Institute of
Applied Mechanics
Institut für Baumechanik

Graz University of Technology

Bachelorarbeit im Studiengang Bauingenieurwissenschaften
am Institut für Baumechanik

Technische Universität Graz

# Automatische Beschriftung von Mechanikaufgaben

Martin Weber

Graz, 12. November 2024

Betreuer: Assoc.Prof. Dipl.-Ing. Dr.techn.
Benjamin Marussig

# Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Graz, _____ _____

Date Signature

# Eidesstattliche Erklärung[1]

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz, am _____ _____

Datum Unterschrift

[1] Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008; Genehmigung des Senates am 1.12.2008

## Abstract

The objective of this thesis is to further develop an existing software for the automatic generation of mechanical exercise assignments, created by Michael H. Gfrerer at the Institute of Applied Mechanics at TU Graz. The program enables the generation and visualisation of beam structures and is written in the Python programming language. The TikZ package of LaTeX is used for the graphical display of these beam structures.

However, a shortcoming of the program is the frequent overlap of labels with other graphic elements or with each other, which can result in a cluttered and unclear visualisation. The objective of this work is, therefore, to enhance the display quality of these structures by implementing a method for the automatic placement of labels that prevents such overlaps.



## Zusammenfassung

Die vorliegende Arbeit verfolgt das Ziel, eine Software zur automatischen Generierung von Mechanikaufgaben, welche ursprünglich von Michael H. Gfrerer am Institut für Baumechanik der TU Graz entwickelt wurde, weiterzuentwickeln.

Das Programm ermöglicht die Generierung bzw. Darstellung unterschiedlicher Stabtragwerke mit verschiedenen Belastungen. Im Rahmen dieser Arbeit wurde die Darstellung der erzeugten Grafiken optimiert indem eine Systematik für die automatisierte Positionierung diverser Beschriftungen implementiert wurde. Die Umsetzung dieser Funktionalität erfolgte unter Verwendung der Programmiersprache Python und LaTeX mit TikZ wird verwendet um die Grafiken zu erzeugen.

# INHALTSVERZEICHNIS

# 1. EINLEITUNG

Das bestehende Programm [1] hat die Funktionalität, ein PDF-Dokument mit einer grafischen Darstellung eines Stabtragwerkes zu generieren. Dabei werden die Spezifikationen des Stabtragwerks aus einer Python-Datei als Input verwendet. Das Stabtragwerk kann gerade und gekrümmte Stäbe, verschiedene Auflagertypen, Belastungsfunktionen und Einzellasten, Koordinatensysteme sowie die dazugehörige Beschriftung umfassen.

Die Funktionsweise des Programms lässt sich grob wie folgt skizzieren: Aus einer Angabedatei werden alle Daten entnommen, welche die Bestandteile des Systems, deren Anfangs- und Endkoordinaten, die Art der Verbindung (gelenkig oder starr) sowie die Auflager und Belastungen auf den jeweiligen Stäben beschreiben.

Die grafische Darstellung des Systems erfolgt unter Verwendung von LaTeX mit TikZ. Dabei wird für jeden Bestandteil des Systems in sequentieller Abfolge ein der LaTeX- und TikZ-Syntax entsprechender String in ein .tex-Dokument geschrieben. Hierbei sind auch Angaben für die notwendigen bzw. gewünschten Beschriftungen enthalten.

Die Angabe der Position für die Beschriftung erfolgt jedoch ohne Berücksichtigung der Tatsache, dass im weiteren Verlauf weitere Elemente der Grafik, inklusive weiterer Beschriftungen, in das LaTeX-Dokument bzw. zur TikZ-Grafik hinzugefügt werden. Dies kann zur Folge haben, dass Beschriftungen aufgrund von Überlappungen mit anderen Beschriftungen oder sonstigen Elementen der Grafik nicht lesbar sind.

Das Ziel dieser Arbeit bestand in der Weiterentwicklung des bestehenden Programms, sodass automatisch eine passende Position für jede Beschriftung gefunden wird und eine Überlappung von Beschriftungen mit anderen Beschriftungen oder sonstigen Komponenten der grafischen Darstellung des Stabsystems in den erzeugten PDF-Dateien ausgeschlossen werden kann.

Die Positionsfindung für die Beschriftungen muss automatisiert erfolgen, da das Programm dafür verwendet werden soll eine große Anzahl individueller Hausübungsangaben zu generieren und der Aufwand die Beschriftung in jeder Angabe händisch anzupassen hoch wäre.

# 2. AUFBAU UND FUNKTIONSWEISE DES PROGRAMMS

## 2.1 Benötigte Software

Das Programm verwendet Funktionalitäten aus mehreren Python-Libraries, welche nicht in der Python Standard Library enthalten sind. Daher ist eine separate Installation der nachfolgend aufgeführten Python-Libraries erforderlich:

- sympy
- shapely
- matplotlib
- PIL
- numpy
- yaml

Des Weiteren ist die Installation einer LaTeX-Distribution erforderlich, welche das kompilieren einer .tex-Datei über die Kommandozeile mit dem Befehl *pdflatex* ermöglicht.

## 2.2 Settings Datei

Zum Programm gehört eine Datei *settings.yaml*. In dieser Datei können verschiedene Einstellungen getroffen werden, die die Qualität der Ergebnisse und die Laufzeit des Programms beeinflussen.

Abbildung 2.1 zeigt einen Auszug der Settings-Datei mit den maßgebenden Parametern zur Beeinflussung des Programmablaufs.

Die Variable *movement_distance* beeinflusst wie hoch die Präzision ist, mit der das Programm arbeitet, weitere Erläuterungen folgen in Kapitel 2.4.2. Dieser Wert wird in Zentimetern angegeben.

Die variable *scale* beschreibt den Skalierungsfaktor der vom Programm generierten Grafik. Ein Wert von Eins entspricht dabei 100% und bedeutet, dass die vom Programm generierte Grafik Maßstabsgetreu dargestellt wird. Bei einem Skalierungsfaktor, der ungleich Eins ist, werden alle Bestandteile der Grafik entsprechend vergrößert oder verkleinert, mit Ausnahme der Beschriftung. Deren Schriftgröße bleibt konstant, lediglich ihre Position wird entsprechend dem Skalierungsfaktor angepasst.

```yaml
#YAML settings file
---
movement_distance: 0.1 #distance in cm

scale: 1 #1=100%

text_priority:
  node: 1
  global_coordinate_system: 11
  local_coordinate_system: 22
  control_variable: 33
  arc_element_angle: 44
  load: 55

iam_lager_polygons:
  festlager:
    p1: [0, 0]
    p2: [-0.25, -0.4]
    p3: [-0.36, -0.4]
    p4: [-0.36, -0.7]
    p5: [0.36, -0.7]
    p6: [0.36, -0.4]
    p7: [0.25, -0.4]

  loslager:
    p1: [0, 0]
    p2: [-0.25, -0.4]
    p3: [-0.36, -0.4]
    p4: [-0.36, -0.8]
    p5: [0.36, -0.8]
    p6: [0.36, -0.4]
    p7: [0.25, -0.4]

  einspannung:
    p1: [-0.36, 0]
    p2: [-0.36, -0.3]
    p3: [0.36, -0.3]
    p4: [0.36, 0.0]
```

Abbildung 2.1: Auszug setting.yaml Datei

Im Abschnitt *text_priority* erfolgt die Definition der Prioritäten verschiedener Beschriftungen. Der Zahlenwert gibt die Priorität der Beschriftungen an, die zu der jeweiligen Kategorie gehören. Dabei gilt je kleiner der Zahlenwert, desto höher die Priorität. Eine höhere Priorität bzw. Wichtigkeit bewirkt, dass diese Beschriftungen zuerst platziert werden. Bei Beschriftungen niedriger Priorität ist die Grafik zum Zeitpunkt an dem sie positioniert werden bereits dichter belegt, wodurch die Wahrscheinlichkeit, dass die optimale Position bereits durch andere Beschriftungen blockiert ist, höher ist. Das hat zur Folge, dass Beschriftungen mit niedriger Priorität meist weiter von ihrer optimalen Position entfernt platziert werden verglichen mit Beschriftungen, die eine hohe Priorität aufweisen.

| **Bezeichnung** | **Darstellung** | **Darstellung mit Begrenzungspolygon** |
|---|---|---|
| Festlager |  |  |
| Loslager |  |  |
| Einspannung |  |  |
| Parallelfuehrung |  |  |
| Schiebehülse |  |  |
| Wegfeder |  |  |
| Drehfeder |  |  |

Tabelle 2.1: Lagertypen und Begrenzungspolygone

Der letzte Abschnitt in der Settings-Datei ist die Kategorie *iam_lager_polygons*. In diesem Abschnitt werden für die verschiedenen Lagertypen jeweils Polygonpunkte mit Koordinatenwerten definiert. Die aus der Verbindung dieser Punkte resultierenden Polygone entsprechen einer Näherung der Umrisse, welche die Lager in der Grafik aufweisen. Beschriftungen sind innerhalb dieser Polygone nicht zulässig, da dies die Lesbarkeit beeinträchtigen würde. Die implementierten Lagertypen und ihre zugehörigen Begrenzungspolygone sind in Tabelle 2.1 dargestellt.

## 2.3 Nachbildung der LaTeX TikZ Grafik mit Geometrieobjekten in Python

Um die Belegung der Bereiche in der mittels LaTeX und TikZ generierten Grafik durch verschiedene Objekte wie Auflager, gerade oder gekrümmte Stäbe, Darstellungen von Belastungen und Koordinatensysteme zu ermitteln und festzulegen, wo keine Beschriftung platziert werden kann, ist eine Abbildung der Elemente der Grafik bzw. der Geometrie dieser Grafik innerhalb des Python-Programms erforderlich. Mit diesen Geometriedaten wird im Anschluss die geeigneten Positionen für die verschiedenen Beschriftungen der Grafik ermittelt.

Es wird zwischen zwei verschiedene Arten von Geometrieobjekten unterschieden. Eine Art der Geometrieobjekte ist für Beschriftungen vorgesehen und die andere für alle übrigen Elemente der Grafik. Im Verlauf des Programmablaufs erfolgt eine schrittweise Übertragung sämtlicher Elemente der Grafik in ein LaTeX-Dokument, wobei zugleich die entsprechenden Geometrieobjekte, welche die einzelnen Elemente repräsentieren, erstellt und gespeichert werden.

Die Beschriftungen werden zu diesem Zeitpunkt noch nicht in das LaTeX-Dokument integriert, da die genaue Positionierung erst stattfinden kann wenn alle Geometrieobjekte, die nicht dem Typ Beschriftung zuzuordnen sind, platziert wurden. Stattdessen wird für Beschriftungen vorerst lediglich ein Geometrieobjekt in Python erstellt, welches Informationen zum Inhalt, zur Priorität, Farbe, Boundingbox, sowie zur optimalen Position der Beschriftung enthält. Unter optimaler Positionierung ist in diesem Kontext jene Position zu verstehen, an der die Beschriftung platziert werden würde, sofern diese Position nicht durch andere Objekte blockiert ist. Als Bounding Box wird das kleinstmögliche achsenparallele Rechteck, das den Text einschließt bezeichnet.

Die Ermittlung der Bounding Box der einzelnen Beschriftungen erfolgt unter Zuhilfenahme der in den Python-Bibliotheken matplotlib und PIL vorhandenen Funktionen. Dabei wird die Möglichkeit der Python-Bibliothek matplotlib genutzt, Text mithilfe von LaTeX zu rendern indem ein Bild erzeugt wird, welches den mit LaTeX gerenderten Text enthält. Anschließend wird mit der Python-Bibliothek PIL die Bounding Box des im Bild enthaltenen Textes ermittelt. Da das Programm beim geplanten Einsatzzweck eine Vielzahl ähnlicher Stabsysteme mit sehr ähnlichen oder identen Beschriftungen generieren soll, wurde nach einer Möglichkeit gesucht, die Prozedur nicht für jeden Text zu wiederholen zu müssen. Deshalb wird der Inhalt bzw. String der Texte mit den jeweiligen Abmessungen der zugehörigen Bounding Box in einer CSV-Datei gespeichert. Bei der erneuten Erzeugung eines Textobjekts mit gleichem Inhalt bzw. String ist folglich keine erneute Berechnung der Bounding Box erforderlich. Stattdessen kann der Inhalt direkt aus der CSV-Datei ausgelesen werden.

## 2.4 Algorithmus zur Textpositionierung

Nachdem alle Objekte, die nicht dem Typ Beschriftung zuzuordnen sind, in das LaTeX-Dokument integriert wurden und für alle diese Elemente ein dazu passendes Geometrieobjekt in Python erstellt wurde, kann mit der Bestimmung der Positionen für die Textobjekte begonnen werden. Die Ermittlung der Position der Textobjekte erfolgt auf Basis der in Algorithmus 1 mit Pseudocode dargestellten Methodik bzw. mit der Python-Methode aus Abbildung 2.2. Im Folgenden werden die grundlegenden Operationen erläutert, die für jedes einzelne Textobjekt durchgeführt werden.

**Algorithmus 1 :** Text positioning algorithm

```
Input : fid, settings
movement_distance ← settings["movement_distance"] ;
move_counter ← 0 ;
for each text in text_objects do
    round_counter ← 0 ;
    position_counter ← 0 ;
    found_position ← False ;
    while found_position is False do
        collision ← False ;
        for each geometry in geometry_objects do
            if bbox_overlap(text.get_bbox(), geometry.get_bbox()) is True then
                while shapely.intersects(text.get_collision_geometry(),
                  geometry.get_collision_geometry()) do
                    collision ← True ;
                    (round_counter, position_counter) ←
                      text.move(movement_distance, round_counter, position_counter)
                      move_counter ← move_counter + 1 ;
        if collision is False then
            found_position ← True ;
    text.print_latex(fid) ;
    geometry_objects.append(text) ;
```

In einem ersten Schritt wird ermittelt, ob sich das Textobjekt bzw. dessen Bounding Box mit der Bounding Box eines anderen Geometrieobjekts überlappt, siehe dazu Kapitel 2.4.1. Sofern es keine Überlappungen gibt kann das Textobjekt an dieser Position platziert werden. Sollte es zu Überlappungen kommen ist eine genauere Kollisionskontrolle notwendig. Dabei wird überprüft, ob es zwischen der Bounding Box des zu platzierenden Textes und der tatsächlichen Geometrie des Objekts, bei dem eine Überlappung der Bounding Boxen festgestellt wurde, zu einer Kollision bzw. Überschneidung kommt.

Sollte es zu keiner Kollision kommen kann der Text an der vorgesehenen Position platziert werden. Sollte jedoch eine Kollision vorliegen, ist eine Verschiebung des Textes erforderlich. Die Verschiebung wird entsprechend der in Kapitel 2.4.2 beschriebenen Methodik durchgeführt. Im Anschluss an jede durchgeführte Verschiebung ist eine erneute Kollisionskontrolle der Bounding Boxen sowie gegebenenfalls die Kollisionskontrolle der exakten Geometrien notwendig.

```
1  def find_text_position(self, fid, settings):
2    movement_distance = settings["movement_distance"]
3    move_counter = 0
4    for text in self.text_objects:
5      round_counter = 0
6      position_counter = 0
7      found_position = False
8      while(not found_position):
9        collision = False
10       for geometry in self.geometry_objects:
11         if bbox_overlap(text.get_bbox(), geometry.get_bbox()):
12           while shapely.intersects(text.get_collision_geometry(),
13                                    geometry.get_collision_geometry()):
14             collision = True
15             round_counter, position_counter = text.move(movement_distance,
16                                                         round_counter,
17                                                         position_counter)
18             move_counter += 1
19       if not collision:
20         found_position = True
21     text.print_latex(fid)
22     self.geometry_objects.append(text)
```

Abbildung 2.2: Funktion zur Positionsbestimmung von Textobjekten

### 2.4.1 Überlappung von Bounding Boxen

Jedem Element der Grafik bzw. dem Python-Geometrieobjekt, durch welches es repräsentiert wird, ist eine Bounding Box zugewiesen. Die Bounding Box stellt dabei das kleinstmögliche achsenparallele Rechteck dar, von dem das betreffende Geometrieobjekt vollständig eingeschlossen wird.

Im ersten Schritt erfolgt keine Überprüfung, ob eine Beschriftung an der vorgesehenen Position eine Überschneidung mit der tatsächlichen exakten Geometrie bereits vorhandener Grafikbestandteile aufweisen würde. Stattdessen wird zunächst analysiert, ob eine Überschneidung mit der Bounding Box eines bereits vorhandenen Bestandteils der Grafik gegeben ist.

Diese Vorgehensweise wurde aus Effizienzgründen gewählt, um nicht mit allen Geometrieobjekten eine Kollisionskontrolle durchführen zu müssen, sondern bereits vorab die grundsätzlich dafür in Frage kommenden Geometrieobjekte eingrenzen zu können.

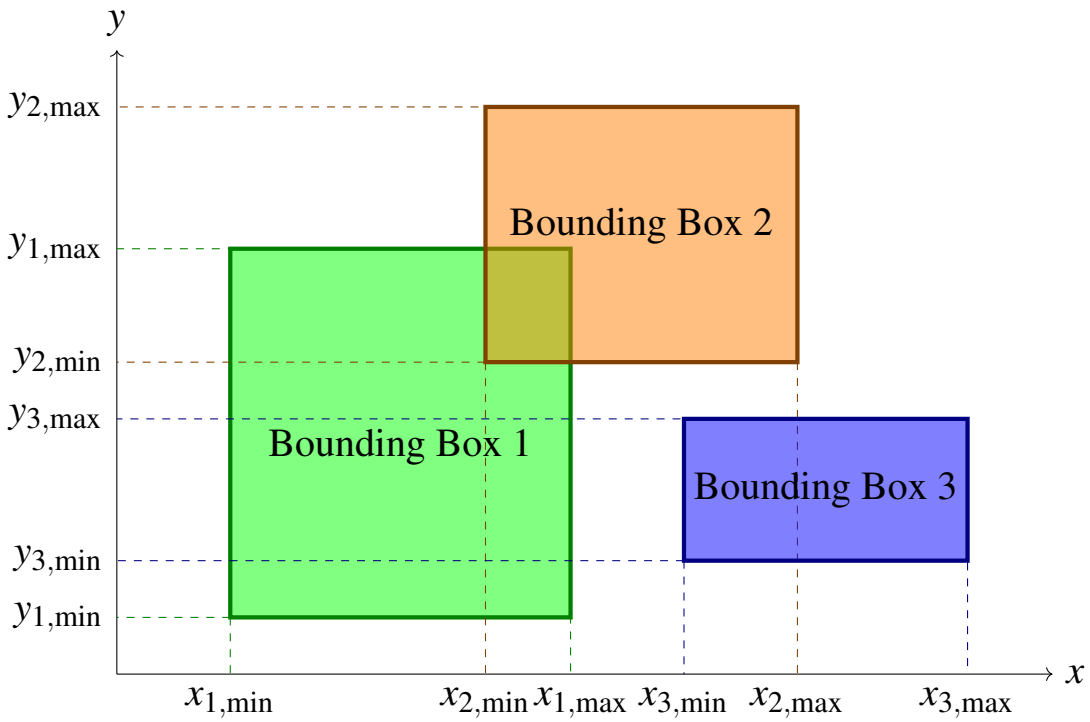



Abbildung 2.3: Beispielhafte Bounding-Boxen

Um zu überprüfen, ob sich zwei Rechtecke beziehungsweise Bounding Boxen überlappen sind vier Bedingungen zu berücksichtigen. Eine Überlappung liegt nur dann vor, wenn alle vier Bedingungen erfüllt sind. Für die Überprüfung der Überlappung zweier Rechtecke $a$ und $b$ stellen sich diese vier Bedingungen wie folgt dar:

$$x_{\min,a} \leq x_{\max,b} \quad \text{und} \quad x_{\max,a} \geq x_{\min,b} \quad \text{und} \quad y_{\min,a} \leq y_{\max,b} \quad \text{und} \quad y_{\max,a} \geq y_{\min,b}$$

Umgelegt auf die und Abbildung 2.3 dargestellten Bounding Boxen ergibt sich beispielsweise:

$$x_{\min,1} \leq x_{\max,2} \quad \text{trifft zu}$$

$$x_{\max,1} \geq x_{\min,2} \quad \text{trifft zu}$$

$$y_{\min,1} \leq y_{\max,2} \quad \text{trifft zu}$$

$$y_{\max,1} \geq y_{\min,2} \quad \text{trifft zu}$$

daraus folgt, dass sich die Bounding Boxen 1 und 2 überlappen.

$$x_{\min,2} \leq x_{\max,3} \quad \text{trifft zu}$$

$$x_{\max,2} \geq x_{\min,3} \quad \text{trifft zu}$$

$$y_{\min,2} \leq y_{\max,3} \quad \text{trifft nicht zu}$$

$$y_{\max,2} \geq y_{\min,3} \quad \text{trifft zu}$$

daraus folgt, dass sich die Bounding Boxen 2 und 3 nicht überlappen.

### 2.4.2 Verschiebealgorithmus

Im Falle einer Kollision des zu platzierenden Textobjekts mit einem anderen Geometrieobjekt muss eine Verschiebung des Textobjekts erfolgen. Dies erfolgt in Form einer spiralförmigen Bewegung. Das Zentrum der Spirale ist die optimale Position des Textobjektes bzw die Koordinaten an denen der Text platziert werden würde, sofern keine Kollisionen auftreten. Der Text wird sodann um einen bestimmten Abstand nach unten verschoben, welcher in der Datei *settings.yaml* festgelegt ist, siehe Kapitel 2.2.

In der Folge wird der Text kontinuierlich um den gleichen Abstand verschoben, wobei die Richtung der nächsten Verschiebung aus der Anzahl der bereits erfolgten Verschiebungen ermittelt wird. Zu diesem Zweck wird gespeichert welche Nummer die derzeitige Position des Textes hat bzw. wie oft schon eine Verschiebung durchgeführt wurde und in welcher Runde des Verschiebeprozesses sich der Text befindet. Eine Runde bedeutet dabei, dass der Text einmal um das Zentrum der Spirale herumgewandert ist. Zum Beispiel beginnt eine neue Runde wenn der Text wie in Abbildung 2.4 von Position 8 auf Position 9 verschoben wird.

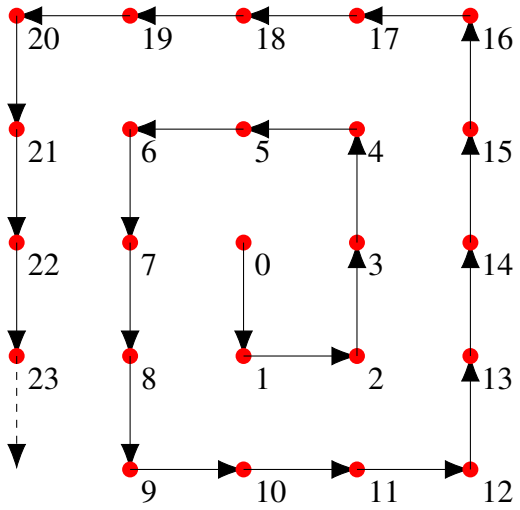
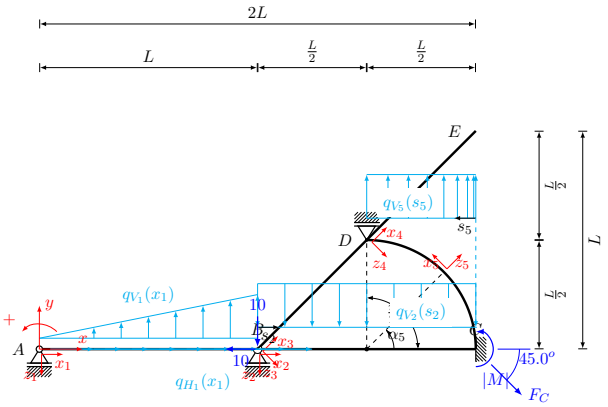



Abbildung 2.4: Darstellung Ablauf Verschiebealgorithmus

# 3. BEISPIEL

TU Graz
Institut für Baumechanik

## Hausübung Systemberechnung



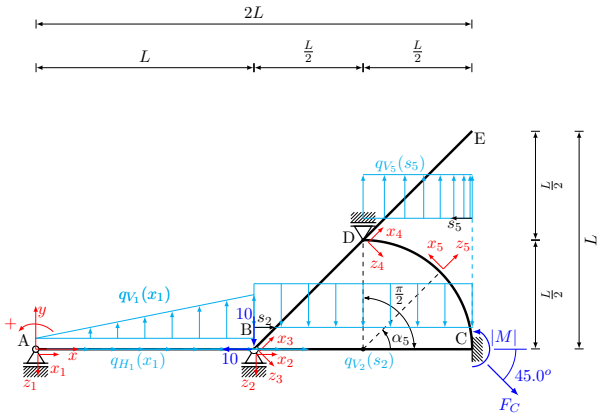

Die Belastungsfunktionen lauten:

$$F_C = \sqrt{2}F$$
$$q_{H_1}(x_1) = q_0$$
$$q_{V_1}(x_1) = \frac{q_0 x_1}{L}$$
$$q_{V_2}(s_2) = q_0$$
$$q_{V_5}(s_5) = 10$$

Abbildung 3.1: Beispielsystem ohne automatische Textpositionierung und Kollisionskontrolle

TU Graz
Institut für Baumechanik

# Hausübung Systemberechnung



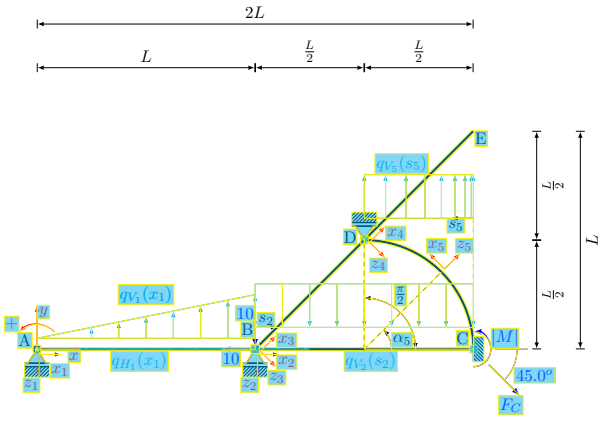

Die Belastungsfunktionen lauten:

$$F_C = \sqrt{2}F$$
$$q_{H_1}(x_1) = q_0$$
$$q_{V_1}(x_1) = \frac{q_0 x_1}{L}$$
$$q_{V_2}(s_2) = q_0$$
$$q_{V_5}(s_5) = 10$$

Abbildung 3.2: Beispielsystem mit automatischer Textpositionierung und Kollisionskontrolle

TU Graz
Institut für Baumechanik

# Hausübung Systemberechnung




Die Belastungsfunktionen lauten:

$$F_C = \sqrt{2}F$$
$$q_{H_1}(x_1) = q_0$$
$$q_{V_1}(x_1) = \frac{q_0 x_1}{L}$$
$$q_{V_2}(s_2) = q_0$$
$$q_{V_5}(s_5) = 10$$

Abbildung 3.3: Darstellung der Geometrie für die Kollisionskontrolle

Wie in Abbildung 3.1 erkennbar, kommt es bei Verwendung des ursprünglichen Programms zu Überlappung von Texten mit anderen Objekten oder untereinander, wodurch die Lesbarkeit beeinträchtigt wird. Anders stellt sich die Situation dar, wenn das weiterentwickelte und um eine Methodik zur automatisierten Positionsbestimmung von Beschriftungen ergänzten Programms verwendet wird, wie in Abbildung 3.2 ersichtlich werden dadurch Überschneidungen von Texten mit anderen Teilen der Grafik erfolgreich vermieden.

In der Abbildung 3.3 ist dargestellt, wie die Bestandteile der Grafik durch Geometrieobjekte im Programm repräsentiert werden. Die gelb markierten Linien repräsentieren Geometrieobjekte, welche sich nicht mit Texten überlappen dürfen. Die von Polygonen begrenzte Fläche ist in der vorliegenden Abbildung in Hellblau eingefärbt. In diesen Bereichen ist die Platzierung von Beschriftungen nicht zulässig. Auch bei allen Textobjekten ist das, in diesem Fall rechteckige Begrenzungspolygon, welches der Bounding Box des Textes entspricht, ersichtlich.

Das dargestellte System umfasst insgesamt 106 Geometrieobjekte, von denen 75 dem Standardtyp und 31 dem Texttyp zuzuordnen sind.

Die Laufzeit des gesamten Programms beträgt für das gezeigte Beispiel wenige Sekunden, in der Regel eine einstellige Anzahl. Beim gezeigten Beispiel entfällt bei einer Gesamtlaufzeit des Programms von rund fünf Sekunden etwa eine Sekunde und damit rund 20% der aufgewendeten Zeit auf die korrekte Platzierung der Beschriftung, welche unter anderem einen Vierschiebealgorithmus und Kollisionskontrollen beinhaltet. Im Laufe dessen wird beim gezeigten Beispiel in Summe 1899 mal ein Textobjekt verschoben, nachdem eine Kollisionskontrolle nicht das gewünschte Ergebnis geliefert hat und damit eine Positionsveränderung notwendig war.

# 4. DISKUSSION

Wie in Kapitel 3 ersichtlich, konnte die grafische Darstellung in Bezug auf die Lesbarkeit der Beschriftung durch die Implementierung einer automatisierten Texpositionierung verbessert werden. Durch die Nachbildung der Grafik mittels einzelner Geometrieobjekte innerhalb des Python-Programms und die Durchführung von Kollisionskontrollen mit diesen, konnten erfolgreich alle Überschneidungen von Texten mit anderen Bestandteilen der Grafik vermieden werden.

Die Möglichkeit, die für die Kollisionskontrollen verwendeten Geometriedaten über der eigentlichen Darstellung des statischen Systems zu visualisieren, wie in Abbildung 3.3 dargestellt, erlaubt eine schnelle Kontrolle und grafische Verifikation der korrekten Arbeitsweise des Programms. Die vorliegende Funktionalität erweist sich deshalb als vorteilhaft bei der Fehlerbehebung, der Weiterentwicklung sowie der Ergänzung neuer Funktionalitäten.

Es sei darauf hingewiesen, dass der Wert für den Parameter *movement_distance* in der Settings-Datei mit Bedacht gewählt werden sollte, da zu kleine Werte zu erhöhten Rechenzeiten führen.

Sollte das Programm in Zukunft auch für wesentlich komplexere mechanische System eingesetzt werden, so kann eine Änderung mancher Datenstrukturen und Abläufe notwendig werden, um die Zeitkomplexität und damit die Laufzeiten zu optimieren, weitere Erläuterungen folgen in Kapitel 5.2.

Für das absehbare Einsatzgebiet wird jedoch, bei passenden Einstellungen in der Settings-Datei, mit akzeptablen Rechenzeiten eine deutlich verbesserte Platzierung der Beschriftungen im Sinne einer besseren Lesbarkeit erreicht.

# 5. FAZIT

## 5.1 Zusammenfasssung

Es wurde erreicht, dass mit dem um eine Funktionalität zur automatischen Platzierung von Beschriftungen ergänzten Programm nun alle Beschriftungen so platziert werden, dass sie sich mit keinen anderen Beschriftungen oder sonstigen Bestandteilen der Grafik überlappen. Dadurch kann nun sichergestellt werden, dass alle Texte gut lesbar sind.

Bei Beispielen, die von der Komplexität bzw. Belegungsdichte der Grafik dem angedachten Verwendungszweck des Programms entsprechen, befinden sich die Beschriftungen auch in ausreichender Nähe zu jenen Elementen der Grafik, welche sie beschreiben. Durch diese räumliche Nähe kann sichergestellt werden, dass die Beschriftungen beim betrachten der Grafik den richtigen Elementen bzw. Bestandteilen der Grafik zugeordnet werden können.

## 5.2 Weiterentwicklungsmöglichkeiten und Optimierungsmöglichkeiten

Das Programm weist derzeit einige geometrische Limitierungen auf. So werden etwa Kreisbögen mit einem Winkel von mehr als $\pi$ sowohl vom bestehenden Programm, als auch von dieser Weiterentwicklung, nicht korrekt dargestellt. Auch sind die Darstellungs- und Verarbeitungsmöglichkeiten des Programms auf mechanische Systeme beschränkt, welche sich aus geraden Stäben und Kreisbögen zusammensetzen. Eine Darstellung von Stäben, welche beispielsweise einer Polynomfunktion folgen, oder von Bögen mit einer nicht konstanten Krümmung ist nicht möglich.

Ein anderer Bereich, in dem Potential zur Weiterentwicklung vorhanden ist, ist die Methodik zur Kollisionskontrolle. Die Laufzeit kann bei Grafiken mit einer größeren Anzahl von Objekten aufgrund der ungünstigen Zeitkomplexität der verwendeten Methodik signifikant ansteigen. Für den absehbaren Einsatzzweck stellt dies jedoch kein Hindernis dar, da nicht mit einer hohen dreistelligen Anzahl an Geometrieobjekten oder gar noch größeren Anzahlen zu rechnen ist.

Sollte es zukünftig dennoch erforderlich sein, mit dem Programm wesentlich größere und komplexere mechanische Systeme zu erzeugen bzw. darzustellen, so ist eine Anpassung des Algorithmus zur Kollisionskontrolle zu empfehlen. Eine Möglichkeit, die Zeitkomplexität zu optimieren, besteht in der Indizierung der Boundingboxen der Geometrieobjekte

in baumartigen Datenstrukturen. In diesem Zusammenhang sind verschiedene Baumstrukturen denkbar, wobei sich der R-Baum besonders anbietet.

Ein R-Baum ist eine hierarchische Datenstruktur, die vom B-Baum abgeleitet und für die effiziente Ausführung von Schnittpunktabfragen konzipiert ist. R-Bäume speichern eine Sammlung von Rechtecken, die sich im Laufe der Zeit durch Einfügungen und Löschungen verändern können. Beliebige geometrische Objekte werden behandelt, indem jedes Objekt durch sein minimales begrenzendes Rechteck dargestellt wird, d. h. das kleinste Rechteck, das das Objekt umschließt. [2]

Eine einfache Möglichkeit wäre die Verwendung der Python-rtree-Bibliothek oder der shapely strtree-Implementierung, welche den in [2] beschriebenen Short-Tile-Recursive-Algorithmus nutzt, um einen Baum zu erzeugen.

Eine weitere Möglichkeit zur Optimierung bestünde in der Parallelisierung der Platzierung von Textobjekten, die eine gleich hohe Priorität bei der Positionierung aufweisen und ausreichend weit voneinander entfernt sind.

# ABBILDUNGSVERZEICHNIS

# LITERATURVERZEICHNIS

[1] Michael H. Gfrerer, Benjamin Marussig, Katharina Maitz, and Mia M. Bangerl. Teaching mechanics with individual exercise assignments and automated correction. *PAMM*, 23(3):e202300168, 2023.

[2] Scott T Leutenegger, Mario A Lopez, and Jeffrey Edgington. Str: A simple and efficient algorithm for r-tree packing. In *Proceedings 13th international conference on data engineering*, pages 497–506. IEEE, 1997.