

TOF Rapport

Kristian Henrik Salen Sørli William B. Sørensen

November 29, 2022

Contents

1	Oppgaver	2
	Oppgave 1	2
	Oppgave 2	2
	Oppgave 3	3
	Oppgave 4	3
	Oppgave 5	4
	Oppgave 6	4
	Oppgave 7	4
2	Appendix	6
	2.1 Refrence to OpenSCAD code	6
	2.2 Recrence to Python JHU-generation code	15
	List of figures	17

Chapter 1

Oppgaver

Oppgave 1

A

I denne oppgaven ble det laget en rekke figurer ut av spagettis. Grunnen til dette var for å se hvordan effekt forskyvings kreftene hadde på de forskjellige figurene med å se hvilken av sidene knakk når figuren ble underlagt trykk.

Fra et polygonalt perspektiv er den mest uniformt integral figuren en likesidet trekant. Dette kommer av at den tåler like mye trekk og forskyvning krefter fra hver side av figuren gjennom at den fordeler krafta likt.

En likebeint trekant er sterkest på den korteste siden. Grunnen til dette er siden forskyvnings-kraften blir fordelt over de to lengere sidene.

En rettvinklet trekant er sterkest på den korteste katen.

En firkant er svakest siden den har mulighet til å oppleve plan-forskyvning.

B

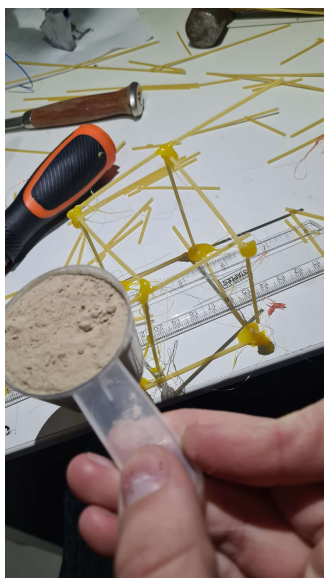
Triangulær pyramide har de samme styrkene som en likesidet trekant hvor hvor den tåler fra hver av sidene og den fordeler likt gjennom hele figuren

Rektangulær trekant har de samme styrkene som en likebeint trekant hvor den korteste kateten tåler mest siden den fordeler kraften på de lengere katetene, men det dumme med rektangulær pyramide er at den kan oppleve planforskyvning.

En kube er den svakeste formen siden den vil oppleve plan forskyvning og knekke enkelt Figure 1.1

Oppgave 2

I denne oppgaven skulle man regne ut en målestokk for gruppens modellbru. Dette gjør mann med å bruke bruen virkelige størrelse og dele det på gruppens



(a) Cube



(b) Undemonstrative cube

Figure 1.1: Cube

utvalgte målestokk

Oppgave 3

I oppgave 3 skulle man bruke målestokken man har regnet ut tidligere for å lage en arbeidstegning. Dette skulle egentlig gjøres på ark som ble gjort, men når dette ble gjort var ikke tegningen nøyaktig og den ble heller programert inn i open SCAD for å få den mest mulig nøyaktig.

Oppgave 4

Lofoten

Fisk: Lofoten er kjent rundt om i verden for sin fisker kultur. Derfor måtte vi integrere deres kultur in i vårt bru design. Så det er 3d printed flere båter og fisker som kan festes på for å symbolisere dette.

Pride: Lofoten er blant de kommunene med høyest rate av mennesker som aksepterer homofile. Dette er også blant de kommunene som var først til å vie homofile med grunnlag på at dette var den kommunen med den første homofile presten og i dag er opp mot halvparten av prestene i kommunen homofil. dette er grunnen til at vår bru har et pride flag festet på seg

Miljø: det å bygge en bru er en prosess som krever mye energi og ressurser



(a) Compression bucle of quad-pyramid



(b) Buckle force direction of tetrehedra

Figure 1.2: Tetrehedra

som er veldig skadelig for miljøet. Ifølge Architecture 2030 så står Bygg og konstruksjons industrien for 40% av årlige utslipp dette er grunnen til at vår bru er bygget med sol celle panel slik at bruen etter mange vil bli karbon nøytral og i tillegg hjelpe lokal miljøet

Oppgave 5

Oppgave 6

Oppgave 7



(a) Compression buckle isomorphism



(b) Buckle force direction of tetrehedra

Figure 1.3: Tetrehedra

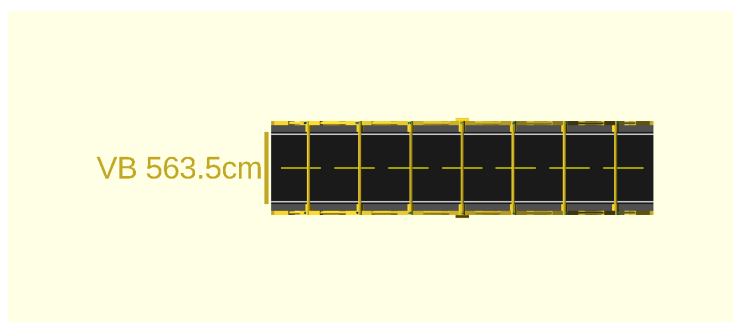


Figure 1.4: OpenSCAD dump of bride including a legend of the different measurements.

Chapter 2

Appendix

2.1 Refrence to OpenSCAD code

```
1 // Global resolution
2 $fs = $preview ? 0.4 : 0.1;
3 $fa = $preview ? 0.5 : 1;
4
5 n = 7;
6
7 // Bridge class A
8 length_irl = 28 * 100;
9
10 length = 80;
11 width = 20;
12 height = 25;
13
14 gain = 3;
15
16 S = 0.5;
17
18 STEP_PERCENT_A = .28;
19 STEP_PERCENT_B = .25;
20 BASE_RESCALE = 6;
21 BEAM_RAD = 0.7*S;
22 BAL_RAD = 1.5*S;
23
24 JOINT_RAD = 0.9*S;
25 MARGIN = 3*S;
26
27 STANDOFF_RAD = 1.38;
28 impl_stansoff_val = STANDOFF_RAD * 2;
29
30 ROAD_H = 1;
31
32 mod2irl = length_irl / length;
33
34 echo ("Print height", (MARGIN + 2*JOINT_RAD) * 10, "mm");
35 echo ("Scale", mod2irl);
```

```

36
37 function cosh (x) = (exp (x) + exp(-x)) / 2;
38 function sinh (x) = (exp (x) - exp(-x)) / 2;
39
40 function norm_gain_cosh (x) = (-cosh(x * gain) + cosh(gain))/(cosh(
    gain) - 1);
41 function norm_gain_cosh_dx (x) = -gain * sinh(gain * x) / (cosh(
    gain) - 1);
42
43 // percent and plusminus norm
44 function p2pm (x) = 2 * ((x) - 0.5);
45 function pm2p (x) = (x + 1) / 2;
46
47 function swXZ (x) = [x.x, x.z];
48
49 // Cyl From To
50 module cft(from, to, rad) {
51     let (
52         vft = to - from,
53         mag = sqrt(vft.x * vft.x + vft.y * vft.y + vft.z * vft.z),
54         rotx = atan2(vft.x, vft.z),
55         roty = atan2(vft.x, vft.y)
56     ) {
57         translate(from)
58         rotate([rotx, 0, roty])
59         cylinder(mag, rad, rad);
60     }
61 }
62
63 module c3ft(from, to, rad) {
64     let (
65         vft = to - from,
66         mag = sqrt(vft.x * vft.x + vft.y * vft.y + vft.z * vft.z),
67         rotx = atan2(vft.x, vft.z),
68         roty = atan2(vft.x, vft.y)
69     ) translate(from)
70     rotate([rotx, 0, roty])
71     translate([-rad, -rad, 0])
72     cube([rad * 2, rad * 2, mag]);
73 }
74
75 module num (n,h=1) {
76     color("teal") translate([0,0,-0.5])
77     linear_extrude(height=h, convexity=4)
78     text(str(n),
79         size=S,
80         font="Consolas",
81         halign="center",
82         valign="center");
83 }
84
85 module arch (points, height,nums=true) {
86     for (i = [0:points -1])
87         let (
88             inc = length / points,
89             center = length / 2,
90             is = i + 0.5,

```



```

91         i1 = is + 1,
92         i2 = is - 1,
93
94         x = p2pm(is / points),
95         y = norm_gain_cosh(x),
96
97         x1 = p2pm(i1 / points),
98         y1 = norm_gain_cosh(x1),
99
100        x2 = p2pm(i2 / points),
101        y2 = norm_gain_cosh(x2),
102
103        pt = [is * inc - center, 0, height * y],
104        p1 = [i1 * inc - center, 0, height * y1],
105        p2 = [i2 * inc - center, 0, height * y2],
106        ll = [i * inc - center, 0, 0],
107
108        p1ptB = (p1 - pt)*STEP_PERCENT_B,
109        p2ptB = (p2 - pt)*STEP_PERCENT_B,
110        llptA = (ll - pt)*STEP_PERCENT_A,
111
112        l_ssa1 = [
113            [-llptA.x, llptA.z],
114            swXZ(llptA),
115        ],
116        l_ssa2 = concat(
117            i != points - 1 ? [[0,0], swXZ(p1ptB)] : [[0,0]],
118            l_ssa1
119        ),
120
121        l_ssa3 = concat(
122            l_ssa2,
123            i != 0 ? [swXZ(p2ptB)] : []
124        ),
125
126        angle = atan(norm_gain_cosh_dx(x))
127    ) {
128        translate(pt + [0,MARGIN/2,0])
129        rotate([90,0,0])
130        cylinder(JOINT_RAD*2 + MARGIN, BAL_RAD, BAL_RAD,
131        true);
132
133        if(i != points-1)
134            c3ft(
135                pt,
136                p1ptB + pt,
137                JOINT_RAD
138            );
139        if(i != 0)
140            c3ft(
141                p2ptB + pt,
142                pt,
143                JOINT_RAD
144            );
145        c3ft(
146            pt,
147            llptA + pt,
148            JOINT_RAD

```

```

147         );
148         c3ft(
149             llptA + pt,
150             pt,
151             JOINT_RAD
152         );
153
154         translate(pt)
155             rotate([90,0,0])
156             linear_extrude(JOINT_RAD * 2, center=true)
157             polygon(l_ssa3);
158
159         if(nums)
160             translate(pt)
161                 rotate([0,-angle,0])
162                 translate([0, 0, BAL_RAD - 0.25])
163                 num(points + 1 + i);
164
165         if(i != points-1 && nums)
166             echo ("Num", points + 1 + i, "to", points + 2 + i,
167                 round(10 * (norm(p1 - pt) - impl_stansoff_val)) / 10);
168     }
169
170 module cross_beams (points, height) {
171     for (i = [0:points -1])
172         let (
173             inc = length / points,
174             center = length / 2,
175             is = i + 0.5,
176             x = p2pm(is / points),
177             y_val = norm_gain_cosh(x),
178             base = [is * inc - center, 0, height*y_val]
179         )
180             translate(base)
181                 rotate([90,0,0])
182                 cylinder(width + JOINT_RAD + BEAM_RAD, BEAM_RAD,
183                     BEAM_RAD, true);
184
185 module beams (points, height) {
186     for (i = [0:points - 2])
187         let (
188             inc = length / points,
189             center = length / 2,
190
191             is = i + 0.5,
192
193             x0 = p2pm(is / points),
194             y0 = norm_gain_cosh(x0),
195             x1 = p2pm((is + 1) / points),
196             y1 = norm_gain_cosh(x1)
197         ) cft(
198             [ is * inc - center, 0, height * y0],
199             [(is + 1) * inc - center, 0, height * y1],
200             BEAM_RAD
201         );

```

```

202 }
203
204 module baseline (points,nums=true) {
205     for ( i = [0 : points] )
206         let (
207             inc = length / points,
208             center = length / 2,
209
210             i1 = i + 0.5,
211             i2 = i - 0.5,
212
213             x1 = p2pm(i1 / points),
214             y1 = norm_gain_cosh(x1),
215
216             x2 = p2pm(i2 / points),
217             y2 = norm_gain_cosh(x2),
218
219             pt = [i * inc - center, 0, 0],
220             p1 = [ i1 * inc - center, 0, height * y1],
221             p2 = [ i2 * inc - center, 0, height * y2],
222
223             p1ptB = (p1 - pt)*STEP_PERCENT_A,
224             p2ptB = (p2 - pt)*STEP_PERCENT_A,
225
226             l_ssa1 = [
227                 [JOINT_RAD * BASE_RESCALE, 0],
228                 [-JOINT_RAD * BASE_RESCALE, 0],
229             ],
230             l_ssa2 = concat(
231                 i != points ? [swXZ(p1ptB)] : [],
232                 l_ssa1
233             ),
234             l_ssa3 = concat(
235                 l_ssa2,
236                 i != 0 ? [swXZ(p2ptB)] : []
237             )
238         ) {
239             translate(pt + [0,MARGIN/2,0])
240             cube([
241                 JOINT_RAD * 2 * BASE_RESCALE,
242                 JOINT_RAD * 2 + MARGIN,
243                 JOINT_RAD * 2
244             ], true);
245
246             if(i != points)
247                 c3ft(
248                     pt,
249                     p1ptB + pt,
250                     JOINT_RAD
251                 );
252             if(i != 0)
253                 c3ft(
254                     p2ptB + pt,
255                     pt,
256                     JOINT_RAD
257                 );
258

```

```

259         translate(pt)
260             rotate([90,0,0])
261             linear_extrude(JOINT_RAD * 2, center=true)
262             polygon(l_ssa3);
263
264         if(nums)
265             translate(pt + [0, 0, -JOINT_RAD + 0.3]) num(i);
266
267         if (nums && i != points)
268             echo(
269                 "Num",
270                 i,
271                 "To",
272                 i + n + 1,
273                 round(
274                     (norm(p1 - pt) - impl_stansoff_val)*10
275                 )/10
276             );
277     }
278 }
279
280 module base_beams (points) {
281     for ( i = [0 : points] )
282         let (
283             inc = length / points,
284             center = length / 2
285         )
286             translate([i * inc - center, 0, 0])
287             rotate([90,0,0])
288             cylinder(width + JOINT_RAD + BEAM_RAD, BEAM_RAD,
289                 BEAM_RAD, true);
290 }
291
292 module supports (points, height) {
293     for ( i = [0 : points -1] )
294         let (
295             inc = length / points,
296             center = length / 2,
297             is = i + 0.5,
298             x = p2pm(is / points),
299             y = norm_gain_cosh(x) * height
300         ) cft(
301             [i * inc - center, 0, 0],
302             [is * inc - center, 0,y],
303             BEAM_RAD
304         );
305 }
306
307 for(i = [0:n-1])
308     echo("Num",i,"To",i+1, round(10 * (length / n -
309         impl_stansoff_val)) / 10);
310
311 module standoff () {
312     translate([0,-2* JOINT_RAD,0])
313     rotate([90,0,0])
314     difference() {
315         cylinder(JOINT_RAD*4,STANDOFF_RAD,STANDOFF_RAD,center=true)

```

```

314     ;
315     cylinder(JOINT_RAD*4,BEAM_RAD,BEAM_RAD,center=true);
316 }
317
318 module unifiedstandoff(points) {
319     for ( i = [0 : points] )
320         let (
321             inc = length / points,
322             center = length / 2,
323             pt = [i * inc - center, 0, 0]
324         ) {
325             translate(pt)standoff();
326         }
327     for (i = [0:points - 1])
328         let (
329             inc = length / points,
330             center = length / 2,
331
332             is = i + 0.5,
333
334             x0 = p2pm(is / points),
335             y0 = norm_gain_cosh(x0),
336             x1 = p2pm((is + 1) / points),
337             y1 = norm_gain_cosh(x1)
338         ) translate([is * inc - center, 0, height * y0])standoff();
339 }
340
341 module b (nums=true) {
342     union() {
343         arch (n, height, nums);
344         baseline(n,nums);
345     }
346 }
347
348 module ball_size () {
349     difference() {
350         b();
351         beams(n, height);
352         supports(n, height);
353         rotate([0,0,180])
354             supports(n, height);
355         rotate([0, 90, 0])
356             cylinder(length, BEAM_RAD, BEAM_RAD, true);
357     }
358 }
359
360
361 module balls () {
362     if ($preview) translate([0,-10,0]) cylinder(JOINT_RAD * 2 +
363     MARGIN,1,1);
364
365     rotate([-90,0,0]) {
366         intersection () { unifiedstandoff(n); scale([1,2,1])b(false
367 ); }
368         translate([0, -width / 2 - JOINT_RAD, 0])
369         difference() {

```

```

368         translate([0, width/2,0]) rotate([180,180,0]) ball_size
369     );
370     base_beams(n);
371     cross_beams(n, height);
372 }
373 }
374 module side () {
375     arch (n, height);
376     beams(n, height);
377     baseline(n);
378     supports(n, height);
379     rotate([0,0,180])
380         supports(n, height);
381     rotate([0, 90, 0])
382         cylinder(length, BEAM_RAD, BEAM_RAD, true);
383 }
384
385 module legend () {
386     translate([
387         -(length/2 + JOINT_RAD * BASE_RESCALE +1),
388         0,
389         0
390     ]) let (w = width - MARGIN * 2 - JOINT_RAD * 2) {
391         cube([
392             1,
393             w,
394             ROAD_H
395         ], center=true);
396         translate([-1,0,0])
397             linear_extrude(1,center=true)
398             text(str("VB ",str(w*mod2irl),"cm"),halign="right",
399             valign="center",size=5);
400     }
401 }
402
403 module roadmarking_with_distance (on, off, l, w=0.3, center=true) {
404     let(
405         dst = on + off,
406         num = floor(l/dst)
407     ) translate([center ? -num * dst / 2 : on/2,0,0])
408     for (i = [0:num])
409         translate([i*dst,0,0])
410         cube([on,w,ROAD_H],center=true);
411 }
412
413 module road () {
414     color("#222")
415     translate([0,0,BEAM_RAD + ROAD_H/2])
416     cube([
417         length + 2*JOINT_RAD * BASE_RESCALE,
418         width - MARGIN * 2 - JOINT_RAD * 2,
419         ROAD_H
420     ],center=true);
421     for (i = [-1, 1]) {
422         color("#666")

```

```

423         translate([
424             0,
425             i * (width/2 - MARGIN/2 - JOINT_RAD),
426             JOINT_RAD + ROAD_H / 2
427         ])
428         cube([
429             length + 2 * JOINT_RAD * BASE_RESCALE,
430             MARGIN,
431             ROAD_H
432         ], center=true);
433     }
434
435     for ( i = [-1,1])
436     let (
437         w = width - MARGIN * 2 - JOINT_RAD * 2,
438         sideline = [
439             0,
440             i * (width/2 - MARGIN - JOINT_RAD - 0.5),
441             BEAM_RAD + ROAD_H + 0.01
442         ],
443         major = 9,
444         minor = 3
445     ) if (w * mod2irl > 550) {
446         color("#FFF")
447         translate(sideline)
448         cube([
449             length + 2 * JOINT_RAD * BASE_RESCALE,
450             0.3,
451             0.01
452         ], center=true);
453         color("yellow")
454         translate([0,0,BEAM_RAD + ROAD_H/2 + 0.01])
455         roadmarking_with_distance(major,minor,length);
456     } else
457         color("#FFF")
458         translate(sideline)
459         roadmarking_with_distance(major,minor,length);
460 }
461
462 module pipe () {
463     difference() {
464         cylinder(3,1.2,1.2,center=true);
465         cylinder(10,BEAM_RAD, BEAM_RAD,center=true);
466     }
467 }
468
469 module main() {
470     translate([0, width/2,0]) rotate([0,0,180]) { side(); }
471     translate([0,-width/2,0]) { side(); }
472     cross_beams(n, height);
473
474     road();
475     legend();
476
477     base_beams(n);
478
479     for (i = [-1, 1]) {

```

```

480         rotate([0,90,0])
481         translate([0,i * width/2,0]) pipe();
482     }
483 }
484 main();
485 /* difference () { */
486 /*     balls(); */
487
488 /*     translate([0,0,2.5])cube([100,100,4], center=true); */
489 /* } */

```

2.2 Recrence to Python JHU-generation code

```

1  from math import exp, sqrt
2  import json
3
4  n = 7;
5
6  # Bridge class A
7  length_irl = 28 * 100;
8
9  length = 80;
10 width = 20;
11 height = 25;
12
13 gain = 3;
14
15 STEP_PERCENT_A = .30;
16 STEP_PERCENT_B = .30;
17 BASE_RESCALE = 2;
18 BEAM_RAD = 0.7;
19 BAL_RAD = 1.5;
20
21 JOINT_RAD = 0.9;
22 MARGIN = 3;
23
24 ROAD_H = 1;
25
26 mod2irl = length_irl / length;
27
28 cosh = lambda x: (exp (x) + exp(-x)) / 2
29 norm_gain_cosh = lambda x: (-cosh(x * gain) + cosh(gain))/(cosh(
    gain) - 1)
30
31 p2pm = lambda x: 2 * ((x) - 0.5)
32 pm2p = lambda x: (x + 1) / 2
33
34 dst = lambda a, b: sqrt((a[0] - b[0]) ** 2 + (a[1] - b[1]) ** 2)
35 inc = length / n
36
37 # Calculate forces
38
39 l_m_irl = length_irl/100
40 w_m_irl = (mod2irl * width)/100
41
42 print(l_m_irl, "m", w_m_irl, "m")

```



```

43
44 len_car = 4.5
45 mass_car = 3.5e3
46 lain = 2
47 g = 9.81
48
49 total_force_irl = (2 * mass_car * l_m_irl / len_car) * g
50 print(total_force_irl,"N")
51
52 pascal = total_force_irl / (l_m_irl * w_m_irl)
53 print(pascal,"Pa")
54
55 total_force_model = pascal * (width / 100) * (length / 100)
56 print(total_force_model,"N",total_force_model/g,"kg")
57
58 distribution = total_force_model / (n - 1)
59 print(distribution, "N")
60
61 # Build Bridge
62 nodes = [ ]
63
64 for i in range(n):
65     iz = i + .5
66     x = p2pm(iz / n)
67     y = norm_gain_cosh(x)
68
69     nodes.append((round(i * inc, 2), 0))
70     nodes.append((round(iz * inc, 2), round(y * height, 2)))
71
72 nodes.append((length,0))
73
74 members = []
75 member_dst = {}
76
77 for i in range(0,2*n-1,2):
78     a_dst = f'{i},{i+2}'
79     member_dst[a_dst] = dst(nodes[i],nodes[i+2])
80     members.append(a_dst)
81     if i+3 < n * 2:
82         b_dst = f'{i+1},{i+3}'
83         member_dst[b_dst] = dst(nodes[i+1],nodes[i+3])
84         members.append(b_dst)
85
86 for i in range(n*2):
87     a_dst = f'{i},{i+1}'
88     members.append(a_dst)
89     member_dst[a_dst] = dst(nodes[i],nodes[i+1])
90
91 with open('out.json','w') as f:
92     f.write(json.dumps({
93         "nodes": list(map(lambda x: f'{x[0]},{x[1]}',nodes)),
94         "members": members,
95         "supports": { "0": "P", f"{n*2}": "Rh" },
96         "forces": list(map(lambda x: f'{x*2},0,{-distribution}',
97             range(1,n))),
98         "workspace": {
99             "workspace-width": 187,

```

```

99         "workspace-height": 102,
100         "workspace-width-pixels": 1412,
101         "Yaxis-dist-from-left": 88.9971671388102,
102         "Xaxis-dist-from-bottom": 30.433852691218124,
103         "grid-x": 1,
104         "grid-y": 1,
105         "force-scale": 100
106     }
107 })
108
109 print(nodes[1], nodes[3])
110
111 for k,v in member_dst.items():
112     print(f'{k:<6} {round(v,1)}')
```

List of Figures

1.1	Cube	3
1.2	Tetrehedra	4
1.3	Tetrehedra	5
1.4	OpenSCAD dump of bride including a legend of the different measurements.	5