Graz University of Technology
Institute of Applied Mechanics

Masterprojekt im Studiengang Bauingenieurwissenschaften
am Institut für Baumechanik

Technische Universität Graz

Fachwerkgenerator

Erstellen eines Fachwerks durch zufällige Punkte

Ralph Stöckl

Graz, 15.11.2018

Betreuer: Dipl.-Ing. Michael Leitner, BSc.

CONTENTS

1 Einleitung **1**

2 Programmbeschreibung **3**

3 Ergebnis Fachwerk **19**

References **23**

1 EINLEITUNG

Dieses Masterprojekt hat das Ziel mit Hilfe von Python ein Programm zu erschaffen, welches automatisch Fachwerke generiert. Um eine zügige Entwicklung zu gewährleisten, werden die Methoden des Systems Engineering (SE) angewandt. Diese beinhalten das Denken in Varianten, die Vorgehensweise vom Groben ins Detail, das Arbeiten in Projektphasen und der Anwendung des Problemlösezykluses. Zu Beginn dieses Projektes wurden zunächst mögliche Programmstrukturen erarbeitet, aus welchen anschließend die geeignetste für die Umsetzung ausgewählt wurde. Anschließend erfolgte die Programmierarbeit und die Validierung.

2 PROGRAMMBESCHREIBUNG

Folgendes Kapitel behandelt den Pythoncode, welcher es ermöglicht, ein Fachwerk zu generieren. Hierfür werden acht verschiedene Pythonfiles benötigt, welche jeweils nachfolgend vorgestellt werden.

Um ein neues Fachwerk zu generieren muss zunächst das File main.py ausgeführt werden. Dieses greift auf die anderen Files zu und erstellt mit deren Hilfe eine Text-Datei, welche anschließend mit einem Latex-Compiler in eine Pdf-Datei umgewandelt werden muss. Folgender Code ist in main.py beinhaltet.

```
from Python.erstellen import erstellen_doc

latexcode = erstellen_doc("Versuch1", 5, 20, 20)
# erstellen(name_doc, anzahl_punkte <26, breite, höhe)
f = open('Latexcode.txt', 'w')
f.write(latexcode)
f.close()
```

Zu Beginn wird die Funktion erstellen_doc aufgerufen. Dieser muss der Name der zu erstellenden Text-Datei, die Anzahl der Knoten, die Länge und die Breite des Feldes übergeben werden. Die Abmessungen des Feldes sind notwendig, um die Größe des Fachwerks an verschiedenen Papierformate anpassen zu können. An dieser Stelle ist es wichtig zu erwähnen, dass aufgrund er alphabetischen Beschriftung maximal 26 Punkte eingefügt werden können. Im Zuge einer Erweiterung dieses Programmes ist es möglich die Anzahl der Punkte durch doppel und dreifach Buchstaben zu erhöhen.

Die vorhin beschriebene Funktion befindet sich im File erstellen.py und beinhaltet folgenden Code.

```
from Python.bemassung import bemassung
from Python.richtung import *
from Python.Document import Document
from Python.Point import Point
from Python.verbinden import *
import itertools

def erstellen_doc(name_doc, anzahl_punkte, breite, hoehe):
    # erstellt die Documentenklasse
    doc = Document(name_doc)
```

```
punkteliste = erstellen_point(anzahl_punkte, breite, hoehe)
koordinaten_punkte = [] #  nur koordinaten
ymin = hoehe/2
ymax = hoehe/2
xmin = breite
xmax = 0
punkteliste = sorted(punkteliste,
                     key=lambda k: [round(k.x, 3), k.y])

punktdavor_x = punkteliste[0]
for i in punkteliste:
    b = [i.x, i.y]
    under_free = True
    if punktdavor_x == i.x:
        # Kontrolle ob ein Punkt darunter liegt
        under_free = False
    koordinaten_punkte.append(b)
    doc.add_point(i)
    if i.y < ymin:
        ymin = i.y
    if i.y > ymax:
        ymax = i.y
    if i.x < xmin:
        xmin = i.x
    if i.x > xmax:
        xmax = i.x
    punktdavor_x = i.x


#Verbindungen
verbinden_elemente =\
    statisch_bestimmt_verbinden_drei(koordinaten_punkte)
set_verbindungen_index = verbinden_elemente[0]
for i in set_verbindungen_index:
    doc.add_verbindung(punkteliste, i)

for i in punkteliste:
    doc.add_hinge(i)

# Auflager
idx_koord_Lager = verbinden_elemente[1]
winkel_Lager =(verbinden_elemente[2], verbinden_elemente[3])
```

```
    doc.add_lager(idx_koord_Lager, punkteliste, winkel_Lager)

    #Beschriftung nodes
    doc.add_beschriftung(punkteliste, idx_koord_Lager)

    #bemassung
    bemassungskomponenten = bemassung(punkteliste)
    doc.add_bemassung(bemassungskomponenten, ymin, xmin)

    # Rahmen
    doc.add_rahmen(xmin, ymin, xmax, ymax,
                   len(bemassungskomponenten[1][1])-1)

    return doc.get_latex_code()


def erstellen_point(anzahl_punkte, breite, hoehe):
    # erstellt Punkte
    listepunkte = []
    xdavor = 1.000
    ydavor = hoehe / 2

    i = 0
    while i < anzahl_punkte:
        p = Point()
        rich = richtung()
        p.x = round(xdavor + rich[0], 3)
        # + zufällige Richtung x Koordinate
        p.y = round(ydavor + rich[1], 3)
        # + zufällige Richtung y Koordinate
        p.name = chr(ord("A") + i)
        if i > 1:
            a = True
            while a:
                dreierliste = []
                zweierliste = \
                    itertools.combinations(listepunkte, 2)
                for points in zweierliste:
                    dreierliste.append([points[0], points[1], p])

                for points in dreierliste:
                    if arecolinear(points):
```

```
                        a = True
                        rich = richtung()
                        p.x = round(xdavor + rich[0], 3)
                        # + zufällige Richtung x Koordinate
                        p.y = round(ydavor + rich[1], 3)
                        # + zufällige Richtung y Koordinate
                    else:
                        a = False

        p = kontrolle(breite, hoehe, p)
        #Kontrolle ob Punkt im Feld ist
        if nicht_doppelt(p, listepunkte):
            #Kontrolle ob Punkt doppelt ist
            listepunkte.append(p)
            i = 1 + i
            xdavor = p.x
            ydavor = p.y

    return listepunkte



def arecolinear(points):   #linear abhängig
    xdiff1 = float(points[1].x - points[0].x)
    ydiff1 = float(points[1].y - points[0].y)
    xdiff2 = float(points[2].x - points[1].x)
    ydiff2 = float(points[2].y - points[1].y)
    if xdiff1 == 0 or xdiff2 == 0:
        return xdiff1 == xdiff2
    elif ydiff1/xdiff1 - ydiff2/xdiff2 < 0.0001:
        return True
    else:
        return False
```

Zu Beginn wird ein neues Document doc, welchem eine eigene Klasse zugrunde liegt, und eine Punkteliste mit Hilfe der Funktion erstellen_point, welche im nächsten Absatz näher erläutert wird, erzeugt. Anschließend werden von jedem Punkt-Objekt, welches x-Koordinate, y-Koordinate und Beschriftung beinhaltet, die Koordinaten in einer seperaten Liste zwischengespeichert und die einzelnen Punkt-Objekte dem Document doc hinzugefügt. Daraufhin werden die Verbindungen, welche durch die Funktion statisch_bestimmt_verbinden_drei erstellt werden und die Gelenke dem Document doc

hinzugefügt. Schlussendlich werden noch die Auflager, die Bemaßung und der Rahmen des Feldes dem Document hinzugefügt. Die Funktion erstellen_point befindet sich ebenfalls in diesem Pythonfile. Um diese aufzurufen muss die Anzahl der Punkte sowie die Breite und Höhe des Feldes übergeben werden. Anschließend werden in einer while-Schleife solange Punkte erzeugt und bei bestandener Kontrolle der Liste hinzugefügt, bis die gewünschte Anzahl erreicht ist. Ausgehend vom letzen aufgenommenen Punkt wird der nächste neue Punkt in einem Winkel von $-90°$, $-60°$, $-45°$, $-30°$, $0°$, $30°$, $45°$, $60°$ oder $90°$ hinzugefühgt. Dieser neue Richtungsvektor wird durch die Funktion richtung bestimmt. Ein Punkt wird nur nach erfolgreicher Prüfung der Punkteliste hinzugefügt. Diese Kontrolle beinhaltet einerseits die Einhaltung der Grenzen des Feldes und andererseits das Sicherstellen, dass keine linear abhängigen und keine gleichen Punkte in der Liste aufgenommen werden. Ersteres wird durch die Funktion kontrolle erreicht. Zweiteres wird durch die Funktion arecolinear gewährleistet.

Die Funktion richtung ermöglicht das Finden neuer Punkte, die einen bestimmten Abstand zum letzten erfolgreich aufgenommenen Punkt aufweisen. Folgender Code wird hierfür benötigt.

```
from random import randint

def richtung():
    a = [0, 0, 0]  # [x,y,b]
    b = randint(1, 5)
    #b gibt die Richtungen vor. bei 1-3 keine 30,60
    a[2] = b
    if b == 1:  #hor
        a[0] = 1
        a[1] = 0

    elif b == 5:  #30
        a[0] = 0.866
        a[1] = 0.5

    elif b == 3:  #45
        a[0] = 1
        a[1] = 1

    elif b == 4:  #60
        a[0] = 0.5
        a[1] = 0.866

    elif b == 2:  #90
```

```
        a[0] = 0
        a[1] = 1

    len = randint(1, 2)
    quadrant = 1
    if randint(1,2) < 2:
        quadrant = -1

    a[0] = a[0] * len
    a[1] = a[1] * len * quadrant
    return a


def kontrolle(breite, hoehe, point):
    p = point
    if (point.x > breite)or(point.y > hoehe)or(point.y < 0):
        p.x = 1
        p.y = hoehe/2

    return p


def nicht_doppelt(punkt, listepunkte):
    if len(listepunkte) > 0:
        for element in listepunkte:
            if (punkt.x == element.x)and(punkt.y == element.y):
                return False
    else:
        return True
    return True
```

Diese Funktion ist sehr schlicht. Nach dem Aufruf, bei dem keine Argumente übergeben werden müssen, wird zunächst eine Zufallszahl zwischen eins und fünf gewählt. Abhängig von dieser wird daraufhin eine vorgegebene Richtung gewählt. Schlussendlich wird durch eine weitere Zufallszahl der 1. oder 4. Quandrant und die Länge, sprich eins oder zwei, des neuen Richtungsvektors gewählt.

Das gesamte Programm beinhaltet die zwei eignene Klassen, Punkt und Document. Beide sind notwendig, um das Latexfile des Fachwerks ausgeben zu können. Die Klasse Punkt beinhaltet folgenden Code.

```
class Point:

    def __init__(self, x=None, y=None, name=None):
        self.x = x
        self.y = y
        self.name = name
```

Ein Punkt besteht lediglich aus einer x-Koordinate, einer y-Koordinate und einer Bezeichnung.

Die KLasse Document weist folgenden Programmcode auf:

```
#Erstellt das Document, in dem die einzelnen
# Zeilen Code hinzugefügt werden

from Python.con2text import *

# Textbaustein
start_doc = """\\documentclass[varwidth]{standalone}
\\usepackage{amsmath}
\\usepackage{standalone}
\\usepackage{tikz}
\\usetikzlibrary{arrows,snakes,backgrounds,patterns,
matrix,shapes,fit,calc,shadows,plotmarks}
\\usepackage{elementlibrary}
\\usepackage{nicefrac}
\\usepackage{IAM_Lager}
\\begin{document}\n"""
start_tik = """\\newline
\\begin{minipage}{1\\textwidth}
\\begin{center}

\\begin{tikzpicture}[>=latex,scale=0.9]\n"""
end_doc = """
\n\\end{tikzpicture}
\\newline
\\end{center}
\\end{minipage}
\\end{document}"""

title = "\\centering Fachwerk:\n \\newline"
```

```
class Document:
    #Methoden für das Hinzufügen von den Bestandteilen
    def __init__(self, name):
        self.name = name
        self.list = [start_doc, title, start_tik]

    def add_point(self, point):
        self.list.append(coordinate2latex(point))

    def add_rahmen(self, xmin,ymin,xmax,ymax, bemassungsanzahl):
        self.list.append(rahmen2latex(xmin, ymin,
                                      xmax, ymax, bemassungsanzahl))

    def add_hinge(self, point):
        self.list.append(gelenk2latex(point))

    def add_lager(self, idx_lager, punkteliste, winkel):
        self.list.append(lager2latex(idx_lager,
                                     punkteliste, winkel))

    def add_bemassung(self, komponenten, ymin, xmin):
        double = komponenten[0]
        abstand = komponenten[1]
        self.list.append(abmessungen_x_2latex(
            double[0], ymin, abstand[0]))
        self.list.append(abmessungen_y_2latex(
            double[1], xmin, abstand[1]))

    def add_verbindung(self, punkteliste,
                       set_verbindungen_index):
        self.list.append(verbindung2latex_set(
            punkteliste, set_verbindungen_index))

    def get_latex_code(self):
        tmp = ""
        for s in self.list:
            tmp += s
        tmp += end_doc
        return tmp

    def add_beschriftung(self, punkteliste,
                         idx_koord_Lager):
```

```
        self.list.append(beschriftung2latex(punkteliste,
                                            idx_koord_Lager))
```

Hierbei werden zu Beginn die Import-Textbausteine des Latexfiles definiert. Über die Eingabe eines Namens kann ein solches Objekt initialisiert werden. Es besteht einerseits aus dem Namen und andererseits aus einer Liste. Durch verschiedene Methoden kann dieser Liste der jeweil notwendige Latexcode hinzugefügt werden. Jede dieser Methoden greift auf eine Funktion zurück, welche im Namen 2latex beinhaltet.

Diese Funktionen werden im File con2text.py dargestellt und ermöglichen das Überführen der einzelnen Daten, beispielsweise Verbindungen, Punkte oder Abmessungen, in den dazugehörigen Latexcode. Dieses File beinhaltet folgende Zeilen Code.

```
# Satzbausteine für Latex-Code

def coordinate2latex(point):
    a = "\\coordinate (" + point.name + ") at (" + str(point.x) \
        + "," + str(point.y) + ");\n"
    return a


def rahmen2latex(xmin, ymin, xmax, ymax, bemassungsanzahl):
    z = 2.5 #abstand Rahmen
    d = "\n%Rahmen\n"
    a = "\\draw (" + str(xmin-z-bemassungsanzahl*0.7) + "," + \
        str(ymin-z) + \
        ") rectangle (" + str(xmax+z) + "," + str(ymax+z) + ");\n"
    return d + a


def gelenk2latex(point):
    #\Vollgelenk{shift}[scale]
    a = "\\Vollgelenk{" + point.name + "}[0.7]\n"
    return a


def lager2latex(idx_lager, punkteliste, winkel):
    #\Festlager{shift}[rotate][scale]
    #\Loslager{shift}[rotate][scale]
    winkelhora = int(winkel[0]-99)
    winkelhorb = int(winkel[1] - 99)
    a = "\n\\Festlager{" + punkteliste[idx_lager[0]].name +\
```

```
        "}[" + str(winkel[0]) + "][0.65]\n" + \
        "\\node at ($(" + punkteliste[idx_lager[0]].name + \
        ") + (" + str(winkelhora) + \
        ":0.7)$)[xshift=0mm,yshift=0mm]{\\tiny " + \
        punkteliste[idx_lager[0]].name + "};\n"
    b = "\\Loslager{" + punkteliste[idx_lager[1]].name + \
        "}[" + str(winkel[1]) + "][0.65]\n" + \
        "\\node at ($(" + punkteliste[idx_lager[1]].name +\
        ") + (" + str(winkelhorb) + \
        ":0.7)$)[xshift=0mm,yshift=0mm]{\\tiny " + \
        punkteliste[idx_lager[1]].name + "};\n\n"
    awinkel = int(90 -(winkel[0] - 90 * int(winkel[0]/90)))
    c = ""
    d = ""
    e = ""
    rotate = int(90 * int(winkel[0]/90))
    print(awinkel)
    if awinkel == 30 or awinkel == 45 or awinkel == 60:
        c = "\n\\begin{scope}[rotate="+str(rotate)+\
            ", shift = {(" + punkteliste[idx_lager[0]].name\
            + ")}]\n"
        d = "\\draw[dashed, very thin] (0,0)--+(-"+\
            str(awinkel)+":1.4);\n \\draw[dashed, " \
        "very thin] (0,0)--+(-90:1.2);\n " \
        "\\draw[|<->|,thin] ($(0,0)+(-"+str(awinkel)+\
            ":1.)$) arc (-"+str(awinkel)+".1:-90:1.);\n" \
        "\\node at ($(0,0)+(-"+\
            str(awinkel)+":1.)$) [xshift=-3mm,yshift=0mm]{$" +\
            darstellung_winkel(int(winkel[0]
                                   - 90 * int(winkel[0]/90)))\
            +"$};\n"
        e="\\end{scope}"

    bwinkel = int(90 - (winkel[1] - 90 * int(winkel[1]/90)))
    bc = ""
    bd = ""
    be = ""
    rotate = int(90 * int(winkel[1]/90))
    print(bwinkel)
    if bwinkel == 30 or bwinkel == 45 or bwinkel == 60:
        bc = "\\begin{scope}[rotate="+str(rotate)+\
             ", shift = {(" + punkteliste[idx_lager[1]].name\
```

```
            + ")}]\n"
        bd = "\\draw[dashed, very thin] (0,0)--+(-"+\
            str(bwinkel)+":1.4);\n \\draw[dashed, " \
        "very thin] (0,0)--+(-90:1.2);\n " \
        "\\draw[|<->|,thin] ($(0,0)+(-"+str(bwinkel)+\
            ":1.)$) arc (-"+str(bwinkel)+\
            ".1:-90:1.);\n" \
        "\\node at ($(0,0)+(-"+str(bwinkel)+\
            ":1.)$) [xshift=-3mm,yshift=0mm]{$" + \
            darstellung_winkel(int(winkel[1]
                                   - 90 * int(winkel[1]/90)))\
            +"$};\n"
        be= "\\end{scope}"

    return a + b + c + d + e + bc + bd + be


def verbindung2latex_set(punkteliste, set_verbindungen_index):
    a = "\\draw (" + punkteliste[set_verbindungen_index[0]].name\
        + ")--(" + \
        punkteliste[set_verbindungen_index[1]].name + ");\n"
    return a


def abmessungen_x_2latex(liste_x_buchstaben, ymin,
                         abstandx_zw_punkten):
    #\dimensioning{hor = 1, vert. = 2 }{c}{d}{-9.5}
    # [\small$\ell$];
    a = "\n%Bemassung\n"
    i = 1
    yabstand = ymin - 2
    while i < len(liste_x_buchstaben):
        a = a + "\\Bemassung{" + liste_x_buchstaben[i - 1] +\
            "}{" + liste_x_buchstaben[i] + \
            "}{\\tiny $" + \
            str(darstellung_zahl(abstandx_zw_punkten[i])) + \
            "a$}[" + str(yabstand) + "][1];\n"

        i = i + 1
    return a
```

```
def abmessungen_y_2latex(liste_y_buchstaben, xmin,
                         abstandy_zw_punkten):
    #\dimensioning{hor = 1, vert. = 2 }{c}{d}{-9.5}
    # [\small$\ell$];
    a = ""
    i = 1
    while i < len(liste_y_buchstaben):
        a = a + "\\Bemassung{" + liste_y_buchstaben[i - 1] +\
            "}{" + liste_y_buchstaben[i] +\
            "}{\\tiny $" + \
            str(darstellung_zahl(abstandy_zw_punkten[i])) + \
            "a$}[" + str(xmin-0.8-i*0.7) +\
            "][2][midway, sloped, above];\n"
        i = i + 1
    return a

def darstellung_zahl(zahl): # Zahl in Bruch umwandeln
    if 0.499 < zahl < 0.5001:
        return "\\frac{1}{2}"
    if int(zahl*10)%10 == 8:
        return "\\frac{\\sqrt3}{2}"
    if 1.73 < zahl < 1.74:
        return "\\sqrt3"
    if 0.365 < zahl < 0.367001:
        return "\\frac{\\sqrt3-1}{2}"
    if round(zahl,0) == 1.0:
        return "1"
    if round(zahl,0) == 2.0:
        return "2"
    if  0.133 < zahl < 0.135:
        return "\\frac{2-\\sqrt3}{2}"
    if 0.133*2 < zahl < 0.135*2:
        return "2-\\sqrt3"
    if 0.365*2 < zahl < 0.367001*2:
        return "\\sqrt3-1"
    return round(zahl, 2)

def darstellung_winkel(winkel):
    if winkel == 30:
        return "\\frac{\pi}{6}"
    if winkel == 45:
        return "\\frac{\pi}{4}"
```

```
    if winkel == 60:
        return "\\frac{\pi}{3}"



def verbindung2latex(punkteliste, liste_verbindungen_index):
    a = "\\draw (" + \
        punkteliste[liste_verbindungen_index[0]].name + ")--(" \
        + \
        punkteliste[liste_verbindungen_index[2]].name + ")--(" \
        + punkteliste[liste_verbindungen_index[1]].name + ");\n"
    return a


def belastung2latex(point):
    a = "\\draw[ <-, very thick] (" + point.name \
        + ") -- +(0., 1.2) node[midway, right] {$F$};\n"
    return a

def beschriftung2latex(punkteliste, idx_koord_Lager):
    print("Punkteliste")
    punkteliste = sorted(punkteliste, key=lambda k: [k.x, k.y])
    for i in punkteliste:
        print (i.x)
        print(i.y)
        print("")
    punktdavor_x = punkteliste[0].x
    c = ""
    for point in punkteliste:
        if not (punkteliste.index(point) in idx_koord_Lager):
            under_free = True
            if punktdavor_x == point.x and \
                    punkteliste.index(point) > 0:
                under_free = False

            if under_free:
                c = c + "\\node at (" + point.name \
                    + ")[xshift=0mm,yshift=-2mm]{\\tiny " \
                    + point.name + "};\n"
            else:
                c = c + "\\node at (" + point.name \
                    + ")[xshift=0mm,yshift=2mm]{\\tiny " \
```

```
                    + point.name + "};\n"
        punktdavor_x = point.x
    return c
```

Weiters wird ersichtlich, dass in diesem File zwei weitere Funktionen beinhaltet sind, welche für die Bruchdarstellung der Bemaßung und der Winkel benötigt werden.

Die Berechung der Parameter für die Bemaßung erfolgt im File bemassung.py, welches nachfolgend gezeigt wird.

```
def bemassung (punkteliste):# gibt die Anfang und
    #  Endpunkte, sowie den Abstand zwischen diesen wieder
    y_sortiert_punkteliste = sorted(punkteliste,
                                    key=lambda k: [k.y])
    x = set()
    y = set()
    x_buchstaben = []
    y_buchstaben = []
    x_abstand = []
    y_abstand = []
    alterpunktx = punkteliste[0]
    alterpunkty = y_sortiert_punkteliste[0]
    for punkt in punkteliste:
        lengthx = len(x)
        x.add(punkt.x)
        if lengthx < len(x):
            x_buchstaben.append(punkt.name)
            x_abstand.append(punkt.x-alterpunktx.x)
            alterpunktx = punkt


    for punkt in y_sortiert_punkteliste:
        lengthy = len(y)
        y.add(punkt.y)
        if lengthy < len(y):
            y_buchstaben.append(punkt.name)
            y_abstand.append(abs(punkt.y-alterpunkty.y))
            alterpunkty = punkt
    x_y_buchstaben = (x_buchstaben, y_buchstaben)
    abstand = (x_abstand, y_abstand)
    print(x_y_buchstaben)
    print(abstand)
```

```
    return x_y_buchstaben, abstand
```

Dieser Funktion ist die zu Beginn erstellte Punkteliste zu übergeben. Für die x und y Bemassung wird jeweils ein Set erstellt, in dem die x-Koordinaten und y-Koordinaten der sortierten Punkte gespeichert werden. Weiters wird eine Liste erzeugt, in welcher der Name der Punkte gespeichert wird, falls die Koordinaten dem Set hinzugefügt werden. Das heißt, ist eine Koordinate bereits im Set beinhaltet, kann diese nicht nochmals hinzugefügt werden, weil jedes Element in einem Set nur einmal vorkommen kann. Verändert sich die Länge des Sets somit nicht, wird der Buchstabenliste kein Element hinzugefügt. Neben den Buchstaben werden ebenfalls die dazugehörigen Abmessungen zwischen den Punkten in einer Liste gespeichert. Schlussendlich werden die Liste der Buchstaben und die passenden Abstände für die horizontalen und vertikalen Beschriftungen ausgegeben.

Schlussendlich wird noch die Funktion statisch_bestimmt_verbinden_drei, welche sich im Pythonfile verbinden.py befindet, vorgestellt. Dieser ist die Koordinatenliste der einzelnen Punkte zu übergeben, damit die Indizes der Verbindungen, die Platzierungen der Auflager, sowie deren Verdrehungen zurückgegeben werden können. Um die Punkte, welche vorgegebene Abstände zueinander aufweisen, statisch bestimmt und ohne Überschneidungen zu verbunden, werden konvexe Hüllen zu Hilfe genommen. Ausgehend von den ersten zwei Punkten der nach x sortierten Koordinatenliste wird zunächst der folgende Punkt in der Liste, die den Namen verbunden trägt, aufgenommen. Anschließen wird aus dieser Liste eine konvexe Hülle erstellt und die Verbindungen in einem Set gespeichert. Nachdem alle Punkte der sortierten Koordinatenliste in verbunden beinhaltet sind, weist das Set alle dazugehörigen Zusammenhänge auf, wobei keine doppelt vorkommen. Nach dem Erzeugen der Verbindungen gilt es die Auflager zu platzieren. Hierfür werden aus der letzen konvexen Hülle zwei miteinander verbundene Punkte ausgewählt und deren Verbindung gelöscht, um die statische Bestimmtheit zu gewährleisten. Nach dem Aussuchen der Lagerpunkte wird die Verdrehung der Lager bestimmt. Dies erfolgt über das Suchen nach dem größten freien Winkel zwischen den Verbinungen, die von diesem Punkt weggehen. Hierfür ist die Funktion winkel, welcher eine Vektorliste der Vektoren zu allen verbundenen Punkten zu übergeben ist, geschrieben worden. Diese Funktion berechnet die Winkel eines jeden Vektors zur horizontalen, zieht diese voneinander ab und sucht den größten aus. Dieser wird anschließend halbiert und alle vorherigen Winkel werden dazuaddiert. Um zu gewährleisten, dass der Winkel ein vielfachen von $0°$, $30°$, $45°$, $60°$ oder $90°$ beträgt wird schlussendlich noch die Funktion schoene_winkel auf den zuvor berechneten Winkel angewandt. Diese sucht den nächstgelegensten Winkel.

3 ERGEBNIS FACHWERK

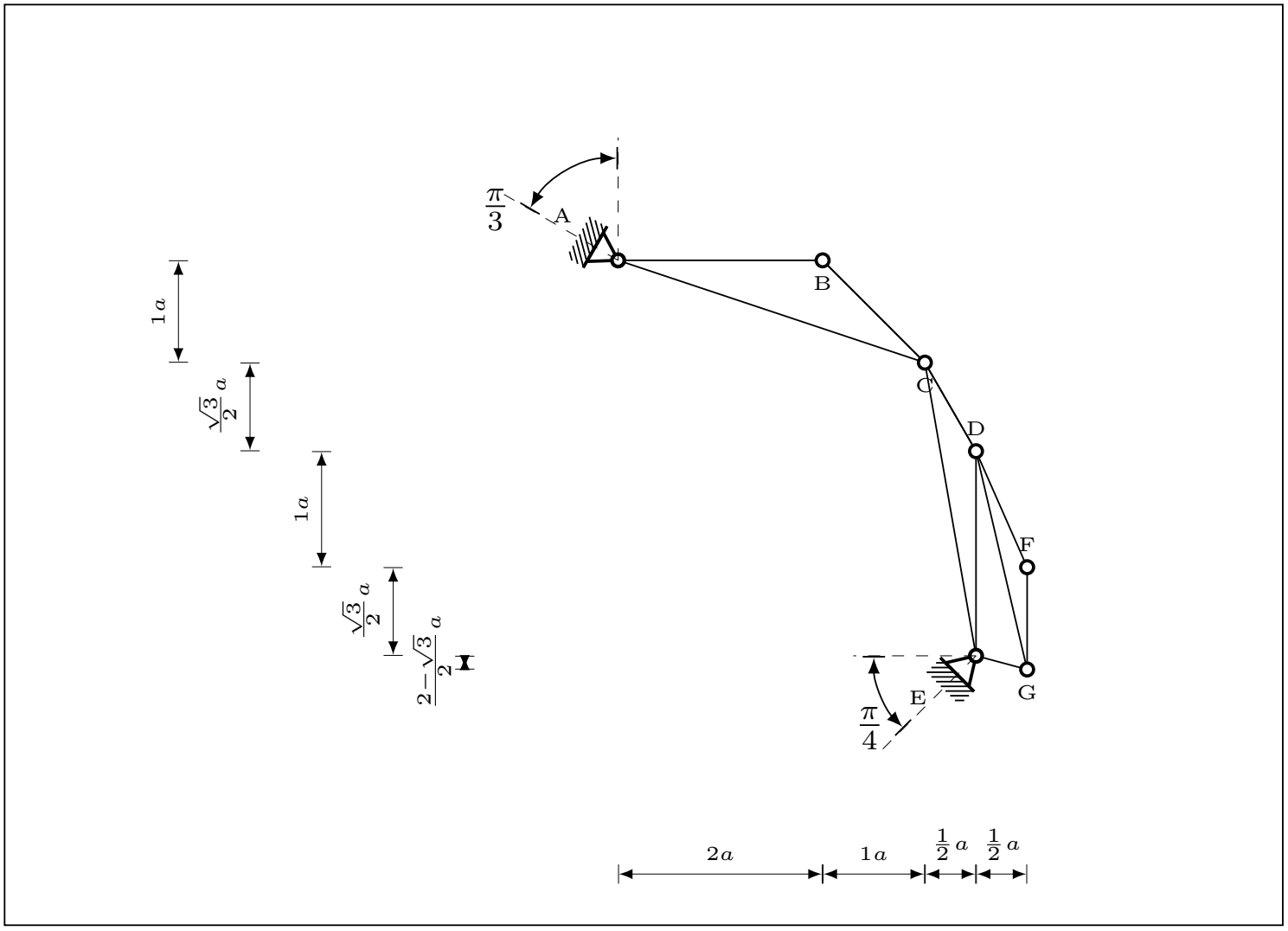
Abschließend wird in diesem Kapitel ein generiertes Fachwerk und der dazugehörige Latexcode gezeigt. Dieses Fachwerk besitzt 5 zufällige Punkte.

```
\documentclass[varwidth]{standalone}
\usepackage{amsmath}
\usepackage{standalone}
\usepackage{tikz}
\usetikzlibrary{arrows,snakes,backgrounds,patterns,
matrix,shapes,fit,calc,shadows,plotmarks}
\usepackage{elementlibrary}
\usepackage{nicefrac}
\usepackage{IAM_Lager}
\begin{document}
\centering Fachwerk:
 \newline\newline
\begin{minipage}{1\textwidth}
\begin{center}

\begin{tikzpicture}[>=latex,scale=0.9]
\coordinate (A) at (3.0,10.0);
\coordinate (B) at (5.0,10.0);
\coordinate (C) at (6.0,9.0);
\coordinate (E) at (6.5,6.134);
\coordinate (D) at (6.5,8.134);
\coordinate (G) at (7.0,6.0);
\coordinate (F) at (7.0,7.0);
\draw (B)--(C);
\draw (F)--(D);
\draw (D)--(G);
\draw (C)--(B);
\draw (C)--(A);
\draw (C)--(E);
\draw (D)--(E);
\draw (D)--(C);
\draw (B)--(A);
\draw (C)--(D);
\draw (F)--(G);
\draw (E)--(G);
```

```
\Vollgelenk{A}[0.7]
\Vollgelenk{B}[0.7]
\Vollgelenk{C}[0.7]
\Vollgelenk{E}[0.7]
\Vollgelenk{D}[0.7]
\Vollgelenk{G}[0.7]
\Vollgelenk{F}[0.7]

\Festlager{E}[315.0][0.65]
\node at ($(E) + (216:0.7)$)[xshift=0mm,yshift=0mm]{\tiny E};
\Festlager{A}[240.0][0.65]
\node at ($(A) + (141:0.7)$)[xshift=0mm,yshift=0mm]{\tiny A};


\begin{scope}[rotate=270, shift = {(E)}]
\draw[dashed, very thin] (0,0)--+(-45:1.4);
 \draw[dashed, very thin] (0,0)--+(-90:1.2);
 \draw[|<->|,thin] ($(0,0)+(-45:1.)$) arc (-45.1:-90:1.);
\node at ($(0,0)+(-45:1.)$) [xshift=-3mm,yshift=0mm]{$\frac{\pi}{4}$};
\end{scope}\begin{scope}[rotate=180, shift = {(A)}]
\draw[dashed, very thin] (0,0)--+(-30:1.4);
 \draw[dashed, very thin] (0,0)--+(-90:1.2);
 \draw[|<->|,thin] ($(0,0)+(-30:1.)$) arc (-30.1:-90:1.);
\node at ($(0,0)+(-30:1.)$) [xshift=-3mm,yshift=0mm]{$\frac{\pi}{3}$};
\end{scope}\node at (B)[xshift=0mm,yshift=-2mm]{\tiny B};
\node at (C)[xshift=0mm,yshift=-2mm]{\tiny C};
\node at (D)[xshift=0mm,yshift=2mm]{\tiny D};
\node at (G)[xshift=0mm,yshift=-2mm]{\tiny G};
\node at (F)[xshift=0mm,yshift=2mm]{\tiny F};

%Bemassung
\Bemassung{A}{B}{\tiny $2a$}[4.0][1];
\Bemassung{B}{C}{\tiny $1a$}[4.0][1];
\Bemassung{C}{E}{\tiny $\frac{1}{2}a$}[4.0][1];
\Bemassung{E}{G}{\tiny $\frac{1}{2}a$}[4.0][1];
\Bemassung{G}{E}{\tiny $\frac{2-\sqrt3}{2}a$}[1.5000000000000002][2][midway, slope
\Bemassung{E}{F}{\tiny $\frac{\sqrt3}{2}a$}[0.8000000000000003][2][midway, sloped
\Bemassung{F}{D}{\tiny $1a$}[0.10000000000000053][2][midway, sloped, above];
\Bemassung{D}{C}{\tiny $\frac{\sqrt3}{2}a$}[-0.5999999999999996][2][midway, slope
\Bemassung{C}{A}{\tiny $1a$}[-1.2999999999999998][2][midway, sloped, above];

%Rahmen
```

```
\draw (-3.0,3.5) rectangle (9.5,12.5);


\end{tikzpicture}
\newline
\end{center}
\end{minipage}
\end{document}
```

Fachwerk:


A
B
C
D
E
F
G
$\frac{\pi}{3}$
$\frac{\pi}{4}$
$1a$
$\frac{\sqrt{3}}{2}a$
$1a$
$\frac{\sqrt{3}}{2}a$
$\frac{2-\sqrt{3}}{2}a$
$2a$
$1a$
$\frac{1}{2}a$
$\frac{1}{2}a$

REFERENCES

[1] Dietmar Gross, Werner Hauger, and Peter Wriggers. *Technische Mechanik 4 Hydromechanik, Elemente der Hheren Mechanik, Numerische Methoden*. Springer-Verlag, Berlin Heidelberg, 2011.

[2] Hauger Gross, Werner Hauger, J Schröder, and WA Wall. *Technische Mechanik 2 Elastostatik*. Springer-Verlag, Berlin Heidelberg, 2011.