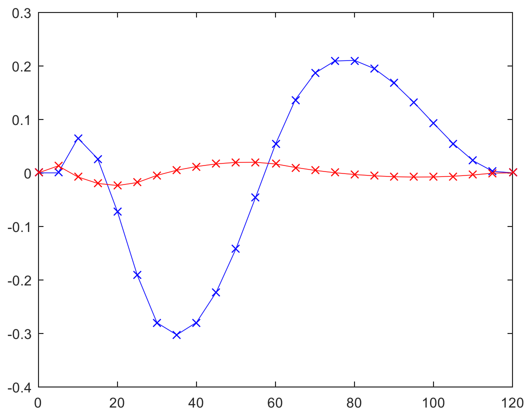
Finding some of these has only been able because of our problem approach, randomly-generated initial trajectories. However, this approach does not go any further than a random-search initial conditions. If there were minima whose convergence region was not reached by our initial state generator we would not have discovered them. It is likely that such thing happened, but we have focused on generating ‘close’ to straight trajectories whose jerk, i.e. objective function, is smaller. These trajectories initially advance towards the tentative final point which changes to accommodate the final distance traveled. The initial trajectory can also be considered to be demanding as many initial trajectories are not able to converge using SQP, this adds up to a third of the simulations.

All of this means that, overall, we can consider to have achieved the global optimal trajectory. We have focused on initial trajectories that deviated from the straight line rather than looping, this means that while we have not found all local minima we must have found all optima closer to the straight line which is the trajectory of least jerk.

We will now show some of the trajectories founds, the trajectories imply position and heading values, but also roll and roll rate values. Instead of elaborating on a table for the optimal design we will plot them in two figures: the 2D trajectory that provides information on x_i, y_i in meters and ψ_i values (the last one is the heading, which is implicit in terms the heading between a point and the next one), and the time series showing ϕ_i and r_i in radians and radians per second respectively. Finally, the outputs of the optimizer: objective function value, exit flag, iterations and function evaluations are displayed.

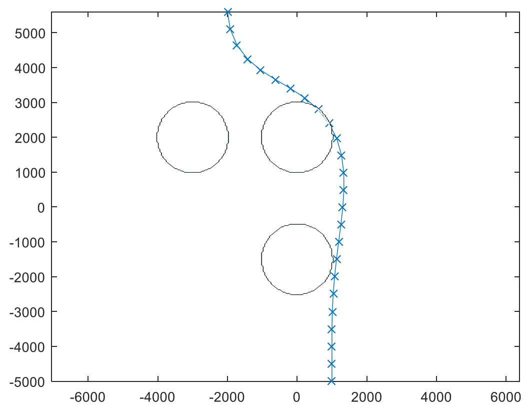
The following trajectories show the two most optimal trajectories which belong to two different families of trajectories. The optimal trajectory is the following:

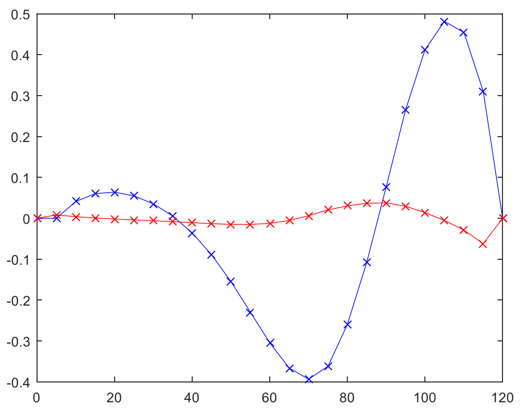




Parameter	$f(x^*)$	Exit flag	Iterations	Function evaluations
Values	0.0136	1	670	85387

The second most optimal trajectory is:





Parameter	$f(x^*)$	Exit flag	Iterations	Function evaluations
Values	0.0535	1	481	60732

More optimal results were found. For the detailed results please see Annex C.

Conclusions

In this section we will draw the conclusions on the problem formulation and the results obtained. First of all, we have tested to algorithms: SQP and Genetic Algorithm. We have observed that GA is does not properly for this problem. The high number of equality constraints plus penalizes global methods that relay on random changes or fixed increments, concluding that calculus information is necessary.

SQP has been successfully used with a robust initial trajectory generation to obtain follow a thorough multi start strategy. This generation is random-based and has been run for hundreds of times, this proves to be necessary as close minima were found but it has given a high confidence that the global optima was achieved.

Annex A: Problem Formulation

In this Annex A we will cover two topics: the first one is the process followed to arrive to the problem formulation and the second one are the parameters used for the exact problem being solved.

A.1. Problem formulation

Trajectory optimization is a problem that generally looks at optimizing a function rather optimizing parameters. In order to make the problem suitable for the methods seen in AAE50 we have discretized it. The original problem consists of:

Minimize:

$$\frac{1}{t_f - t_0}\int_{t_0}^{t_f}\left(\frac{g}{1+\phi(t)}r(t)\right)^2 \, dt$$

Subject to:

$$\dot{\phi}(t) = \cos\phi(t) \; r(t)$$

$$\dot{\psi}(t) = \frac{g}{v}\tan(\phi(t))$$

$$\dot{x}(t) = v\cos(\psi(t))$$

$$\dot{y}(t) = v\sin(\psi(t))$$

The boundary constraints and the bounds for the functions are identical to the same one we have used in the problem formulation. However, the problem above consists is a functional problem that does not match tools covered in AAE550. This can be worked around by discretizing the problem:

Minimize:

$$\sum_{i=0}^{n-1}\int_{t_0+T\cdot i}^{t_0+T\cdot(i+1)}\left(\frac{g}{1+\phi(t)}r(t)\right)^2 \, dt$$

Subject to:

$$\phi\big(t_0+T\cdot(i+1)\big) - \phi(t_0+T\cdot i) = \int_{t_0+T\cdot i}^{t_0+T\cdot(i+1)} \cos\phi(t) \; r(t) \, dt$$

$$\psi\big(t_0+T\cdot(i+1)\big) - \psi(t_0+T\cdot i) = \int_{t_0+T\cdot i}^{t_0+T\cdot(i+1)} \frac{g}{v}\tan(\phi(t)) \, dt$$

$$x\big(t_0+T\cdot(i+1)\big) - x(t_0+T\cdot i) = \int_{t_0+T\cdot i}^{t_0+T\cdot(i+1)} v\cos(\psi(t)) \, dt$$

$$y\big(t_0+T\cdot(i+1)\big) - y(t_0+T\cdot i) = \int_{t_0+T\cdot i}^{t_0+T\cdot(i+1)} v\sin(\psi(t)) \, dt$$

Now it is up to finding a numerical integration method based on discrete values of all the design variables. In our case we have the simplest available method, a rectangular integration. This yields the problem description covered in the main part of the report so it won't be reproduced here.

A.2. Problem parameters

Aircraft parameters

Number of points used for discretization: $N = 25$

Time increment between consecutive points: $dt = 5s$

Maximum roll angle (used as bound): $\phi_{max} = \frac{\pi}{4}$

Maximum roll angular rate: $r_{max} = \frac{\pi}{8}s^{-1}$

Acceleration of gravity: $g = 9.81m/s^2$

Aircraft speed (held constant): $v = 100m/s$

Initial Final Conditions

Initial x position: $x_0 = -5e3$

Initial y position: $y_0 = 1e3$

Final x position: $x_f = 5e3$

Final y position: $y_f = -2e3$

No Fly zones

Zone 1

Center: $x_{1c} = -1.5e3$ $y_{1c} = 0$
Radius: $r_1 = 1e3$

Zone 2

Center: $x_{1c} = 2e3$ $y_{1c} = 0$
Radius: $r_1 = 1e3$

Zone 3

Center: $x_{1c} = 2e3$ $y_{1c} = -3e3$
Radius: $r_1 = 1e3$

Annex B: MATLAB code

We will start by covering the functions that implement the problem formulation. Next, we will cover the optimization code itself. Finally, we will cover all the auxiliary code used for visualization and optimization logging.

B.1. Problem formulation

We will start with the implementation of the objective function. It has been implemented in such a way that detects the size of the input to allow for fast reconfiguration of the problem:

```
function f = fTotalJerk(x)
% Compute the mean squared jerk for a given trajectory

%% Handle input vector;
checkTrajectoryVector(x);

[trajectory, m, n] = vectorToTrajectory(x);

x = trajectory(1,:);
y = trajectory(2,:);
heading = trajectory(3,:);
roll = trajectory(4,:);
rollRate = trajectory(5,:);

%% Evaluate function

jerk = rollRate .* 9.81.* (1 + roll.^2);
jerkError = jerk.^2;
totalJerk = sum(jerkError);

f = 1/n .* totalJerk;

end
```

The next file implements the inequality constraints:

```
function g = gNoFlyConstraints(x)
% Compute the inequality constraints values for the plane's maneuverability
% limits
% All g is linear
%% Handle input vector;
checkTrajectoryVector(x);

[trajectory, m, n] = vectorToTrajectory(x);

x = trajectory(1,:);
y = trajectory(2,:);
heading = trajectory(3,:);
roll = trajectory(4,:);
rollRate = trajectory(5,:);

noFlyCenter = [-1.5e3     2e3     2e3;    0    0    -3e3];
```

```
noFlyRadius = [1e3      1e3     1e3];
%% Evaluate function
k = size(noFlyCenter,2);
g = zeros(n*k,1);
for i=1:k
    radius = (x-noFlyCenter(1,i)).^2 + (y-noFlyCenter(2,i)).^2;
    index = (i-1)*n+1;
    g(index:index+n-1) = (-radius + noFlyRadius(i).^2)';
end
end
```

The next two files implement the equality constraints. The first one contains the boundary conditions:

```
function h = hInitialFinalConditions(x, parameters)
% Compute the equality constraint value for the problem's initial and final
% conditions
% All h is linear
%% Handle input vector;
checkTrajectoryVector(x);

[trajectory, m, n] = vectorToTrajectory(x);

x = trajectory(1,:);
y = trajectory(2,:);
heading = trajectory(3,:);
roll = trajectory(4,:);
rollRate = trajectory(5,:);

%% Evaluate function

global initialPoint
global initialSatisfy
global finalPoint
global finalSatisfy
h = zeros(10,1);

% Initial conditions
h(1:5) = (initialSatisfy').*(trajectory(:,1) - initialPoint');

% Final conditions
h(6:10) = (finalSatisfy').*(trajectory(:,end) - finalPoint');
end
```

This second file implements the dynamics constraints:

```
function h = hKinematicConstraints(x, parameters)
% Compute the equality constraints value for the problem's kinematics
% The n-1 first h elements are linear
%% Handle input vector;
checkTrajectoryVector(x);

[trajectory, m, n] = vectorToTrajectory(x);
```

```
x = trajectory(1,:);
y = trajectory(2,:);
heading = trajectory(3,:);
roll = trajectory(4,:);
rollRate = trajectory(5,:);

v = parameters.v;
dt = parameters.dt;

%% Evaluate function
constraintsPerInstant = 4;
h = zeros((n-1)*constraintsPerInstant,1);
for i=1:n-1
    index = n-1;
    h(i) = roll(i+1) - (roll(i) + cos(roll(i))*rollRate(i)*dt);
    h(index+i) = heading(i+1) - (heading(i) + (9.81/v)*tan(roll(i))*dt);
    h(2*index+i) = x(i+1) - (x(i) + v*cos(heading(i))*dt);
    h(3*index+i) = y(i+1) - (y(i) + v*sin(heading(i))*dt);
end

end
```

The following files are only wrappers. They are only function handles that group functions by their function: objective, inequality and equality constraints. These only serve code cleanness, they were initially written to fit possible variations of the problem formulation:

Objective function wrapper:

```
function f = fObjectiveTrajectory(x, parameters)
f = fTotalJerk(x);
end
```

Constraints wrappers (in this case the maneuverability were implemented as constraints):

```
function g = gConstraintsTrajectory(x, parameters)
% Group all inequality constraints

g1 = gManeuverabilityConstraints(x, parameters);
g2 = gNoFlyConstraints(x);

% They can be implemented as bounds
g = [g2];

end
```

```
function h = hConstraintsTrajectory(x, parameters)
% Group all inequality constraints

h1 = hInitialFinalConditions(x, parameters);
h2 = hKinematicConstraints(x, parameters);
```

```
h = [h1; h2];

end
```

Finally, a wrapper includes both types of constraints for the calling convention of fmincon:

```
function [g,h] =  ghConstraintsTrajectory(x, parameters)
% Group all constraints

g = gConstraintsTrajectory(x, parameters);
h = hConstraintsTrajectory(x, parameters);

end
```

B.2. Optimization Problem

This script performs most all of the optimization functionality.

First of all it is possible to set the runtime options: the number of points used to discretize the trajectory, which was set to 25 for the report. It also allows for sequential discretization; it runs several optimization increasing the granularity of the trajectory to allow for convergence of more detailed trajectory. Finally, the number of multistart points can be chosen.

Next, the initial conditions can be chosen and which boundary conditions need to be enforced (for example a final point has to be chosen but only the lateral coordinate is enforced). The trajectory is initialized for each loop using the initial and final points just set. The bounds are set to enforce the maneuverability defined in the problem formulation and the possible trajectory coordinates are chosen is a function of the initial and final points.

The optimization is run using the initial conditions and bounds defined before and the previous implementation of the functions and their wrapper are called.

Finally, if the convergence is successful, the minima is tested to be new in terms of the objective function value. If it is a new it is logged or displayed in terms of the runtime options.

```
%% Trajectory Optimization

% Optimize a trajectory in terms of flight error and jerk

%% Setup workspace

close all
clear
clear global
addpath(genpath(pwd))
clc

%% Runtime options

nOfPoints = 25;
```

```
display = 1;
log = 1;
sequentialRuns = 0; % 0 for 1 run
multiStart = 100;

%% Initialize Problem Parameters

nOfPoints = (nOfPoints) / (2.^sequentialRuns);

d =4e3;
maxRoll     = pi/4;  % 45 degrees
maxRollRate = pi/8;  % 22.5 deg/s

f0 = 'tf';              v0 = 125;
f1 = 'dt';              v1 = v0/nOfPoints;
f2 = 'v';               v2 = 100;
f3 = 'maxRoll';         v3 = maxRoll;
f4 = 'maxRollRate';     v4 = maxRollRate;

global parameters
parameters = struct(f0,v0, f1,v1, f2,v2, f3,v3, f4,v4);

% Set boundary conditions
global initialPoint
initialPoint    = [-5e3,    1e3,    0,      0, 0];
global initialSatisfy
initialSatisfy  = [1,       1,      1,      1, 1];
global finalPoint
finalPoint      = [5e3,     -2e3,   0,      0, 0];
global finalSatisfy
finalSatisfy    = [0,       1,      1,      1, 1];

global trajName

%% Handle multistart
checkForNewMinima(0, 1);
if(multiStart < 1 || multiStart > 10000)
    multiStart=0;
end
runsLeft = multiStart + 1;
for dummy = 1:runsLeft
    tic
    disp(['minimization attempt number:' num2str(dummy)])
    dt = parameters.dt;

    %% Generate Initial Trajectory

    trajectory = [initialPoint; finalPoint]';

    % Deterministically
%     initialTrajectory = interpolateTrajectory(trajectory, parameters.dt,
nOfPoints-2);
    % Randomly
    initialTrajectory = generateInitialTrajectory(trajectory, nOfPoints);
    trajectory = initialTrajectory;

    sequentialRunsLeft = sequentialRuns;
    while(sequentialRunsLeft >= 0)
        if(sequentialRunsLeft ~= sequentialRuns)

            % Interpolate trajectory
            [trajectory, dt] = interpolateTrajectory(trajectory, dt, 1);
        end
        sequentialRunsLeft = sequentialRunsLeft - 1;
        n = size(trajectory,2);

        % Initial State
        x0 = trajectoryToVector(trajectory);

        %% Setup problem

        % Bounds
        lb = [-2*d*ones(n,1);-2*d*ones(n,1);   -pi*ones(n,1);    -
maxRoll*ones(n,1);   -maxRollRate*ones(n,1)];
        ub = [2*d*ones(n,1);  2*d*ones(n,1);    pi*ones(n,1);
maxRoll*ones(n,1);    maxRollRate*ones(n,1)];

        % Initial state

        x = x0;
        % Options
        maxFunEvals = inf;
        maxIter = 2000;
        options = optimoptions('fmincon','Algorithm','sqp',
'Display','none','MaxFunEvals',maxFunEvals,'MaxIter',maxIter);

        % Functions
        f = @(x) fObjectiveTrajectory(x, parameters);
        ghNL = @(x) ghConstraintsTrajectory(x, parameters);

        %% Solve problem

        [x, fVal, exitflag, output, lambda, fGrad] = ...
            fmincon(f , x0, [], [], [],[], lb, ub, ghNL, options);
        if(exitflag ~= 1)
            warning(['Local minimum may not be achieved: exitflag ='
num2str(exitflag)]);
        end

        trajectory = vectorToTrajectory(x);
    end

    %% Handle results
    new=0;
    if(exitflag>0)
        new = checkForNewMinima(fVal, 0);
        if(new)
            disp('New local minimum found');
            fVal = fObjectiveTrajectory(x, parameters);
            [g, h] = ghConstraintsTrajectory(x, parameters);

            date = char(datetime('now','Format','yyyyMMdd''_''HHmmss'));

            % display
            if(display)
                trajName = ['Initial_Trajectory_' date];
                plotTrajectory(initialTrajectory, parameters.dt, 0, log);
                cf = gcf();
                plotConstraints(cf.Number);

                trajName = ['Final_Trajectory_' date];
                plotTrajectory(trajectory, parameters.dt, 0, log);
                cf = gcf();
                plotConstraints(cf.Number);

                plotTrajectory(trajectory, parameters.dt, 2, log);
                pause(1);
            end

            % log
            if(log)
                trajName = ['Initial_Trajectory_' date];
                logTrajectory(trajName,initialTrajectory, parameters,
exitflag, output.iterations, output.funcCount);
                trajName = ['Final_Trajectory_' date];
                logTrajectory(trajName,trajectory, parameters, exitflag,
output.iterations, output.funcCount);
            end
        else
            disp('local minimum previously found');
        end
    end
    toc
    fprintf('\n\n')
end
```

The function that generates the initial trajectory is inlined below. It uses the initial and final points to rotate and translate a randomly generated Fourier Series. The Fourier Series is set so it matches the required boundaries and penalizes terms as their frequency increases to avoid high jerk trajectories:

```
function trajectory = generateInitialTrajectory(trajectory, nOfPoints)

%% Extract information from trajectory
dx = trajectory(1,end)-trajectory(1,1);
dy = trajectory(2,end)-trajectory(2,1);

dist = sqrt(dx^2 + dy^2);
heading = atan2(dy,dx);
```

```
distChar = 1e3;
n = floor(dist/distChar); %number of modes in fourier series

%% Generate random fourier series
x = 0:1/(nOfPoints-1):1;
y = zeros(size(x));
maxY = .33;
for i=1:n
    random = randn(1)/i^2;
    y = y + maxY .* random .* sin(i*pi*x);
end


quotient = max(abs(y))/maxY;
if(quotient>1)
    y = y/quotient;
end
%% Rotate to match trajectory axis
rotMat = [cos(heading) -sin(heading); sin(heading) cos(heading)];

position = rotMat * dist * [x; y] + trajectory(1:2,1)*ones(1,nOfPoints);

trajectory = zeros(5,nOfPoints);
trajectory(1:2,:) = position;
trajectory(3,:) = heading;

end
```

Finally, the function to check for new minima is inlined:

```
function new = checkForNewMinima(fNew, reset)

persistent fOld
if(reset)
    fOld=[];
    return
end

new = 1;
for i=1:numel(fOld)
    ratio = fNew / fOld(i);

    if(ratio > .98 && ratio < 1.02)
        new = 0;
        break
    end
end

if(new)
    fOld = [fOld fNew];
end

end
```

B.3. Auxiliar Tools

This functions are used for translating the design vector to a more understandable matrix containing different physical variables in rows and the columns define the time instant. They also allow to plot, log, and load trajectories. This way the multistart script is run first and then results are loaded and analyzed.

The first scripts handle design vector and trajectory matrix translation:

```
function [x, l] = trajectoryToVector(trajectory)

[m, n] = size(trajectory);
l = m*n;
x = zeros(l, 1);

for i=1:m
    index = n*(i-1);
    x(index+1 : index+n) = trajectory(i,:);
end
checkTrajectoryVector(x);

end
```

```
function [trajectory, m, n, dt] = vectorToTrajectory(x)

checkTrajectoryVector(x);

l = numel(x);
m = 5;
n = l/5;

trajectory = zeros(m, n);
for i=1:m
    index = n*(i-1);
    trajectory(i,:) = x(index+1 : index+n);
end
end
```

```
function valid = checkTrajectoryVector(x)

valid = 1;

%% Check input validity

if(nargin ~= 1)
    error('error 1: wrong number of inputs');
end

if(size(size(x),2) ~= 2)
    error('error 2: input must be a matrix (2D array)')
end
```

```
[m, n] = size(x);
if(m ~= 1 && n ~= 1)
    error('error 3: input must be a either a row or column vector')
else
    l = m*n;
end

if(mod(l,5) ~= 0)
    error('error 4: input vector must have size multiple of 5');
end

end
```

The next scripts are used to plot the trajectory, the No Fly zones and the roll and roll rate:

```
function plotTrajectory(trajectory, dt, plots, log)

global trajName
%% Handle inputs
[m, n] = size(trajectory);

x = trajectory(1,:);
y = trajectory(2,:);
heading = trajectory(3,:);
roll = trajectory(4,:);
rollRate = trajectory(5,:);

t = 0 : dt : dt*(n-1);
%% Plot trajectory

if(plots<=1)
    figure()
    plot(y,x,'-x')
    axis equal
    cf = gcf();
    plotConstraints(cf.Number);

    if(log)
        fileName = ['logs\' trajName];
        saveas(cf.Number,fileName);
    end
end

if(plots>=1)
    figure()
    plot(t,roll,'b-x');
    hold on
    plot(t,rollRate,'r-x');
    hold off
    cf = gcf();
    plotConstraints(cf.Number);
    if(log)
        trajId = trajName(18:end);
        fileName = ['logs\state_'  trajId];
        saveas(cf.Number,fileName);
    end
end

end
```

The constraints can be plotted if the following script is called:

```
function plotConstraints(figureId)

nPoints = 300;

figure(figureId)
ax = gca();

xLimit = ax.XLim;
yLimit = ax.YLim;

deltaX = xLimit(2)-xLimit(1);
deltaY = yLimit(2)-yLimit(1);


x = xLimit(1):deltaX/(nPoints-1):xLimit(2);
y = yLimit(1):deltaY/(nPoints-1):yLimit(2);

[X,Y] = meshgrid(x,y);
Z = zeros(nPoints);

for i = 1:nPoints
    for j = 1:nPoints
        traj = [Y(i,j); X(i,j); 0; 0; 0];
        Z(i,j) = sum(max(gNoFlyConstraints(traj),0));
    end
end

Z = double(Z > 0.5);
hold on
contour(X,Y,Z);
colormap(bone(2));
hold off
end
```

These previous scripts also allow to save the MATLAB figures for later inspection. Other than the figures the outputs of the optimization: the trajectory and the optimizer diagnostics can also be saved and loaded into the workspace using the following scripts:

```
function logTrajectory(fileName, trajectory, parameters, exitflag,
iterations, funevals)

%% Evaluate trajectory
```

```
[~,n] = size(trajectory);

x = trajectory(1,:);
y = trajectory(2,:);
heading = trajectory(3,:);
roll = trajectory(4,:);
rollRate = trajectory(5,:);

xState = trajectoryToVector(trajectory);
f = fObjectiveTrajectory(xState, parameters);
h = hConstraintsTrajectory(xState, parameters);

%% Define format for logging
float4 = '%.6e\t';

formatSpecTitle = [ fileName    '\n\r'];
formatSpecHeaders=[ 'time\t'    'x\t'        'y\t'   'heading\t' 'roll\t'
'roll rate\t'   'f(x*)\t'   'hmax(x*)\t'     'exitflag'       'iterations'
'functionEvaluations'    '\n\r'];
formatSpec1 = [     '%d\t'      float4       float4  float4      float4
float4          float4      float4           '%d\t'           '%d\t'
'%d\t'                  '\n\r'];
formatSpec3 = [     '%d\t'      float4       float4  float4      float4
float4          '\t'        '\t'             '\t'             '\t'
'\t'                    '\n\r'];

%% Log
global trajName
fileName = ['logs\' trajName '.txt'];
fid=fopen(fileName,'w');
fprintf(fid, formatSpecTitle);
fprintf(fid, formatSpecHeaders);

t = 0;
fprintf(fid, formatSpec1, t, x(1), y(1), heading(1), roll(1), rollRate(1),
f(1), max(h), exitflag, iterations, funevals);
for i=2:n
    t = t + parameters.dt;
    fprintf(fid, formatSpec3, t, x(i), y(i), heading(i), roll(i),
rollRate(i));
end

fclose(fid);

end
```

```
function [trajectory, dt, f, exitflag, iterations, funevals] =
loadTrajectory(fileName)
%% Define format for logging
float4 = '%.6e\t';

% formatSpecTitle = [ fileName    '\n\r'];
% formatSpecHeaders=[ 'time\t'    'x\t'        'y\t' 'heading\t' 'roll\t'
'roll rate\t'   'f(x*)\t'   'hmax(x*)\t'    'exitflag'       'iterations'
'functionEvaluations'   '\n\r'];
% formatSpec1 = [      '%d\t'      float4      float4   float4      float4
float4          float4      float4          '%d\t'          '%d\t'
'%d\t'                  '\n\r'];
% formatSpec3 = [      '%d\t'      float4      float4   float4      float4
float4          '\t'        '\t'            '\t'            '\t'
'\t'                    '\n\r'];



%% Load data
fid=fopen(fileName,'r');

fgets(fid);
fgets(fid);

i=1;
string = fgets(fid);
cell = strsplit(string,'\t');
num = str2double(cell);
trajectory(i,:) = num(1:6);
f = num(7);
exitflag = num(9);
iterations = num(10);
funevals = num(11);
while(1)
    i = i+1;
    string = fgets(fid);
    if(string==-1)
%           display('end of file reached');
        break;
    else
        cell = strsplit(string,'\t');
        num = str2double(cell);
        trajectory(i,:) = num(1:6);
    end

end

fclose(fid);

dt = trajectory(2,1);

trajectory = trajectory(:,2:6)';

end
```

Finally, a script was developed to run all post-analysis, it loads all trajectories and find the two most optimal:

```
%% Load trajectories
%% Setup workspace

close all
clear
addpath(genpath(pwd))
clc

display = 1;
%% Load all trajectories in logs folder

logs = dir('logs');

n = numel(logs);
j = 1;
fAll = [];
for i = 1:n
    if(~isempty(regexp(logs(i).name,'txt$','ONCE')) &&
~isempty(regexp(logs(i).name,'^Final','ONCE')))
        logId = logs(i).name(18:end-4);

        disp(['Trajectory Id: ' logId]);

        initialTrajectory =
loadTrajectory(['logs\Initial_Trajectory_' logId '.txt']);
        [trajectory, dt, f, exitflag, iterations, funevals] =
loadTrajectory(['logs\Final_Trajectory_' logId '.txt']);

        disp(['objective function value: ' num2str(f)]);

        if(display)
            open(['logs\Initial_Trajectory_' logId '.fig']);
            initialFigure = gcf();
            open(['logs\Final_Trajectory_' logId '.fig']);
            finalFigure = gcf();
            open(['logs\state_' logId '.fig']);
            stateFigure = gcf();
        end

        f0 = 'fVal';                    v0 = f;
        f1 = 'initialTrajectory';       v1 = initialTrajectory;
        f2 = 'finalTrajectory';         v2 = trajectory;
        f3 = 'exitflag';                v3 = exitflag;
        f4 = 'iterations';              v4 = iterations;
        f5 = 'funevals';                v5 = funevals;

        if(display)
            f6 = 'initialFigure';           v6 = initialFigure;
            f7 = 'finalFigure';             v7 = finalFigure;
            f8 = 'stateFigure';             v8 = stateFigure;

            trajectoryInfo(j) = struct(f0,v0, f1,v1, f2,v2, f3,v3, f4,v4,
f5,v5, f6,v6, f7,v7, f8,v8);
        else
            trajectoryInfo(j) = struct(f0,v0, f1,v1, f2,v2, f3,v3, f4,v4,
f5,v5);
```

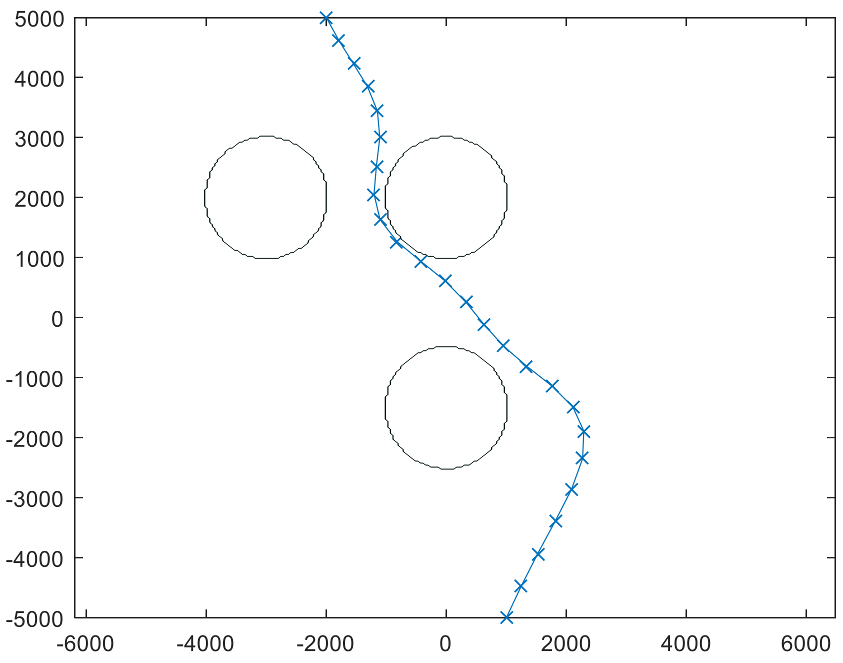
```
        end
        fAll(j) = f;

        j = j+1;
    end
end

%% Find minimum

%locate minimum
[fMin, indexMin] = min(fAll);

% locate 2nd minimum
fAll(indexMin) = inf;
[fMin2, indexMin2] = min(fAll);
```

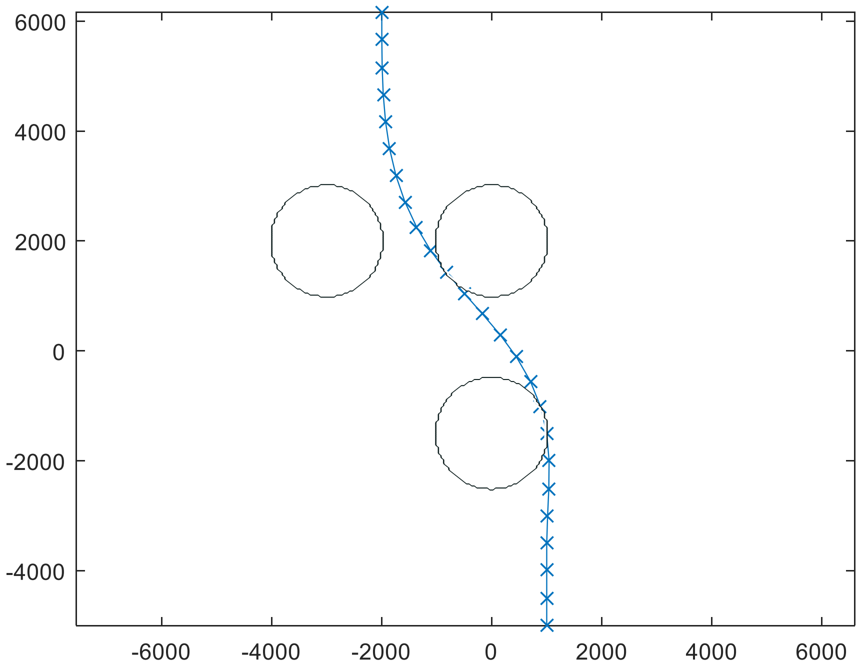
Annex C: Detailed Results

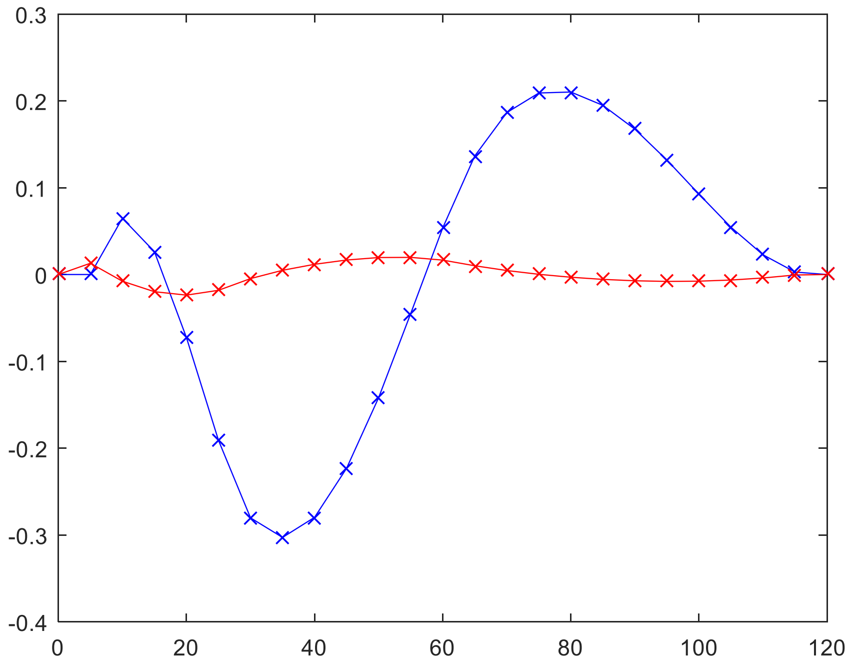
The total number of runs to the multistart algorithm was 100. From this runs, almost a third did not converge to a feasible solution, 28 found minima that were not found previously and the rest were already existing minima.

We will not describe all found minima, but we will show the convergence of different families of trajectories and the existence of local minima within a single family. We will start by showing all found families are their corresponding initial trajectories, and later we will cover the local minima in a family. All trajectories will be presented with 3 plots, the initial trajectory, the optimal achieved, and the roll angle and its rate to achieve that trajectory.

The first family is the family that contains all most optimal trajectories. This one contains the global optimal trajectory:

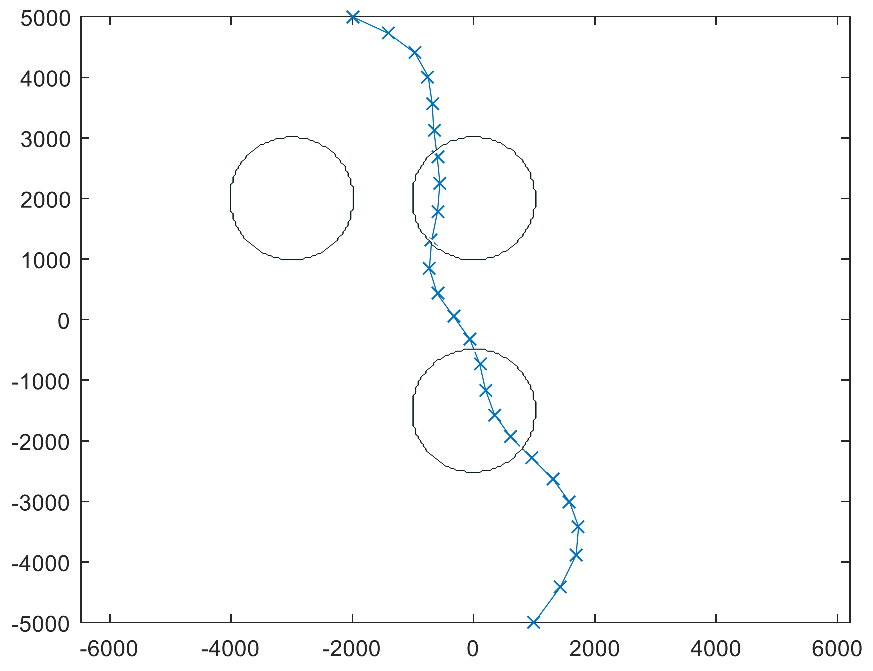


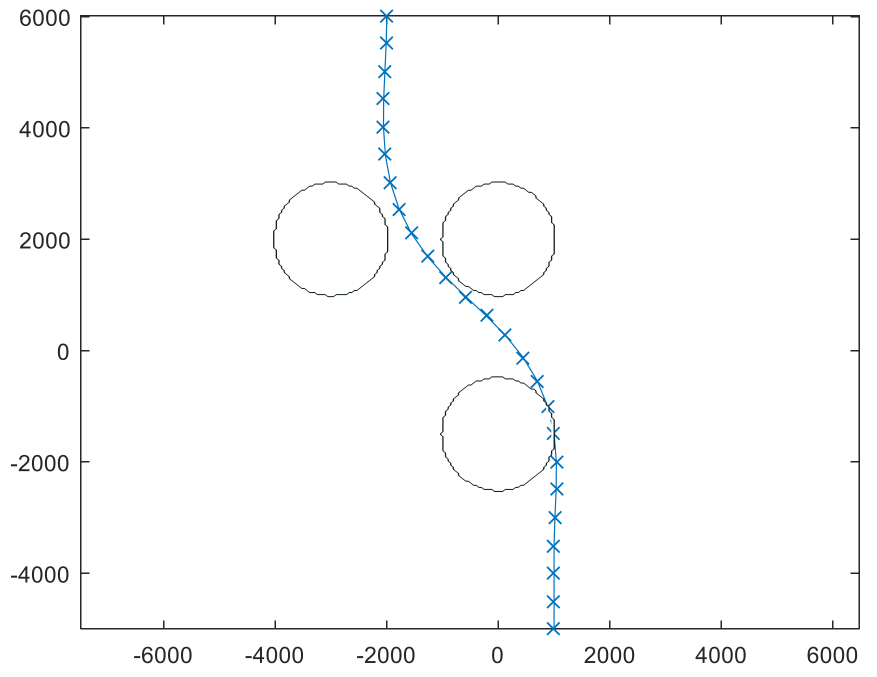


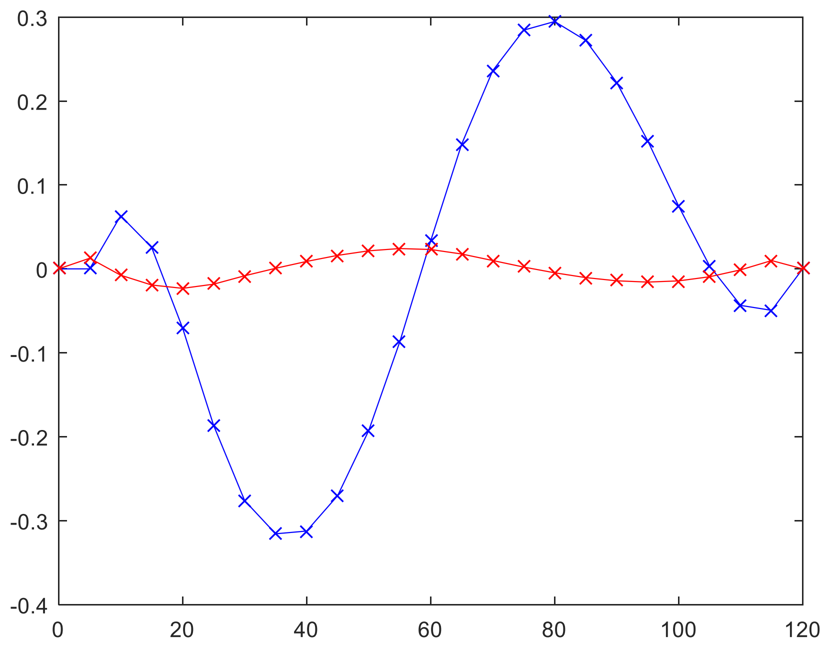


Parameter	$f(x^*)$	Exit flag	Iterations	Function evaluations
Values	0.0136	1	670	85387

This trajectory was previously shown in the results. We see that the initial trajectory already shows the patter, the optimizer performs modifications on the path and computes the inputs to achieve the optimal trajectory. For changes in the initial trajectory we may converge into other trajectories of the same family such as:

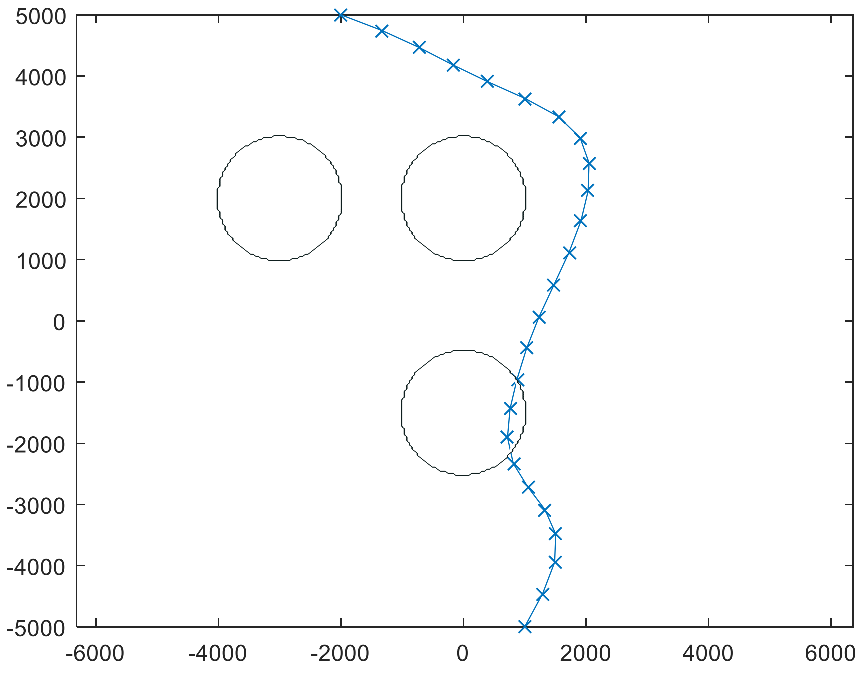


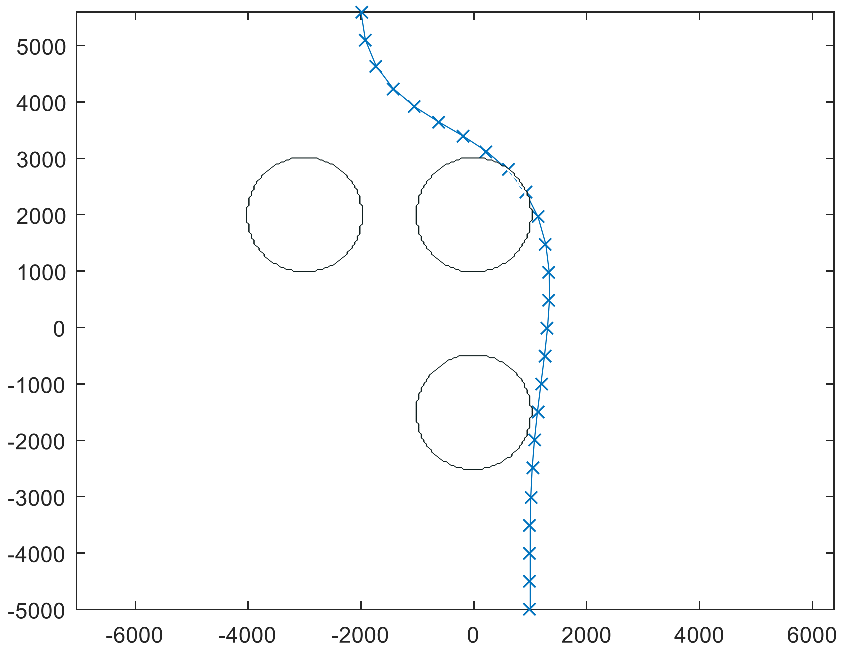


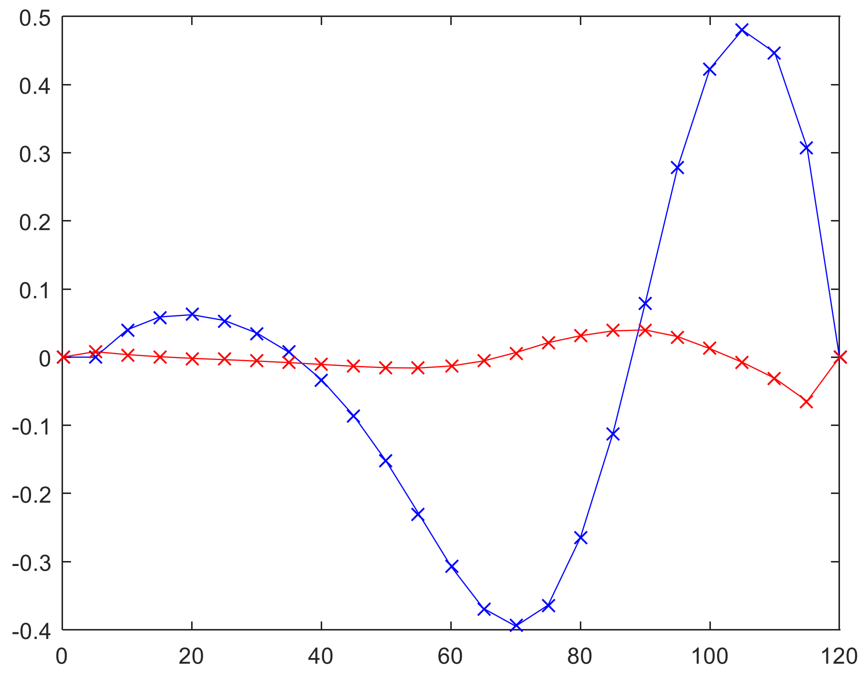


Parameter	$f(x^*)$	Exit flag	Iterations	Function evaluations
Values	0.0195	2	888	115323

Another possible family would be that contains the second solution we have shown. This one leaves all o Fly zones on its left before converging. The optimal of this family is:

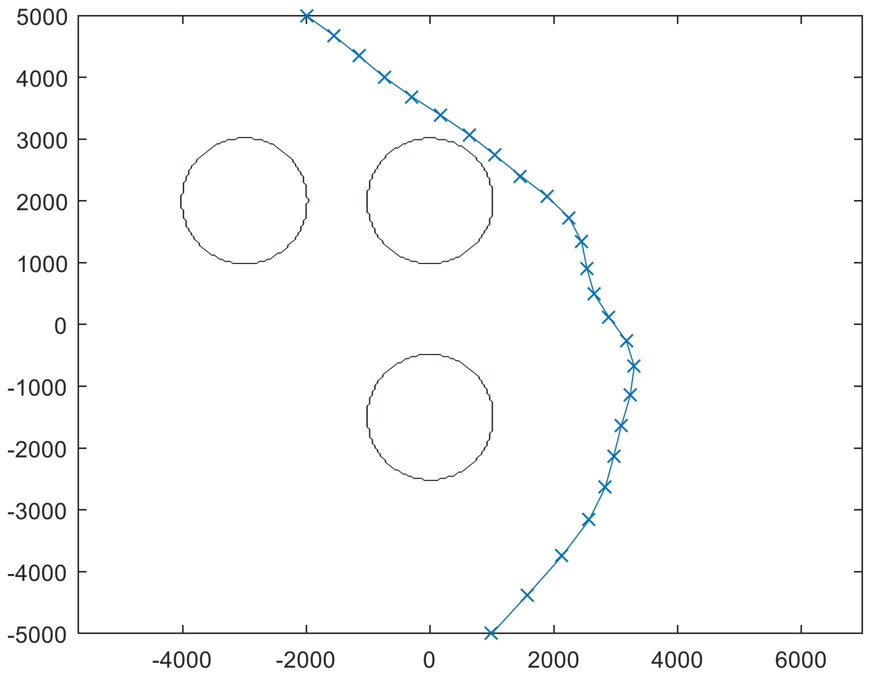

5000
4000
3000
2000
1000
0
-1000
-2000
-3000
-4000
-5000
-6000
-4000
-2000
0
2000
4000
6000

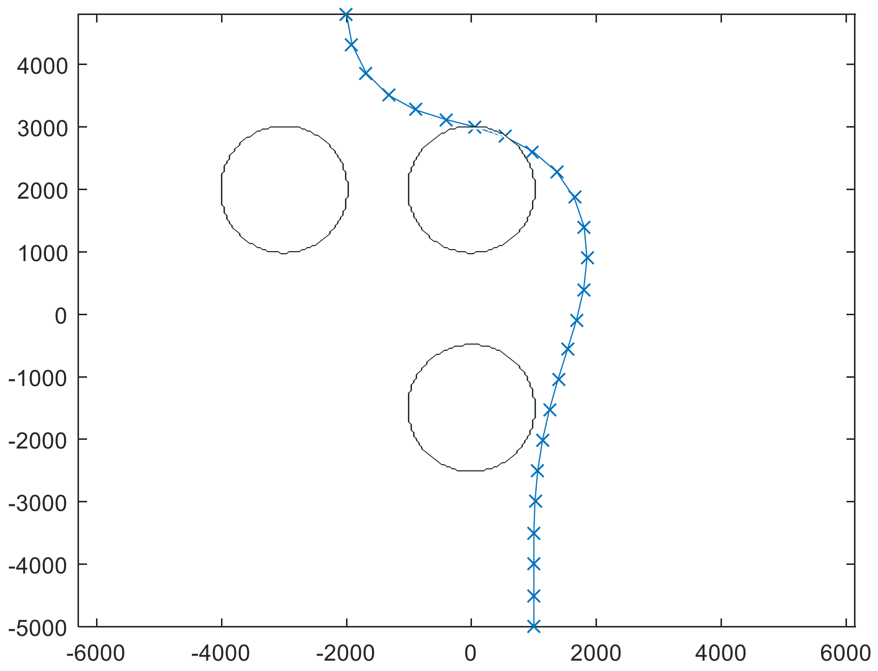

5000
4000
3000
2000
1000
0
-1000
-2000
-3000
-4000
-5000
-6000
-4000
-2000
0
2000
4000
6000

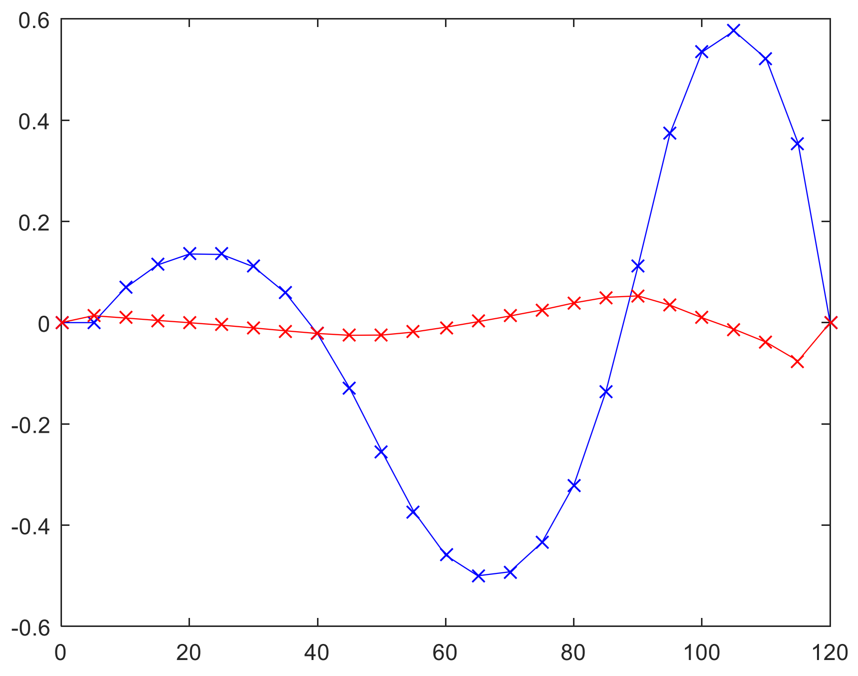


Parameter	$f(x^*)$	Exit flag	Iterations	Function evaluations
Values	0.0535	1	481	60732

We can observe that even though this trajectory is the most optimal of this family it still less optimal that the second minima of the previous family. Another example of trajectories within that curve that we have converged as a local minima are:

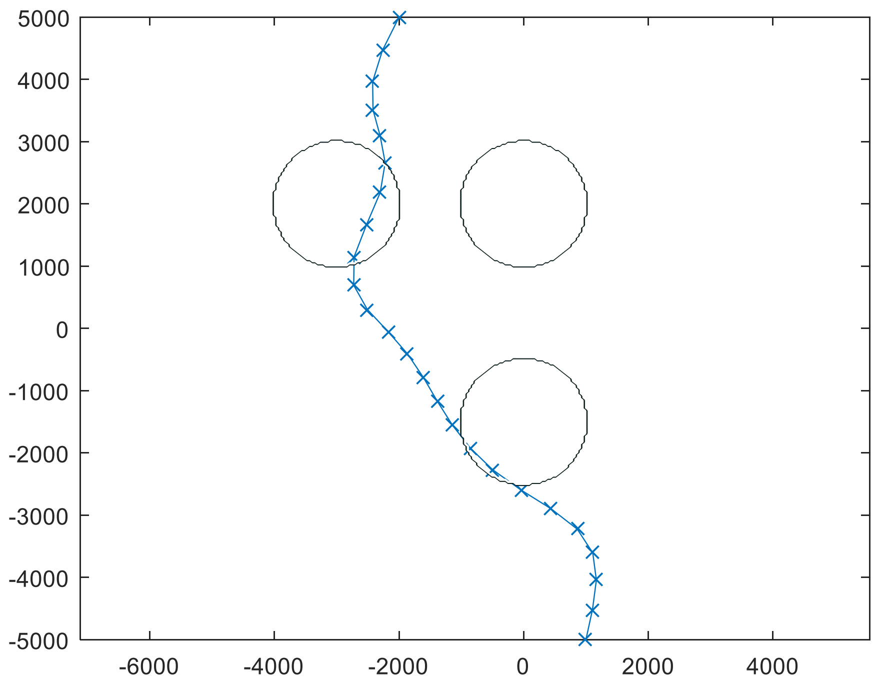

5000
4000
3000
2000
1000
0
-1000
-2000
-3000
-4000
-5000
-4000
-2000
0
2000
4000
6000

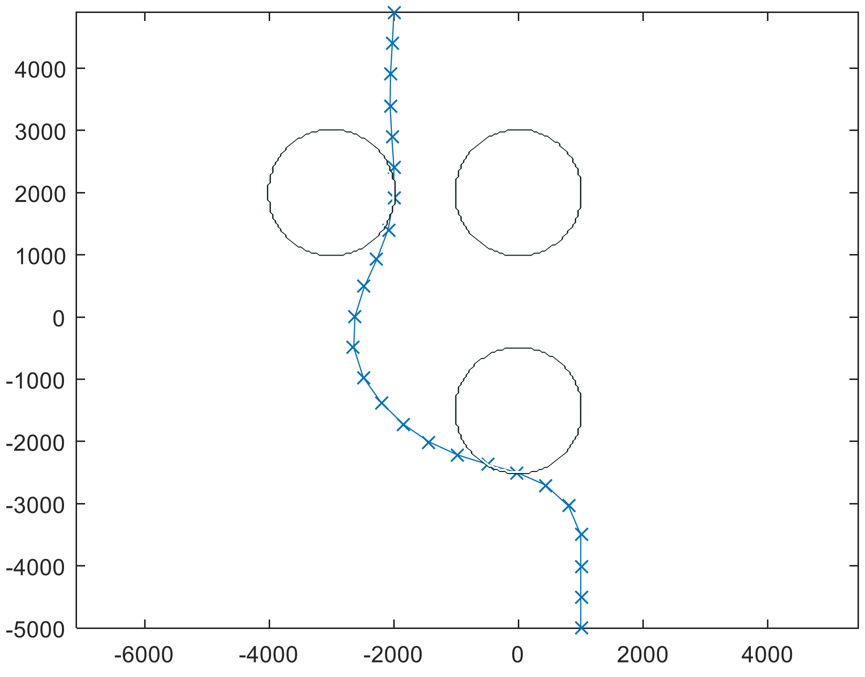

4000
3000
2000
1000
0
-1000
-2000
-3000
-4000
-5000
-6000
-4000
-2000
0
2000
4000
6000

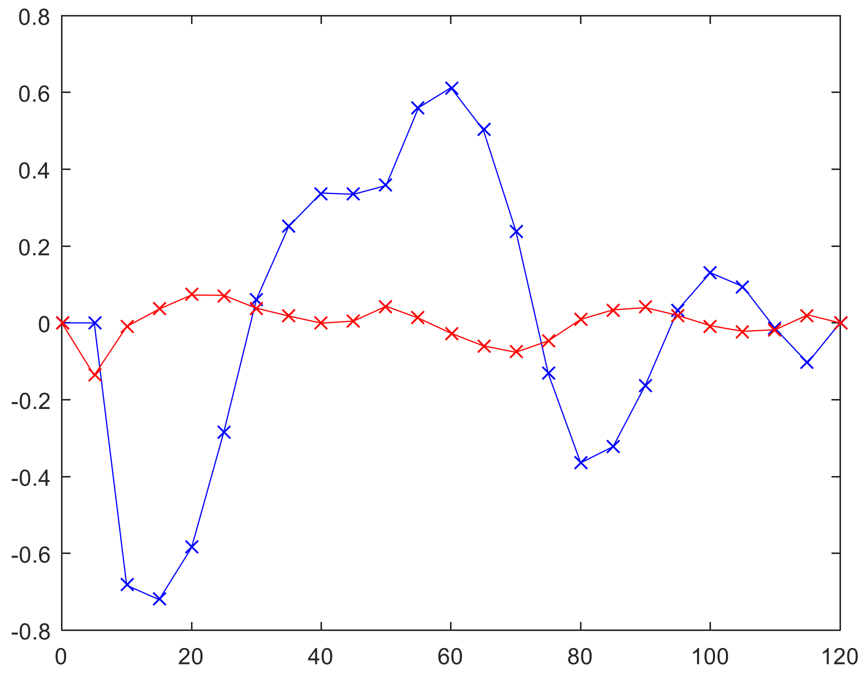


Parameter	$f(x^*)$	Exit flag	Iterations	Function evaluations
Values	0.0891	2	987	130475

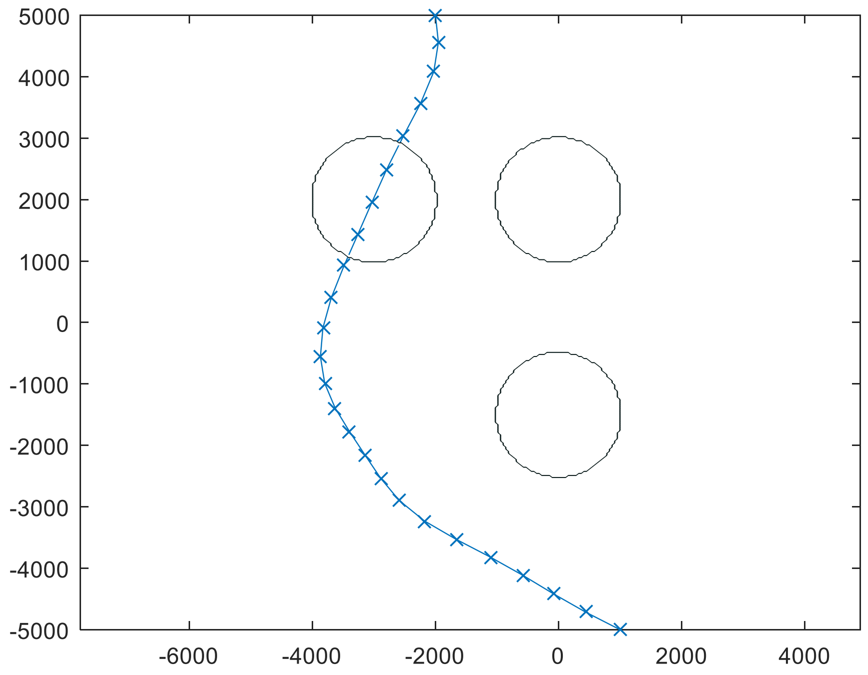
Finally, we can also observe that there are other families of trajectories such as:

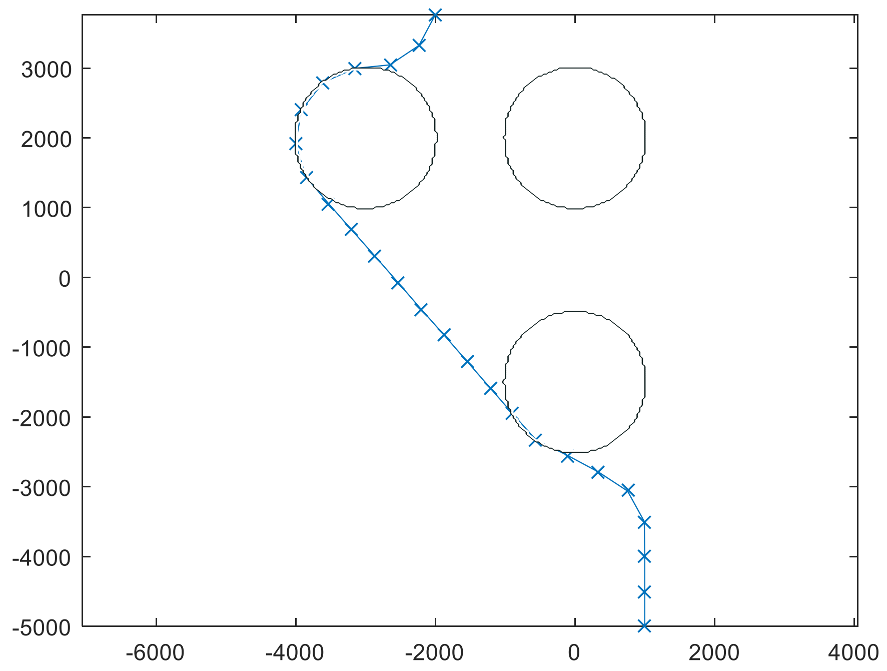


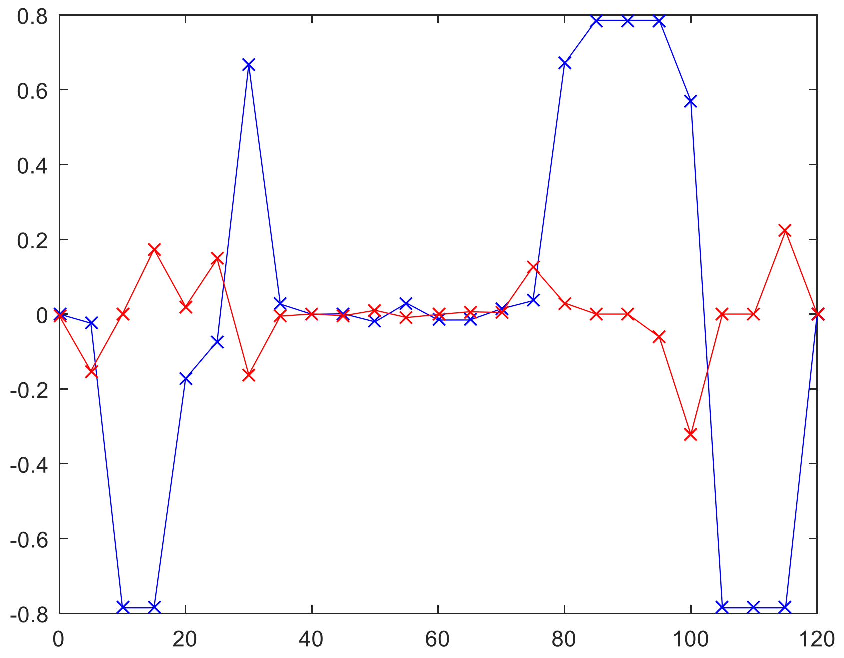




Parameter	$f(x^*)$	Exit flag	Iterations	Function evaluations
Values	0.2405	2	404	51886


5000
4000
3000
2000
1000
0
-1000
-2000
-3000
-4000
-5000
-6000
-4000
-2000
0
2000
4000


3000
2000
1000
0
-1000
-2000
-3000
-4000
-5000
-6000
-4000
-2000
0
2000
4000



Parameter	$f(x^*)$	Exit flag	Iterations	Function evaluations
Values	1.9926	2	49	6438