Cube Polywell Wiffleball Modeling
Using Method of Images

Indrek Mandre <indrek@mare.ee>
Icarus @ http://talk-polywell.org/
July 29, 2008

Abstract
It has been speculated that the wiffleball effect in the cube polywell could be modeled as a superconducting sphere in the middle of the system. To do that the method of images is a perfect solution.

1 Method of Images

From http://en.wikipedia.org/wiki/Method_of_images:

Method of images (or mirror images) is a mathematical tool for solving differential equations in which the domain of the sought function is extended by the addition of its mirror image with respect to a symmetry hyperplane, with the purpose of facilitating the solution of the original problem. It is used in electrostatics to simply calculate or visualize the distribution of the electric field of a charge in the vicinity of the conducting surface. It is based on the fact that the tangential component of the electrical field to the surface of a conductor is zero, and that some field \( \mathbf{E} \) with \( \nabla \times \mathbf{E} = 0 \) and \( \nabla \cdot \mathbf{E} = 0 \) in some region is uniquely defined by its normal component over the surface which confines this region (the uniqueness theorem).

The method may also be used in magneto-statics for calculating the magnetic field of a magnet which is close to a superconducting surface. Here, the component of the magnetic field normal to the superconductor is zero. Another use is in fluid dynamics: the movement of a vortex near a wall may be calculated using an image vortex.

To apply the method of images we shall place inverted coils within the system so that the resulting field lines on the surface of the superconducting sphere would be tangential to it. This method is known to be correct for regions exterior and upon the sphere itself, integral quantities of physical properties for the domain interior to the sphere may also be related to the image system physics.
2 The Image Coils


*In the plane, the inverse of a point P with respect to a circle of center O and radius r is a point P' such that P and P' are on the same ray going from O, and OP times OP' equals the radius squared, $|OP||OP'| = r^2$."

Nomenclature:
- $a$ - radius of the spherical inversion boundary
- $r_p$ - planar radius of physical coils
- $s_p$ - spacing of physical coils
- $c_p$ - conical radius to physical coil from center of Polywell
- $r_i$ - planar radius of image coils
- $s_i$ - spacing of image coils
- $c_i$ - conical radius to image coil from center of Polywell
- $l$ - half-length of side of the Polywell cube

![Figure 1: Cross-section of the cube polywell with image coils](image)

The spherical inversion has the property that all points exterior to the sphere are reflected to points interior to the sphere by the relation

$$r_{int} = \frac{a^2}{r_{ext}}$$

where $r_{ext}$ is the radius of the point exterior to the sphere and $r_{int}$ is the radius of the point interior to the sphere. Using the spherical inversion property for the points on the physical coil circular axis we get

$$c_i = \frac{a^2}{c_p}$$

and by extension

$$\frac{c_i}{c_p} = \left(\frac{a}{c_p}\right)^2$$

2
The half-length of a side of the cubical framework that Polywell coils lie upon is given by

\[ l = r_p + \frac{s_p}{\sqrt{2}} \]

and by Pythagoras it then follows that

\[ c_p^2 = r_p^2 + l^2 = r_p^2 + \left( r_p + \frac{s_p}{\sqrt{2}} \right)^2 \]

(3)

and further, by similar triangles we can show that

\[ \frac{c_i}{c_p} = \frac{r_i}{r_p} = \frac{s_i}{s_p} \]

So taking equation 2 and equation 3 we get

\[ \frac{c_i}{c_p} = \frac{a^2}{r_p^2 + \left( r_p + \frac{s_p}{\sqrt{2}} \right)^2} = \zeta \]

where we shall call \( \zeta \) our spherical inversion factor.

Now to find the location of our image coils that will generate the spherical boundary, given a specified wiffleball radius, we use

\[ r_i = r_p \zeta \quad \text{and} \quad s_i = s_p \zeta \]

To get the correct solution the field generated by the image coils has to be corrected. We can do this through modifying the current \( I_i \) running in the image coils:

\[ I_i = -I_p \sqrt{\frac{r_p}{r_i}} = -\frac{I_p}{\sqrt{\zeta}} \]

How this equation arises is currently unclear but it gives us a perfect solution where the field lines are tangential to the sphere. By speculation this could come from the Kelvin transformation. These equations are also described in [sezginer].

Figure 2: Cube polywell with image coils
3 Field visualizations

Figure 3: Field lines at cube polywell cross-section (meridian 0)
Figure 4: Field lines at cube polywell cross-section (meridian 45)
Figure 5: Field lines on the wiffleball sphere
Figure 6: Field magnitude on the wiffleball sphere
4 Cusps

If we define cusps for the wiffleball sphere as points on the sphere where field lines are perpendicular to the sphere surface, we can identify three types of cusps:

- 6 cusps at the faces
- 8 cusps at the corners
- 12 singularities where coils are the closest

It appears there are no line cusps. Cusps are the places where the electrons can more easily leak out of the wiffleball.

References

A Octave code for calculating the fields

A.1 loop_bfield.m - off-axis magnetic field radial and axial components of a loop of current

```octave
function [Br, Ba] = loop_bfield(radius, current, r, a)
f1 = 2 * 10^(-7) * current;
R = radius;
q = r * r + R * R + a * a + 2 * r * R;
m = 4 * r * R / q;
[K, E] = ellipke(m);
sqrt_q = sqrt(q);
Br = f1 * a * (E * (q - 2 * r * R) / f2 - K) / (sqrt_q * r);
Ba = f1 * (E * (R * R - r * r - a * a) / f2 + K) / sqrt_q;
end function
```

A.2 loop_bfield3d.m - off-axis magnetic field vector components of a loop of current

```octave
function [B] = loop_bfield3d(center, normal, radius, current, pos)
len = size(pos, 1);
v = pos - repmat(center, len, 1); % vector from the center of a coil to the point
nvs = repmat(normal, len, 1); % replicate the normal vectors
a = dot(v, nvs, 2); % axial distance, axial vector is the normal vector
radial = v - nvs * repmat(a, 1, 3);
r = sqrt(dot(radial, radial, 2));
radial = radial ./ repmat(r, 1, 3);
idx = find(any(r, 2) == 0);
if (!isempty(idx))
    radial(idx, :) = zeros(size(idx, 1), 3);
end
[Br, Ba] = loop_bfield(radius, current, r, a);
B = radial * repmat(Br, 1, 3) + nvs * repmat(Ba, 1, 3);
end function
```

A.3 polywell_bfield.m - magnetic field vector components for the entire polywell system

```octave
function [B] = polywell_bfield(radius, spacing, current, pos)
rr = radius + spacing / sqrt(2);
B = loop_bfield3d([rr, 0, 0], [0, 0, 1], [0, 0, 0], radius, current, pos) + \ loop_bfield3d([rr, 0, 0], [0, 0, -1], [0, 0, 0], radius, current, pos) + \ loop_bfield3d([0, 0, 0], [rr, 0, 0], [0, 0, 1], radius, current, pos) + \ loop_bfield3d([0, 0, 0], [rr, 0, 0], [0, 0, -1], radius, current, pos);
end function
```

A.4 camera3dn.m - octave camera view angle

```octave
function [normal] = camera3dn()
[theta, phi] = view();
theta = theta + 270;
phi = phi / 2 * pi;
theta = theta + pi / 180;
normal = [cos(theta) * sin(phi), sin(theta) * sin(phi), cos(phi)];
end function
```
A.5 ball.mat - visualize the field on the modeled wiffleball sphere

% 24-sector symmetry reflections for the cube polywell
sectors = cat(3,
    [1 0 0; 0 0 1; 1 0 0], [1 0 0; 0 0 1; 0 1 0],
    [0 1 0; 0 0 1; 1 0 0], [0 1 0; 0 0 1; 0 1 0],
    [0 0 1; 1 0 0; 0 1 0], [0 0 1; 1 0 0; 0 1 0],
    [0 1 0; 0 0 1; 0 1 0], [0 1 0; 0 0 1; 0 1 0]);

% set up polywell size and the image coil size
global R S CURRENT iR iS CURRENT
R = 0.15; S = 0.04; CURRENT = 1;
A = 0.15; % wiffleball radius
inv_f = (A^2)/(R^2) = (R + S/sqrt(2))^2;
iki = Rotinv; iS = Sininv; CURRENT = -CURRENT / sqrt(inv_f);

% calculate field magnitude on the sphere
[X,Y,Z] = sphere(1000);
X = X - A; Y = Y - A; Z = Z - A;
POS = [x(:, i), y(:, i), z(:, i)];
B = polywell_brfield(R, S, CURRENT, POS);
B = polywell_brfield(R, S, CURRENT, POS);
C = reshape(sqrt(dot(B, B, 2)), size(X));

direction function within the field for our ode
function [d] = dirf(t, x)
global R S CURRENT iR iS CURRENT
pos = reshape(x, 1000, 3);
B = polywell_brfield(R, S, CURRENT, pos);
B = polywell_brfield(R, S, CURRENT, pos);
d = reshape(B, size(X));
endfunction

% varying points for lines for the ode
% assume lines go radially at equal spacing
sp = zeros(8, 3);
for i = 1:8
    phi = (i-1) * pi / 16;
    sp(:, i) = [-A * 0.01, 1e-5 - cos(phi), 1e-5 + sin(phi)];
endfor

% calculate the field lines on the sphere
options = odeset( 'MaxStep', 0.001, 'InitialStep', 0.001,
    'AbsTol', 1e-6, 'RelTol', 1e-4);
[x, y] = ode45(dirf, [0 0.22], sp(:, i), options);
POS = [x(1,:), y(1,:), z(1,:)];

% map the positions into 24 sectors
for j = 1:size(sectors, 2)/3;
    for i = 1:n
        pos = cat(2, x(:, i), y(:, i), z(:, i));
        % erase the curb lines
        idx1 = find(dot(pos, curv) < (A-0.001)^2);
        pos(idx1) = 0;
        % map the positions into 24 sectors
    endfor
endfor

hold on

draw the sphere with field magnitudes
set(axes, 'XTick', []), 'YTick', [], 'ZTick', []);
axis([-A -A -A A]);
axis square;
view([-30, 30]);
set(axes, 'Position', [0.13, 0.13, 0.79, 0.79],
    'shading interp',
    colormap('jet')(24,:));

% draw the field lines
ct = camera3d(1);\ncn = reshape(cm, size(POS)(1), 1);
% erase points behind the view
POS(find(dot(POS, cn, 2) < 0.1,:) = 0;
idx = find(any(POS, 2));
start = 0;
for i = 1:size(POS, 1)
    if any(POS(:, :))
        if start == 0
            start = i;
        end
    else
        if start ~= 0
            data = POS(start:i-1, :);
            x = data(:, 1);
            y = data(:, 2);
            z = data(:, 3);
            plot3(x, y, z, 'color', [0 0 0]);
            view(-15, 30);
        end
        start = 0;
    end
end
hold off
print ("ball.png", ";-S1200,1200");
print ("ball.eps", ";-depsc");
cf
hold on
set(gca, 'XTick',[], 'YTick',[], 'ZTick',[]);
axis([-A A -A A -A A]);
axis square;
% draw the field lines
cl = camera3dn();
cl = repmat(cl, size(POS)(1), 1);
% erase points behind the view
POS(find(dot(POS, cl, 2) < -0.005), :) = 0;
start = 0;
for i = 1:size(POS, 1)
    if any(POS(:, :))
        if start == 0
            start = i;
        end
    else
        if start ~= 0
            data = POS(start:i-1, :);
            x = data(:, 1);
            y = data(:, 2);
            z = data(:, 3);
            plot3(x, y, z, 'color', [0 0 0]);
            view(-15, 30);
        end
        start = 0;
    end
end
hold off
print ("ball2.png", ";-S1200,1200");
print ("ball3.eps", ";-deps");