

```

1  #%%
2
3  **Team 24 -- Quantum Sensor Lab -- Development of a
   Magnetic Field Steering System**
4
5  Andrew Fine - Project Manager \
6  Trevor Peterson - Logistics Manager \
7  Steven Coyan - System Engineer \
8  Nate Howard - Software Engineer \
9  Nick Elsasser - Software Engineer \
10 Aaron Li - Software Engineer \
11 Cole Sites - CAD Engineer/Finance Manager \
12 Dean Allison - Manufacturing Engineer \
13 Thomas Brewster - Test Engineer
14
15 ---
16
17 Questions:
18
19 1) How do the coil numbers, sizes, positions, and currents
   interact to create a magnetic field and how can they be
   manipulated to change the field?
20
21 2) Can we simulate the magnetic field fast enough to test
   PID control loops in real time?
22
23 Ideas:
24
25 Implement a basic magnetic field simulation with some
   simplifying assumptions to get a baseline for how the main
   coil properties interact with one another to create a
   magnetic field and also to get a sense of how long it takes
   to query for the magnetic field at a given (x, y, z). Our
   simulation is based off of the equations from a paper
   provided to us by our client. We also want to be able to
   take slices of the magnetic field to visualize how the
   field is created from the coils.
26
27 #%%
28
29 import numpy as np
30 import matplotlib.pyplot as plt
31 import seaborn as sns
32 from time import perf_counter_ns
33
34 #%%
35
36 #
37 # Simulation follows an implementation of:
38 #

```

```

39 # Equations for the Magnetic Field Produced by One
40 # or More Rectangular Loops of Wire in the Same Plane -
    Misakian
41 #
42 # https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=
    10634
43 #
44 #
45 #
46 #
47 # Define helper functions
48 #
49
50 r1 = lambda x, y, z, a1, b1, z0: np.sqrt((a1 + x) ** 2 + (y
    + b1) ** 2 + (z - z0) ** 2)
51 r2 = lambda x, y, z, a1, b1, z0: np.sqrt((a1 - x) ** 2 + (y
    + b1) ** 2 + (z - z0) ** 2)
52 r3 = lambda x, y, z, a1, b1, z0: np.sqrt((a1 - x) ** 2 + (y
    - b1) ** 2 + (z - z0) ** 2)
53 r4 = lambda x, y, z, a1, b1, z0: np.sqrt((a1 + x) ** 2 + (y
    - b1) ** 2 + (z - z0) ** 2)
54
55 d1 = lambda x, y, z, a1, b1, z0: y + b1
56 d2 = lambda x, y, z, a1, b1, z0: y + b1
57 d3 = lambda x, y, z, a1, b1, z0: y - b1
58 d4 = lambda x, y, z, a1, b1, z0: y - b1
59
60 c1 = lambda x, y, z, a1, b1, z0: a1 + x
61 c2 = lambda x, y, z, a1, b1, z0: a1 - x
62 c3 = lambda x, y, z, a1, b1, z0: -a1 + x
63 c4 = lambda x, y, z, a1, b1, z0: -a1 - x
64
65 def _eval_r(x, y, z, a1, b1, z0):
66     return (r1(x, y, z, a1, b1, z0), r2(x, y, z, a1, b1, z0
    ),
67            r3(x, y, z, a1, b1, z0), r4(x, y, z, a1, b1, z0
    ))
68
69
70 def _eval_d(x, y, z, a1, b1, z0):
71     return (d1(x, y, z, a1, b1, z0), d2(x, y, z, a1, b1, z0
    ),
72            d3(x, y, z, a1, b1, z0), d4(x, y, z, a1, b1, z0
    ))
73
74
75 def _eval_c(x, y, z, a1, b1, z0):
76     return (c1(x, y, z, a1, b1, z0), c2(x, y, z, a1, b1, z0
    ),
77            c3(x, y, z, a1, b1, z0), c4(x, y, z, a1, b1, z0

```

```

77 ))
78
79 #%%
80
81 # Magnetic permeability constant
82 MU_0 = 4 * np.pi * 1.0e-7
83
84 #
85 # Define field component equations, B = <Bx, By, Bz>
86 #
87
88 def Bx(x, y, z, a1, b1, z0, N):
89     """
90     Magnetic flux density in X direction at point (x, y, z
91     ) for a
92     coil of size 2*a1, 2*b1 residing on the x, y plane
93     defined by z0
94
95     :param x: x location
96     :param y: y location
97     :param z: z location
98     :param a1: (half of) coil side length A
99     :param b1: (half of) coil side length B
100    :param z0: Distance of coil from origin
101    :param N: Current
102
103    :return: Magnetic flux density
104    """
105    _k = 100.0 * MU_0 * N / (4 * np.pi)
106    _z = z - z0
107    _r = _eval_r(x, y, z, a1, b1, z0)
108    _d = _eval_d(x, y, z, a1, b1, z0)
109    return _k * np.sum([(-1**i) * _z / (_r[i] * (_r[i] +
110    _d[i]))
111                                for i in range(4)], axis=0)
112
113 def By(x, y, z, a1, b1, z0, N):
114     """
115     Magnetic flux density in Y direction at point (x, y, z
116     ) for a
117     coil of size 2*a1, 2*b1 residing on the x, y plane
118     defined by z0
119
120     :param x: x location
121     :param y: y location
122     :param z: z location
123     :param a1: (half of) coil side length A
124     :param b1: (half of) coil side length B

```

```

122     :param z0: Distance of coil from origin
123     :param N: Current
124
125     :return: Magnetic flux density
126     """
127
128     _k = 100.0 * MU_0 * N / (4 * np.pi)
129     _z = z - z0
130
131     _r = _eval_r(x, y, z, a1, b1, z0)
132     _c = _eval_c(x, y, z, a1, b1, z0)
133
134     return _k * np.sum([(-1**i) * _z / (_r[i] * (_r[i]
135 ] + (-1**i) * _c[i])) for i in range(4)], axis=0)
136
137 def Bz(x, y, z, a1, b1, z0, N):
138     """
139     Magnetic flux density in Z direction at point (x, y, z
140 ) for a
141     coil of size 2*a1, 2*b1 residing on the x, y plane
142     defined by z0
143
144     :param x: x location
145     :param y: y location
146     :param z: z location
147     :param a1: (half of) coil side length A
148     :param b1: (half of) coil side length B
149     :param z0: Distance of coil from origin
150     :param N: Current
151
152     :return: Magnetic flux density
153     """
154
155     _k = 100.0 * 1e6 * MU_0 * N / (4 * np.pi)
156
157     # compute each sub-function once and store result
158     _r = _eval_r(x, y, z, a1, b1, z0)
159     _d = _eval_d(x, y, z, a1, b1, z0)
160     _c = _eval_c(x, y, z, a1, b1, z0)
161
162     # return constant * inner sum of various function
163     combinations
164     # tried to keep things clean, or at least cleaner than
165     the original mathematica code
166
167     return _k * np.sum([((-1**i)*_d[i] / (_r[i] * (_r[
168 i] + (-1**i)*_c[i])) -
169 (_c[i] / (_r[i] * (_r[i] - _d[i]
170 ]))) for i in range(4)], axis=0)

```

```

165
166 ###
167
168 def get_coil_coordinates(a1, b1, s, N, x, y):
169     """
170         Get the (x, y) coordinates for the center of the NxN
171         grid of rectangles
172         so they satisfy the distances set by the rectangle
173         width, height, and spacing.
174         The coils are positioned such that their combined
175         center is at x, y
176
177         :param a1: Rectangle width (x-direction) (cm)
178         :param b1: Rectangle height (y-direction) (cm)
179         :param s: Spacing between rectangles (cm)
180         :param N: Number of rectangles
181         :param x: x position of rectangle's center
182         :param y: y position of rectangle's center
183
184         :return: np.meshgrid in ij format of each coil's
185         center for NxN grid
186     """
187
188     f = lambda x: (s * (N-1) + 2 * x * (N-1)) / 2.0
189
190     f_a1, f_b1 = f(a1), f(b1)
191     xx = np.linspace(-1.0 * f_a1, f_a1, num=N) + x
192     yy = np.linspace(-1.0 * f_b1, f_b1, num=N) + y
193
194     return np.meshgrid(xx, yy)
195
196 def get_field(x, y, z, a1, b1, z0, current, num_coils,
197 coil_spacing):
198     """
199         Compute the magnetic field at (x, y, z) created by N^2
200         coils on plane z0 arranged in an NxN grid with equal
201         spacing
202         and current
203
204         :param x: x coordinate to query (cm)
205         :param y: y coordinate to query (cm)
206         :param z: z coordinate to query (cm)
207         :param a1: Half of the rectangle width (cm)
208         :param b1: Half of the rectangle height (cm)
209         :param z0: z coordinate plane goes through
210         :param current: Current through all coils
211         :param num_coils: Number of coils per row in grid,
212         total coils is NxN
213         :param coil_spacing: Spacing between coils (cm)
214     """

```

```

207         :return: np.array containing the field x, y, z
           components in that order
208         """
209         xx, yy = get_coil_coordinates(a1, b1, coil_spacing,
           num_coils, x, y)
210
211         bx, by, bz = None, None, None
212         for i in range(num_coils):
213             for j in range(num_coils):
214                 if bx is None:
215                     bx = Bx(xx[i, j], yy[i, j], z, a1, b1, z0
           , current)
216                 else:
217                     bx += Bx(xx[i, j], yy[i, j], z, a1, b1, z0
           , current)
218
219                 if by is None:
220                     by = By(xx[i, j], yy[i, j], z, a1, b1, z0
           , current)
221                 else:
222                     by += By(xx[i, j], yy[i, j], z, a1, b1, z0
           , current)
223
224                 if bz is None:
225                     bz = Bz(xx[i, j], yy[i, j], z, a1, b1, z0
           , current)
226                 else:
227                     bz += Bz(xx[i, j], yy[i, j], z, a1, b1, z0
           , current)
228
229         return np.array([bx, by, bz])
230
231 def overlay_coils(a1, b1, num_coils, coil_spacing, unit,
           ax):
232     """
233     Overlay where the coils would be on the x, y plane in
           the magnetic field plot
234
235     :param a1: half of the coil width (cm)
236     :param b1: half of the coil height (cm)
237     :param num_coils: Number of coils per row
238     :param coil_spacing: Spacing between coils (cm)
239     :param unit: Conversion between cm and matplotlib
           pixel units
240     :param ax: matplotlib graph to apply overlay to
241     """
242
243     # convert coil sizes to matplotlib units
244     a1 *= unit
245     b1 *= unit

```

```

246     coil_spacing *= unit
247
248     f = lambda x: (coil_spacing * (num_coils - 1) + 2 * x
    * (num_coils - 1)) / 2.0
249
250     f_a1, f_b1 = f(a1), f(b1)
251
252     # shift up and over s.t. we have the top left corner
of each coil
253     xp = np.linspace(-1.0 * f_a1 + 50, f_a1 + 50, num=
    num_coils) - a1
254     yp = np.linspace(-1.0 * f_b1 + 50, f_b1 + 50, num=
    num_coils) - b1
255     xx, yy = np.meshgrid(xp, yp)
256     for i in range(num_coils):
257         for j in range(num_coils):
258             px, py = xx[i, j], yy[i, j]
259             print(px, py, 2*a1, 2*b1)
260             rect = plt.Rectangle((px, py), 2*a1, 2*b1, fc=
    'none', ec='green', lw=2.0, alpha=0.75)
261             ax.add_patch(rect)
262
263
264
265 def overlay_coilsY(a1, b1, num_coils, coil_spacing, unit,
    ax):
266     # convert coil sizes to matplotlib units
267     a1 *= unit
268     b1 *= unit
269     coil_spacing *= unit
270
271     f = lambda x: (coil_spacing * (num_coils - 1) + 2 * x
    * (num_coils - 1)) / 2.0
272
273     f_a1, f_b1 = f(a1), f(b1)
274
275     # shift up and over s.t. we have the top left corner
of each coil
276     xp = np.linspace(-1.0 * f_a1 + 50, f_a1 + 50, num=
    num_coils) - a1
277     yp = np.linspace(-1.0 * f_b1 + 50, f_b1 + 50, num=
    num_coils) - b1
278     xx, yy = np.meshgrid(xp, yp)
279     for i in range(num_coils):
280         for j in range(num_coils):
281             if i==1:
282                 continue
283             px, py = xx[i, j], yy[i, j]
284             #print(px, py, 2*a1, 2*b1)
285             rect = plt.Rectangle((px, py+(2*unit)), 2*a1, .5

```

```

285 , fc='none', ec='green', lw=2.0, alpha=0.75)
286     ax.add_patch(rect)
287
288 def overlay_coilsX(a1, b1, num_coils, coil_spacing, unit,
    ax):
289     # convert coil sizes to matplotlib units
290     a1 *= unit
291     b1 *= unit
292     coil_spacing *= unit
293
294     f = lambda x: (coil_spacing * (num_coils - 1) + 2 * x
    * (num_coils - 1)) / 2.0
295
296     f_a1, f_b1 = f(a1), f(b1)
297
298     # shift up and over s.t. we have the top left corner
of each coil
299     xp = np.linspace(-1.0 * f_a1 + 50, f_a1 + 50, num=
    num_coils) - a1
300     yp = np.linspace(-1.0 * f_b1 + 50, f_b1 + 50, num=
    num_coils) - b1
301     xx, yy = np.meshgrid(xp, yp)
302     for i in range(num_coils):
303         for j in range(num_coils):
304             if j==1:
305                 continue
306             px, py = xx[i, j], yy[i, j]
307             #print(px, py, 2*a1, 2*b1)
308             rect = plt.Rectangle((px+(2*unit), py), .5, 2*
    b1, fc='none', ec='green', lw=2.0, alpha=0.75)
309             ax.add_patch(rect)
310
311 def create_field(const, val):
312     # define some random constants to plot
313     x_axis = np.linspace(-10, 10, num=100)
314     y_axis = np.linspace(-10, 10, num=100)
315
316     # A1 and B1 are (half of) the rectangle length and
width
317     A1, B1, I, spacing = 2, 2, 1, 1.0
318
319     z_wall_1 = -100.0 # 100 cm back
320     z_wall_2 = 100.0 # 100 cm forward
321
322     GRID = 3
323     xx, yy = get_coil_coordinates(A1, B1, spacing, GRID, 0
    , 0)
324     size = max((GRID-1), 1) * A1 * spacing * 2
325
326     grid = np.zeros((100, 100))

```



```

327
328     axis1 = np.linspace(-size, size, 100)
329     axis2 = np.linspace(-size, size, 100)
330
331     unit = 100 / (2*size) # convert cm grid (2*size x 2*
size) to matplotlib unit grid (100x100 plot)
332     for i, ax1 in enumerate(axis1):
333         for j, ax2 in enumerate(axis2):
334             # for k, z in ...
335             # visualize coil placement by plotting
distance to closest coil for each (x,y)
336
337             # v = [np.sqrt((x - xx[ii, jj])**2 + (y - yy[
ii, jj])**2)
338             #     for ii in range(GRID) for jj in range(
GRID)]
339             # grid[i, j] = min(v)
340
341             # get the magnitude of the field at each x, y
342             if const == 'z':
343                 B_wall_1 = get_field(ax1, ax2, val, A1, B1
, z_wall_1, I, GRID, spacing)
344                 B_wall_2 = get_field(ax1, ax2, val, A1, B1
, z_wall_2, I, GRID, spacing)
345             elif const == 'y':
346                 B_wall_1 = get_field(ax1, val, ax2, A1, B1
, z_wall_1, I, GRID, spacing)
347                 B_wall_2 = get_field(ax1, val, ax2, A1, B1
, z_wall_2, I, GRID, spacing)
348             elif const == 'x':
349                 B_wall_1 = get_field(val, ax1, ax2, A1, B1
, z_wall_1, I, GRID, spacing)
350                 B_wall_2 = get_field(val, ax1, ax2, A1, B1
, z_wall_2, I, GRID, spacing)
351
352                 B = B_wall_1 + B_wall_2
353
354                 u, v, w = B[0], B[1], B[2]
355
356                 grid[i, j] = np.linalg.norm(B, axis=0)
357             # plot the grid
358             xticks = ['' if i % 10 else f'{x_axis[i]:.2f}' for i
in range(len(axis1))][:-1] + [axis1[-1]]
359             yticks = ['' if i % 10 else f'{y_axis[i]:.2f}' for i
in range(len(axis2))][:-1] + [axis2[-1]]
360             ax = sns.heatmap(grid, xticklabels=xticks, yticklabels
=yticks)
361
362             if const == 'z':
363                 overlay_coils(A1, B1, GRID, spacing, unit, ax)

```

```

364
365     plt.title(f'Magnetic Field Magnitude (nT) of a {
GRID}x{GRID} of Coils at Z={val}cm.')
366     plt.xlabel('X position (cm)')
367     plt.ylabel('Y position (cm)')
368
369     plt.show()
370     elif const == 'y':
371         overlay_coilsY(A1, B1, GRID, spacing, unit, ax)
372
373     plt.title(f'Magnetic Field Magnitude (nT) of a {
GRID}x{GRID} of Coils at Y={val}cm.')
374     plt.xlabel('X position (cm)')
375     plt.ylabel('Z position (cm)')
376
377     plt.show()
378     elif const == 'x':
379         overlay_coilsX(A1, B1, GRID, spacing, unit, ax)
380
381     plt.title(f'Magnetic Field Magnitude (nT) of a {
GRID}x{GRID} of Coils at X={val}cm.')
382     plt.xlabel('Y position (cm)')
383     plt.ylabel('Z position (cm)')
384
385     plt.show()
386
387 #%%
388
389 # Define parameters for our simulation that are similar to
    our actual parameters (so far)
390
391 # wall size
392 wall_size = 160.0 # cm
393
394 # number of coils
395 N = 4
396
397 spacing = 10.0 # cm
398
399 # square coils, 4x4 grid taking up 1.6m x 1.6m, 160 / 4 -
    spacing * 3 =
400 a1 = (wall_size / N) - (spacing * (N-1)) # cm
401 b1 = a1 # cm
402
403 # 1.4m of separation, let the origin be directly at the
    center
404 sep = 140.0 # cm
405 z_wall_1 = sep / 2.0 # cm
406 z_wall_2 = -z_wall_1 # cm
407

```

```

408 # for now have the current be exactly the same for all
    coils
409 I = 1.0 # Amps
410
411 # time querying field between the walls at 0, 0, 0
412 s = perf_counter_ns()
413 get_field(0, 0, 0, a1, b1, z_wall_1, I, N, spacing) +
    get_field(0, 0, 0, a1, b1, z_wall_2, I, N, spacing)
414 e = perf_counter_ns() - s
415 print(f'Time to query field: {e/1e6:.2f} milliseconds')
416
417 #%%
418
419 create_field('x',1)
420 create_field('y',1)
421 create_field('z',1)
422
423 #%% md
424
425 Insights gained from prototype:
426
427 1) With a simulation we are easily able to see how the
    variable coil parameters interact with each other to
    create the magnetic field. As our design changes we can
    change our simulation to follow and quickly see how any
    changes would impact the overall performance of magnetic
    field generation.
428
429 2) We also see that our simulation provides a good first
    step to simulating and testing our PID loops as it only
    takes a couple milliseconds to query one point in the
    field. We are also able to visualize our magnetic fields
    and coils through our simulation. Given better
    parallization of the code we may be able to speed up the
    visualizations to near real time, and we are also looking
    into full 3d visualizations of the coils and magnetic
    field as a next step for the simulation.
430

```