TOF Rapport

Kristian Henrik Salen Sørli William B. Sørensen

November 29, 2022

Contents

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

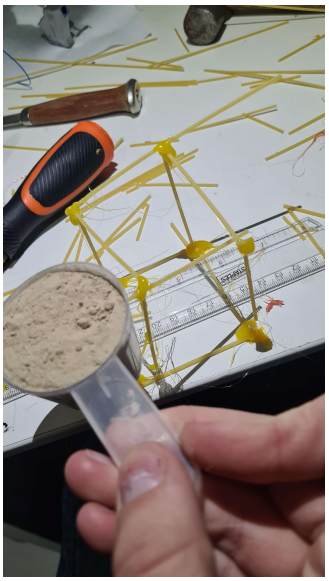
Triangulær pyramide har de samme styrkene som en likesidet trekant hvor hvor den tåler fra hver av sidene og den fordeler likt gjennom hele figuren

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 planforskyving.

En kube er den svakeste formen siden den vil oppleve plan forskyving 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) Undemonstartive 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 programer 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 intigrere 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 bucle isomorphism

(b) Buckle force direction of tetrehedra

Figure 1.3: Tetrehedra

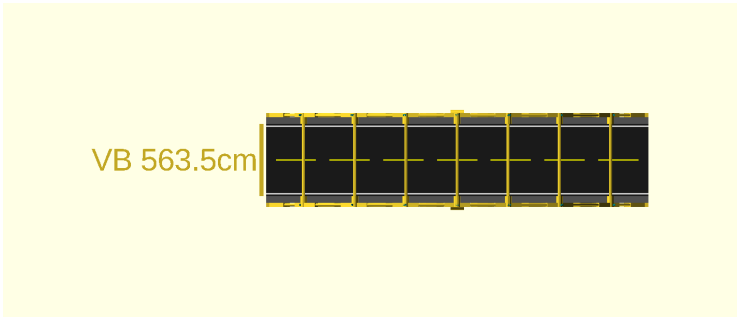


Figure 1.4: OpenSCAD dump of bride including a legend of the different measurements.

Chapter 2

Appendix

2.1 Refrence to OpenSCAD code

```
// Global resolution
$fs = $preview ? 0.4 : 0.1;
$fa = $preview ? 0.5 :   1;

n = 7;

// Bridge class A
length_irl = 28 * 100;

length = 80;
width = 20;
height = 25;

gain = 3;

S = 0.5;

STEP_PERCENT_A = .28;
STEP_PERCENT_B = .25;
BASE_RESCALE = 6;
BEAM_RAD = 0.7*S;
BAL_RAD = 1.5*S;

JOINT_RAD = 0.9*S;
MARGIN = 3*S;

STANDOFF_RAD = 1.38;
impl_stansoff_val = STANDOFF_RAD * 2;

ROAD_H = 1;

mod2irl = length_irl / length;

echo ("Print height", (MARGIN + 2*JOINT_RAD) * 10, "mm");
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,
```

```
            i1 = is + 1,
            i2 = is - 1,

            x = p2pm(is / points),
            y = norm_gain_cosh(x),

            x1 = p2pm(i1 / points),
            y1 = norm_gain_cosh(x1),

            x2 = p2pm(i2 / points),
            y2 = norm_gain_cosh(x2),

            pt = [is * inc - center, 0, height * y],
            p1 = [i1 * inc - center, 0, height * y1],
            p2 = [i2 * inc - center, 0, height * y2],
            ll = [i  * inc - center, 0, 0],

            p1ptB = (p1 - pt)*STEP_PERCENT_B,
            p2ptB = (p2 - pt)*STEP_PERCENT_B,
            llptA = (ll - pt)*STEP_PERCENT_A,

            l_ssa1 = [
                [-llptA.x, llptA.z],
                swXZ(llptA),
            ],
            l_ssa2 = concat(
                i != points - 1 ? [[0,0], swXZ(p1ptB)] : [[0,0]],
                l_ssa1
            ),

            l_ssa3 = concat(
                l_ssa2,
                i != 0 ? [swXZ(p2ptB)] : []
            ),

            angle = atan(norm_gain_cosh_dx(x))
        ) {
            translate(pt + [0,MARGIN/2,0])
                rotate([90,0,0])
                cylinder(JOINT_RAD*2 + MARGIN, BAL_RAD, BAL_RAD,
    true);
            if(i != points-1)
                c3ft(
                    pt,
                    p1ptB + pt,
                    JOINT_RAD
                );
            if(i != 0)
                c3ft(
                    p2ptB + pt,
                    pt,
                    JOINT_RAD
                );
            c3ft(
                pt,
                llptA + pt,
                JOINT_RAD
```

```
            );
            c3ft(
                llptA + pt,
                pt,
                JOINT_RAD
            );

            translate(pt)
                rotate([90,0,0])
                linear_extrude(JOINT_RAD * 2, center=true)
                polygon(l_ssa3);

            if(nums)
                translate(pt)
                    rotate([0,-angle,0])
                    translate([0, 0, BAL_RAD - 0.25])
                    num(points + 1 + i);

            if(i != points-1 && nums)
                echo ("Num", points + 1 + i, "to", points + 2 + i,
    round(10 * (norm(p1 - pt) - impl_stansoff_val)) / 10);
        }
}

module cross_beams (points, height) {
    for (i = [0:points -1])
        let (
            inc = length / points,
            center = length / 2,
            is = i + 0.5,
            x = p2pm(is / points),
            y_val = norm_gain_cosh(x),
            base = [is * inc - center, 0, height*y_val]
        )
            translate(base)
            rotate([90,0,0])
            cylinder(width + JOINT_RAD + BEAM_RAD, BEAM_RAD,
    BEAM_RAD, true);
}

module beams (points, height) {
    for (i = [0:points - 2])
        let (
            inc = length / points,
            center = length / 2,

            is = i + 0.5,

            x0 = p2pm(is / points),
            y0 = norm_gain_cosh(x0),
            x1 = p2pm((is + 1) / points),
            y1 = norm_gain_cosh(x1)
        ) cft(
            [   is    * inc - center, 0, height * y0],
            [(is + 1) * inc - center, 0, height * y1],
            BEAM_RAD
        );
```

```
}

module baseline (points,nums=true) {
    for ( i = [0 : points] )
        let (
            inc = length / points,
            center = length / 2,

            i1 = i + 0.5,
            i2 = i - 0.5,

            x1 = p2pm(i1 / points),
            y1 = norm_gain_cosh(x1),

            x2 = p2pm(i2 / points),
            y2 = norm_gain_cosh(x2),

            pt = [i * inc - center, 0, 0],
            p1 = [ i1   * inc - center, 0, height * y1],
            p2 = [ i2   * inc - center, 0, height * y2],

            p1ptB = (p1 - pt)*STEP_PERCENT_A,
            p2ptB = (p2 - pt)*STEP_PERCENT_A,

            l_ssa1 = [
                [JOINT_RAD * BASE_RESCALE, 0],
                [-JOINT_RAD * BASE_RESCALE, 0],
            ],
            l_ssa2 = concat(
                i != points ? [swXZ(p1ptB)] : [],
                l_ssa1
            ),
            l_ssa3 = concat(
                l_ssa2,
                i != 0 ? [swXZ(p2ptB)] : []
            )
        ) {
            translate(pt + [0,MARGIN/2,0])
                cube([
                    JOINT_RAD * 2 * BASE_RESCALE,
                    JOINT_RAD * 2 + MARGIN,
                    JOINT_RAD * 2
                ], true);

            if(i != points)
                c3ft(
                    pt,
                    p1ptB + pt,
                    JOINT_RAD
                );
            if(i != 0)
                c3ft(
                    p2ptB + pt,
                    pt,
                    JOINT_RAD
                );

```

```
            translate(pt)
                rotate([90,0,0])
                linear_extrude(JOINT_RAD * 2, center=true)
                polygon(l_ssa3);

            if(nums)
                translate(pt + [0, 0, -JOINT_RAD + 0.3]) num(i);

            if (nums && i != points)
                echo(
                    "Num",
                    i,
                    "To",
                    i + n + 1,
                    round(
                        (norm(p1 - pt) - impl_stansoff_val)*10
                    )/10
                );
        }
}

module base_beams (points) {
    for ( i = [0 : points] )
        let (
            inc = length / points,
            center = length / 2
        )
            translate([i * inc - center, 0, 0])
            rotate([90,0,0])
            cylinder(width + JOINT_RAD + BEAM_RAD, BEAM_RAD,
    BEAM_RAD, true);
}

module supports (points, height) {
    for ( i = [0 : points -1] )
        let (
            inc = length / points,
            center = length / 2,
            is = i + 0.5,
            x = p2pm(is / points),
            y = norm_gain_cosh(x) * height
        ) cft(
            [i  * inc - center, 0, 0],
            [is * inc - center, 0,y],
            BEAM_RAD
        );
}

for(i = [0:n-1])
    echo("Num",i,"To",i+1, round(10 * (length / n -
    impl_stansoff_val)) / 10);

module standoff () {
    translate([0,-2* JOINT_RAD,0])
    rotate([90,0,0])
    difference() {
        cylinder(JOINT_RAD*4,STANDOFF_RAD,STANDOFF_RAD,center=true)
```

```
    ;
        cylinder(JOINT_RAD*4,BEAM_RAD,BEAM_RAD,center=true);
    }
}

module unifiedstandoff(points) {
    for ( i = [0 : points] )
        let (
            inc = length / points,
            center = length / 2,
            pt = [i * inc - center, 0, 0]
        ) {
            translate(pt)standoff();
        }
    for (i = [0:points - 1])
        let (
            inc = length / points,
            center = length / 2,

            is = i + 0.5,

            x0 = p2pm(is / points),
            y0 = norm_gain_cosh(x0),
            x1 = p2pm((is + 1) / points),
            y1 = norm_gain_cosh(x1)
        ) translate([is * inc - center, 0, height * y0])standoff();
}

module b (nums=true) {
    union() {
        arch (n, height, nums);
        baseline(n,nums);
    }
}

module ball_size () {
    difference() {
        b();
        beams(n, height);
        supports(n, height);
        rotate([0,0,180])
            supports(n, height);
        rotate([0, 90, 0])
            cylinder(length, BEAM_RAD, BEAM_RAD, true);
    }
}


module balls () {
    if ($preview) translate([0,-10,0]) cylinder(JOINT_RAD * 2 +
    MARGIN,1,1);

    rotate([-90,0,0]) {
        intersection () { unifiedstandoff(n); scale([1,2,1])b(false
    ); }
        translate([0, -width / 2 - JOINT_RAD, 0])
        difference() {
```

```
            translate([0, width/2,0]) rotate([180,180,0]) ball_size
    ();
            base_beams(n);
            cross_beams(n, height);
        }
    }
}
module side () {
    arch (n, height);
    beams(n, height);
    baseline(n);
    supports(n, height);
    rotate([0,0,180])
        supports(n, height);
    rotate([0, 90, 0])
        cylinder(length, BEAM_RAD, BEAM_RAD, true);
}

module legend () {
    translate([
        -(length/2 + JOINT_RAD * BASE_RESCALE + 1),
        0,
        0
    ]) let (w = width - MARGIN * 2 - JOINT_RAD * 2) {
        cube([
            1,
            w,
            ROAD_H
        ], center=true);
        translate([-1,0,0])
            linear_extrude(1,center=true)
            text(str("VB ",str(w*mod2irl),"cm"),halign="right",
    valign="center",size=5);
    }
}

module roadmarking_with_distance (on, off, l, w=0.3, center=true) {
    let(
        dst = on + off,
        num = floor(l/dst)
    ) translate([center ? -num * dst / 2 : on/2,0,0])
        for (i = [0:num])
            translate([i*dst,0,0])
                cube([on,w,ROAD_H],center=true);
}

module road () {
    color("#222")
        translate([0,0,BEAM_RAD + ROAD_H/2])
        cube([
            length + 2*JOINT_RAD * BASE_RESCALE,
            width - MARGIN * 2 - JOINT_RAD * 2,
            ROAD_H
        ],center=true);

    for (i = [-1, 1]) {
        color("#666")
```

```
            translate([
                    0,
                    i * (width/2 - MARGIN/2 - JOINT_RAD),
                    JOINT_RAD + ROAD_H / 2
            ])
            cube([
                    length + 2 * JOINT_RAD * BASE_RESCALE,
                    MARGIN,
                    ROAD_H
            ], center=true);
    }

    for ( i = [-1,1])
    let (
        w = width - MARGIN * 2 - JOINT_RAD * 2,
        sideline = [
            0,
            i * (width/2 - MARGIN - JOINT_RAD - 0.5),
            BEAM_RAD + ROAD_H + 0.01
        ],
        major = 9,
        minor = 3
    ) if (w * mod2irl > 550) {
        color("#FFF")
            translate(sideline)
            cube([
                length + 2 * JOINT_RAD * BASE_RESCALE,
                0.3,
                0.01
            ], center=true);
        color("yellow")
            translate([0,0,BEAM_RAD + ROAD_H/2 + 0.01])
            roadmarking_with_distance(major,minor,length);
    } else
        color("#FFF")
            translate(sideline)
            roadmarking_with_distance(major,minor,length);
}

module pipe () {
    difference() {
        cylinder(3,1.2,1.2,center=true);
        cylinder(10,BEAM_RAD, BEAM_RAD,center=true);
    }
}

module main() {
    translate([0, width/2,0]) rotate([0,0,180]) { side(); }
    translate([0,-width/2,0]) { side(); }
    cross_beams(n, height);

    road();
    legend();

    base_beams(n);

    for (i = [-1, 1]) {
```

```
        rotate([0,90,0])
        translate([0,i * width/2,0]) pipe();
    }
}
main();
/* difference () { */
/*    balls(); */

/*    translate([0,0,2.5])cube([100,100,4], center=true); */
/* } */
```

2.2 Recrence to Python JHU-generation code

```
from math import exp, sqrt
import json

n = 7;

# Bridge class A
length_irl = 28 * 100;

length = 80;
width = 20;
height = 25;

gain = 3;

STEP_PERCENT_A = .30;
STEP_PERCENT_B = .30;
BASE_RESCALE = 2;
BEAM_RAD = 0.7;
BAL_RAD = 1.5;

JOINT_RAD = 0.9;
MARGIN = 3;

ROAD_H = 1;

mod2irl = length_irl / length;

cosh = lambda x: (exp (x) + exp(-x)) / 2
norm_gain_cosh = lambda x: (-cosh(x * gain) + cosh(gain))/(cosh(
    gain) - 1)

p2pm = lambda x: 2 * ((x) - 0.5)
pm2p = lambda x: (x + 1) / 2

dst = lambda a, b: sqrt((a[0] - b[0]) ** 2 + (a[1] - b[1]) ** 2)
inc = length / n

# Calculate forces

l_m_irl = length_irl/100
w_m_irl = (mod2irl * width)/100

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}',
    range(1,n))),
97         "workspace": {
98           "workspace-width": 187,
```

```
            "workspace-height": 102,
            "workspace-width-pixels": 1412,
            "Yaxis-dist-from-left": 88.9971671388102,
            "Xaxis-dist-from-bottom": 30.433852691218124,
            "grid-x": 1,
            "grid-y": 1,
            "force-scale": 100
        }
    }))

print(nodes[1],nodes[3])

for k,v in member_dst.items():
    print(f'{k:<6} {round(v,1)}')
```

List of Figures