```matlab
%---------------------------------------------%
% BEGIN: function Q5P2_Continuous.m %
%---------------------------------------------%
function phaseout = Q5P2_Continuous(input)
% the "output" from the continuous function includes "dynamics" (states)
% "path" (path constraints), and "integrand" (trajectory constraints).
% sprintf('Entering Continuous')

global radius

%% Dynamics
% store auxiliary data locally
% R = input.auxdata.R;
R1 = input.auxdata.R1;
R2 = input.auxdata.R2;
rho = input.auxdata.rho;
I = input.auxdata.I;
m = input.auxdata.m;
c = input.auxdata.c;
k = input.auxdata.k;
f = input.auxdata.f;
r0 = input.auxdata.r0;
sigma = input.auxdata.sigma;
cos_phi1 = input.auxdata.cos_phi1;
sin_phi1 = input.auxdata.sin_phi1;
cos_phi2 = input.auxdata.cos_phi2;
sin_phi2 = input.auxdata.sin_phi2;

% store time locally
t  = input.phase.time;

% store states locally
x  = input.phase.state(:,1);
xd = input.phase.state(:,2);
y  = input.phase.state(:,3);
yd = input.phase.state(:,4);
alpha  = input.phase.state(:,5);
alphad = input.phase.state(:,6);
Ux = input.phase.state(:,7);
Uy = input.phase.state(:,8);

% store controls locally
Uxd = input.phase.control(:,1);
```

```
Uyd = input.phase.control(:,2);

% disturbance
Dx = sin(10*t);
Dy = sin(5*t);

% dynamics equations

x1  = x  - rho*alpha*cos_phi1;
x1d = xd - rho*alphad*cos_phi1;
x2  = x  + rho*alpha*cos_phi2;
x2d = xd + rho*alphad*cos_phi2;
y1  = y  - rho*alpha*sin_phi1;
y1d = yd - rho*alphad*sin_phi1;
y2  = y  + rho*alpha*sin_phi2;
y2d = yd + rho*alphad*sin_phi2;
x0  = x1 - Ux;
x0d = x1d - Uxd;
y0  = y1 + Uy;
y0d = y1d + Uxd;
% x0  = x1;
% x0d = x1d;
% y0  = y1;
% y0d = y1d;

xdot = xd;
xdd = (1/m)*(-c*x0d - k*x0 - c*x2d - k*x2 + Dx);
ydot = yd;
ydd = (1/m)*(-c*y0d - k*y0 - c*y2d - k*y2 + Dy);
adot = alphad;
add = (1/I)*(R1*(c*x0d + k*x0 - c*y2d - k*y2 + Dy)...
           + R2*(-c*x2d - k*x2 + Dx + c*y0d + k*y0));
Uxdot = Uxd;
Uydot = Uyd;

phaseout.dynamics = [xdot, xdd, ydot, ydd, adot, add, Uxdot, Uydot];

%% Path Constraint

% phaseout.path = vnorm;

%% Integral Constraint, Error Minimization
% There is no integral constraint
```

```
%% Cost functional integrand
% beta = .5;
% integrand = beta*sqrt(x.^2+y.^2) + (1-beta)*alpha.^2;
% phaseout.integrand = abs(integrand);

% put the radius of the ball encompassing the focal point walk as the only
% nonzero entry in the integrand such that the integral in the cost
% functional is only the radius. I don't know how to pass a calculated
% value otherwise
radius = calcRadius();
finalCostIntegrand = zeros(length(t),1);
finalCostIntegrand(1) = radius;
phaseout.integrand = finalCostIntegrand;

function r = calcRadius()
    % Calculate radius of focal point evolution ball centered on the nominal
    % mirror vertex position
    x2_nom = R1;
    y2_nom = R2;
    x2  = x2_nom + x + rho*alpha*cos_phi2;
    y2  = y2_nom + y + rho*alpha*sin_phi2;
    % focal point position
    xfoc_nom = x2_nom;
    yfoc_nom = y2_nom + f;
    xfoc = x2 + f*sin(alpha);
    yfoc = y2 + f*cos(alpha);
    d = zeros(length(x),1);
    for i = 1:length(x)
        d(i) = norm([xfoc_nom - xfoc(i), yfoc_nom - yfoc(i)]);
    end
    r = max(d);
end
% sprintf('Leaving Continuous')
end
%----------------------------------------------%
% END: function Q5P2_Continuous.m %
%----------------------------------------------%
```