Grafisch-geometrische Algorithmen

Jan Keuchel

4. November 2025

Inhaltsverzeichnis

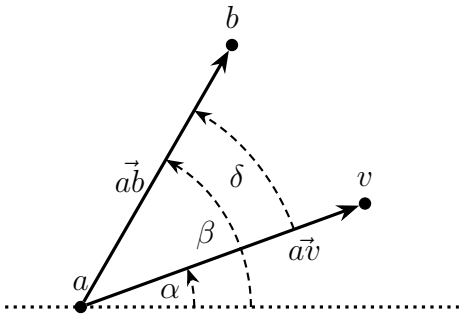
1 Grundlagen

1.1 Orientierung dreier Punkte

Eine grundlegene und oft genutzte Operation, ist zu prüfen, ob sich ein Punkt rechts, links oder auf einer Strecke befindet. Ziel ist es also eine Funktion RightOf zu definieren, welche 3 Punkte der Form $p = (p_x, p_y) \in \mathbb{R}^2$ nimmt und diese auf $\mathbb{R}$ abbildet. Also $\text{RightOf} : \mathbb{R}^6 \longrightarrow \mathbb{R}$, mit

$$\text{RightOf}(a,b,v) \begin{cases} < 0 & \text{falls } v \text{ links der gerichteten Gerade } \overline{ab} \text{ liegt.} \\ = 0 & \text{falls } v \text{ auf der gerichteten Gerade } \overline{ab} \text{ liegt.} \\ > 0 & \text{falls } v \text{ rechts der gerichteten Gerade } \overline{ab} \text{ liegt.} \end{cases}$$

. Gegeben drei Punkte $a, b, v \in \mathbb{R}^2$ hat man folgende Situation:



Man stellt fest, dass v genau dann rechts der gerichteten Gerade $\overline{ab}$ liegt, wenn $\sin\varphi > 0$, links davon, wenn $\sin\varphi < 0$ und darauf liegt, falls $\sin\varphi = 0$ gilt. Somit kann man

$$\text{RightOf}(a,b,v) := \sin\varphi = \sin(\beta - \alpha) = \sin\beta\cos\alpha - \cos\beta\sin\alpha$$

definieren. Da trigonometrische Funktionen aufgrund von Rechenleistung unerwünscht sind, kann RightOf mit

$$\begin{aligned} \sin\beta &= b.y - a.y \\ \cos\alpha &= v.x - a.x \\ \cos\beta &= b.x - a.x \\ \sin\alpha &= v.y - a.y \end{aligned}$$

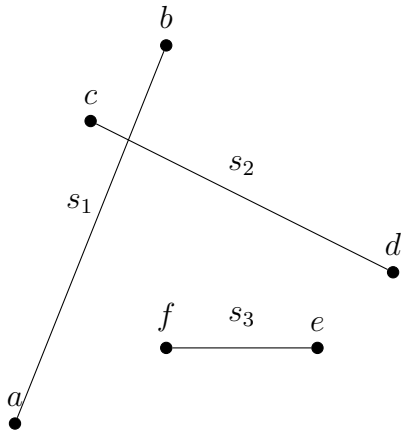
wie folgt dargestellt werden:

$$\text{RightOf}(a,b,v) = (b.y - a.y)(v.x - a.x) - (b.x - a.x)(v.y - a.y)$$

.

1.2 Schnitt zweier Segmente

Eine weitere, oft genutzt Funktion ist der Test, ob sich zwei Segmente schnieden oder nicht. Ziel ist es nicht, zu berechnen, wo ein Schnittpunkt liegt, sondern ob er existiert oder nicht. Wir betrachten also mehrere Segmente s_1, s_2, s_3 der Form $s_i = (p_{i1}, p_{i2})$, mit $p \in \mathbb{R}^2$:



Dabei stellt man fest, dass sich zwei Segmente s_i, s_j in genau einem Fall schneiden: Wenn sowohl die Punkte s_{j1}, s_{j2} auf unterschiedlichen Seiten der Geraden $\overline{s_{i1}s_{i2}}$, und die Punkte s_{i1}, s_{i2} auf unterschiedlichen Seiten der Geraden $\overline{s_{j1}s_{j2}}$ liegen. Wird der Fall, dass ein Punkt p auf einer Geraden $\overline{ab}$ liegt, ebenfalls als Schnittpunkt angesehen, so reicht es aus, dass die beiden Punkt nicht auf der gleichen Seite liegen.

Die Funktion CrossingExists kann also mit Hilfe der vorig beschriebenen Funktion RightOf definiert werden (Hier wird ein Punkt, der auf einer Geraden liegt, bereits als Schnittpunkt angesehen):

$$\textsc{CrossingExists}(a,b,c,d) = \begin{pmatrix}\big(\textsc{RightOf}(a,b,c)\cdot\textsc{RightOf}(a,b,d)\big)\leq 0\end{pmatrix}\ \textbf{and}\\ \begin{pmatrix}\big(\textsc{RightOf}(c,d,a)\cdot\textsc{RightOf}(c,d,b)\big)\leq 0\end{pmatrix}$$

2 Konvexe Hülle

Eine Untermenge C einer Ebene heißt genau dann konvex, wenn $\forall p, q \in C : \overline{pq} \subseteq C$.

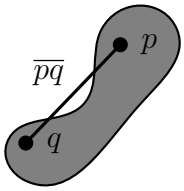


Abbildung 1: Nicht konvexe Menge

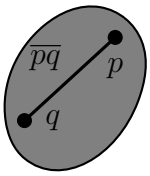


Abbildung 2: Konvexe Menge

Ein Polygon $\mathcal{P}$ ist eine Fläche, die durch eine Folge an $k > 2$ Punkten $\langle p_1, p_2, \ldots, p_k\rangle$ bestimmt wird.

2.1 Definition

Zu einer endlichen Menge P aus n Punkten, ist die konvexe Hülle $\mathcal{CH}(P)$ ein konvexes Polygon, welches aus den Punkten von P besteht und alle Punkte uas P einschließt. $\mathcal{CH}(P)$ wird als eine von einem zufälligen Punkt ausgehende, im Uhrzeigersinn sortierte Liste aller Punkte $p_i \in \mathcal{CH}(P)$ dargestellt.

Bei Abbildung 3 ist $P = \{p_0, p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9\}$ und $\mathcal{CH}(P) = \langle p_0, p_1, p_3, p_7, p_6, p_2\rangle$

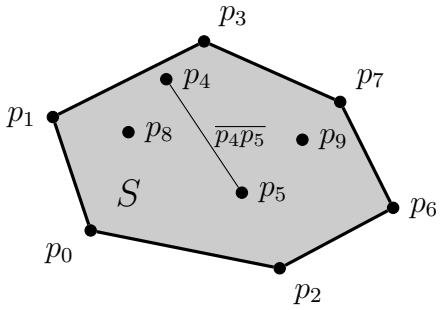


Abbildung 3: Konvexe Hülle

2.2 Algorithmen

Für die Berechnung der Konvexen Hülle werden nun drei Algorithmen betrachtet.

2.2.1 Brute Force

Bei der Betrachtung der Kanten e_i von $\mathcal{CH}(P)$ fällt auf, dass diese eine gemeinsame Eigenschaft haben: All anderen Punkte $p \in P$ befinden sich – da $\mathcal{CH}(P)$ im Uhrzeigersinn sortiert ist – auf der rechten Seite von e_i. Die Kanten E von $\mathcal{CH}(P)$ können also auch wie folgt beschrieben werden:

$$E = \{(p,q) \in P \times P : p \neq q \mid \forall r \in P : r \neq p \wedge r \neq q \wedge \textsc{RightOf}(p,q,r) \geq 0\}$$

. Aus E müsste schließlich noch eine sortierte Folge gebildet werden.

2.2.2 Monotone Ketten

2.2.3 Jarvis' March

2.3 Beweise

Theorem 1. *Der Schnitt zweier konvexer Polygone ist ein konvexes Polygon.*

Beweis. Seien $\mathcal{A}, \mathcal{B}$ konvexe Polygone. Mit $(s,t \in \mathcal{A} \cap \mathcal{B})$ gilt $(\overline{st} \subseteq \mathcal{A} \wedge \overline{st} \subseteq \mathcal{B}) \Leftrightarrow (\overline{st} \subseteq \mathcal{A} \cap \mathcal{B})$. □

Theorem 2. *Unter allen Polygonen, welche eine Punktemenge P enthalten, ist dasjenige mit dem kleinsten Umfang konvex.*

Beweis. Sei $\mathcal{P} = \langle p_0, p_1, \ldots, p_{n-1} \rangle$ das Polygon mit dem kleinsten Umfang, welches P enthält. Ferner sei $p_i^+ = p_{(i+1) \bmod n}$ der Nachfolger von p_i und $U(\mathcal{A})$ der Umfang eines Polygons $\mathcal{A}$. Für drei Punkte A, B, C ist $\overline{AC} \leq \overline{ABC}$. Somit gilt für $a_1, \ldots a_n$, dass $\overline{a_1 a_n} \leq \overline{a_1 \ldots a_n}$. Sei $\mathcal{P}$ nicht konvex, dann

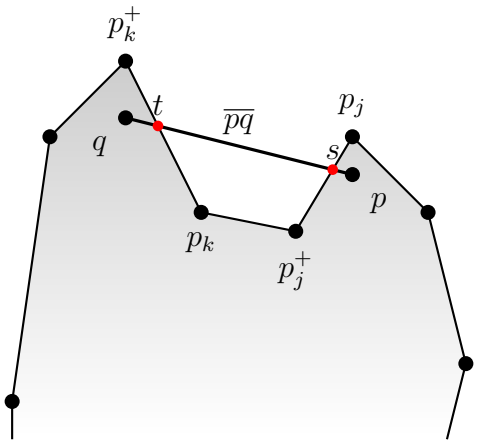
$$\exists p, q \in \mathcal{P} : \overline{pq} \not\subseteq \mathcal{P} \Rightarrow \exists j, k < n \wedge j < k : \exists s, t \in \left(\overline{p_j p_j^+} \cup \overline{p_k p_k^+}\right) \cap \overline{pq}$$

. Mit $\mathcal{P}' = \langle p_1, \ldots, p_j, s, t, p_k^+, \ldots p_n \rangle$ ist $U(\mathcal{P}') < U(\mathcal{P})$.

□

Theorem 3. *Jede konvexe Menge, die eine Punktemgne P beinhaltet, ist eine Obermenge des kleinsten Polygons $\mathcal{P}$, welches P beinhaltet.*

Beweis. Sei $\mathcal{A} \supseteq P$ eine konvexe Menge. Nach Definition ist $\mathcal{P} \supseteq P$ und $\mathcal{A} \supset \mathcal{P}$. □



3 Schnitt von Linien-Segmenten

4 Mengenoperationen

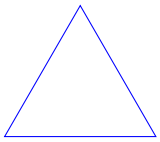
5 Triangulation

6 TikZ Test

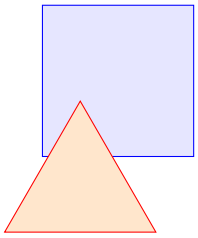
6.1 Lines, filling opacity and movement

6.1.1 Draw a simple line

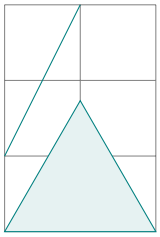
6.1.2 Draw a cycle between points



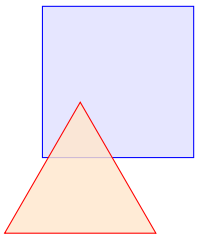
6.1.3 Fill a polygon



6.1.4 – – operator



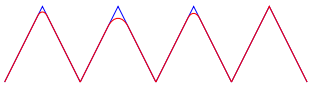
6.1.5 Opacity of the filling



6.2 Styles

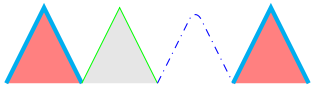
6.2.1 Line thickness

6.2.2 Rounded corners



6.2.3 Line styles

6.2.4 Creating styles

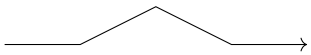


6.2.5 Change style of every line

6.3 Arrows

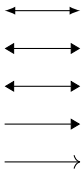
6.3.1 Basics

6.3.2 Arrows only appear at the end



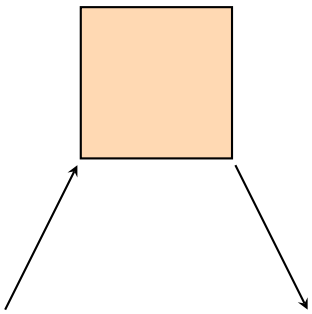
6.3.3 Arrows don't appear in a cycle

6.3.4 Arrow heads



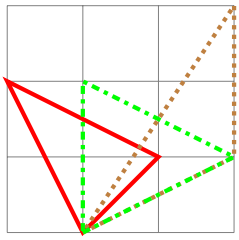
6.3.5 Style of arrow heads

6.3.6 Shorten key: Pointing at something



6.4 Relative Coordinates

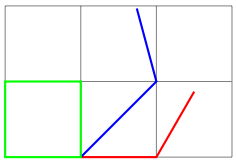
6.4.1 + and ++



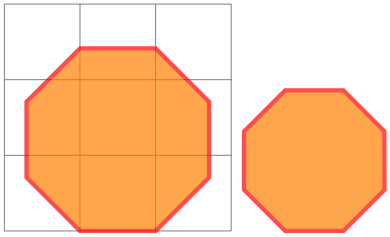
6.4.2 Two diamonds Explicit vs. Implicit

Explicit:  Using +:  Using ++: 

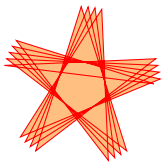
6.4.3 Relative turns



6.4.4 Stop Sign

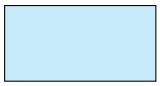


6.4.5 Testing turning and walking



6.5 Rectangles

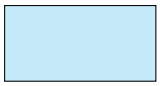
6.5.1 Explicit



6.5.2 Rectangle path

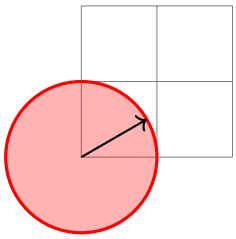
6.5.3 Rectangle path implicit

6.5.4 Rectangle path implicit default start

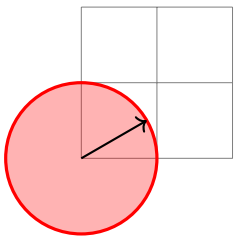


6.6 Circles and Ellipses

6.6.1 Circle I

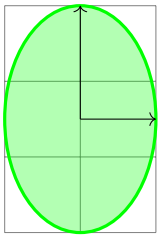


6.6.2 Circle II

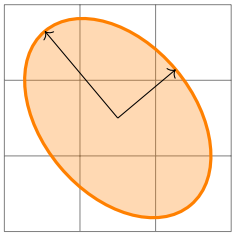


6.6.3 Circle III

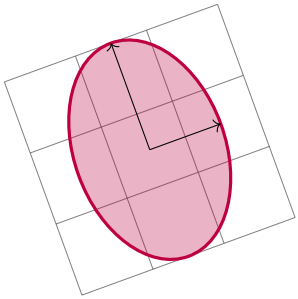
6.6.4 Ellipses I



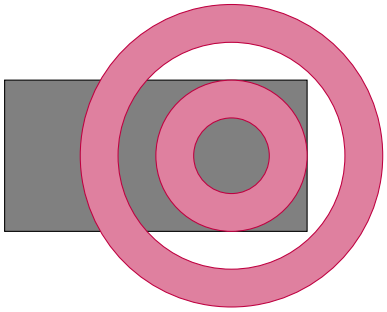
6.6.5 Ellipses II: rotation



6.6.6 Testing: "global"rotation

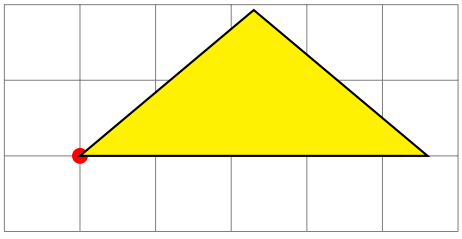


6.6.7 Transparent circle: even odd rule

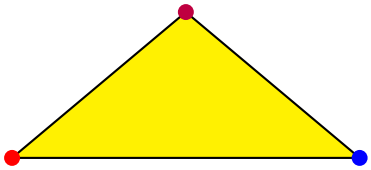


6.7 Naming Coordinates

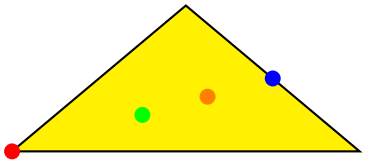
6.7.1 Names



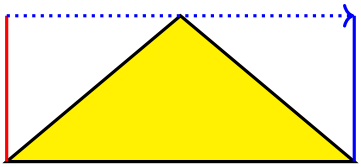
6.7.2 Naming coordinates in the path command



6.7.3 Naming coordinates along the path

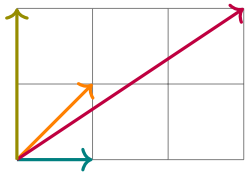


6.7.4 Mixing coordinates: dash-bar and bar-dash

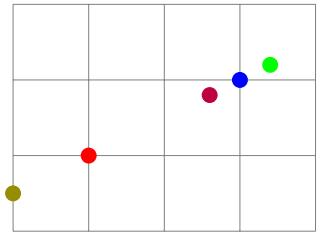


6.8 Coordinate Calculations

6.8.1 Linear Combinations

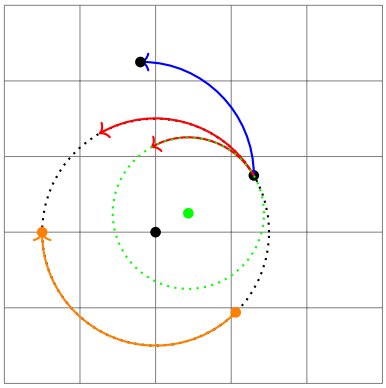


6.8.2 Linear interpolation

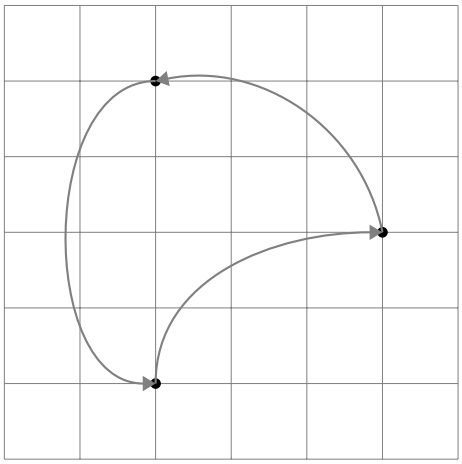


6.9 Curved Paths

6.9.1 `arc` command

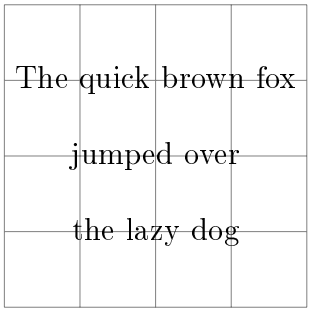


6.9.2 `topath` command

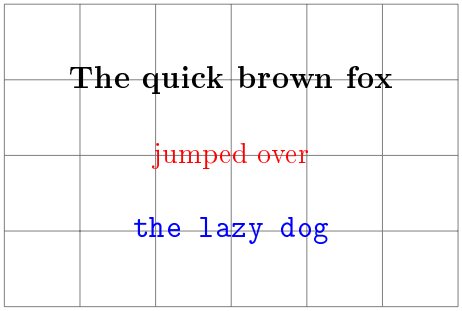


6.10 Node

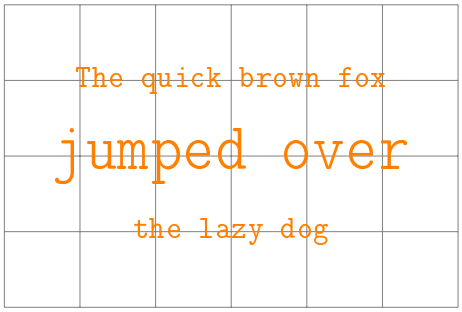
6.10.1 Writing text



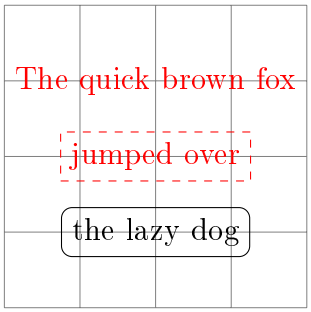
6.10.2 Font and color



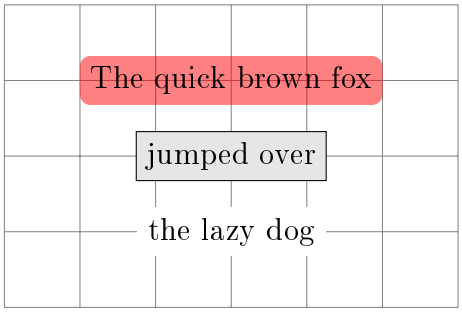
6.10.3 Scope



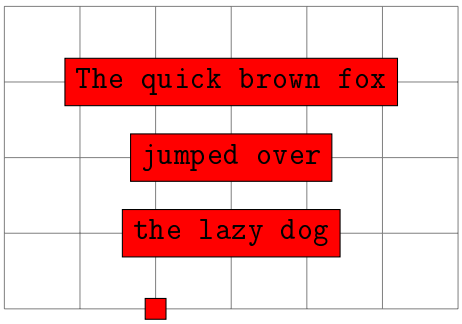
6.10.4 Outline



6.10.5 Background

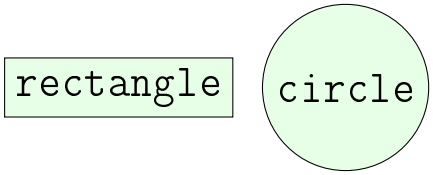


6.10.6 Default styles

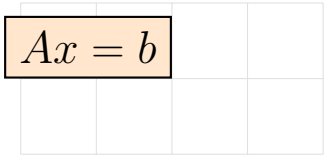


6.11 Positioning of nodes and anchors

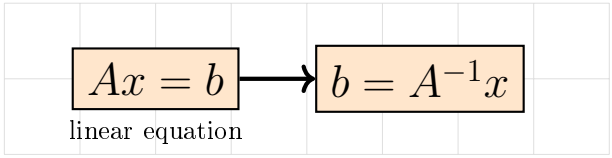
6.11.1 Shapes



6.11.2 Compass Point Node Anchors



6.11.3 Node anchors as reference



6.12 Positioning package

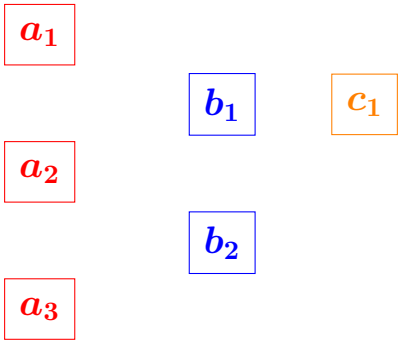
6.12.1 Some nodes...

a_1

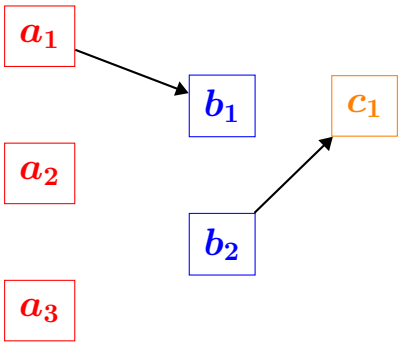
a_2 b_1 c_1

a_3 b_2

6.12.2 Custom distances



6.12.3 Edges and nodes



6.13 Nils BSc Diagramm

6.13.1 Block Diagramm

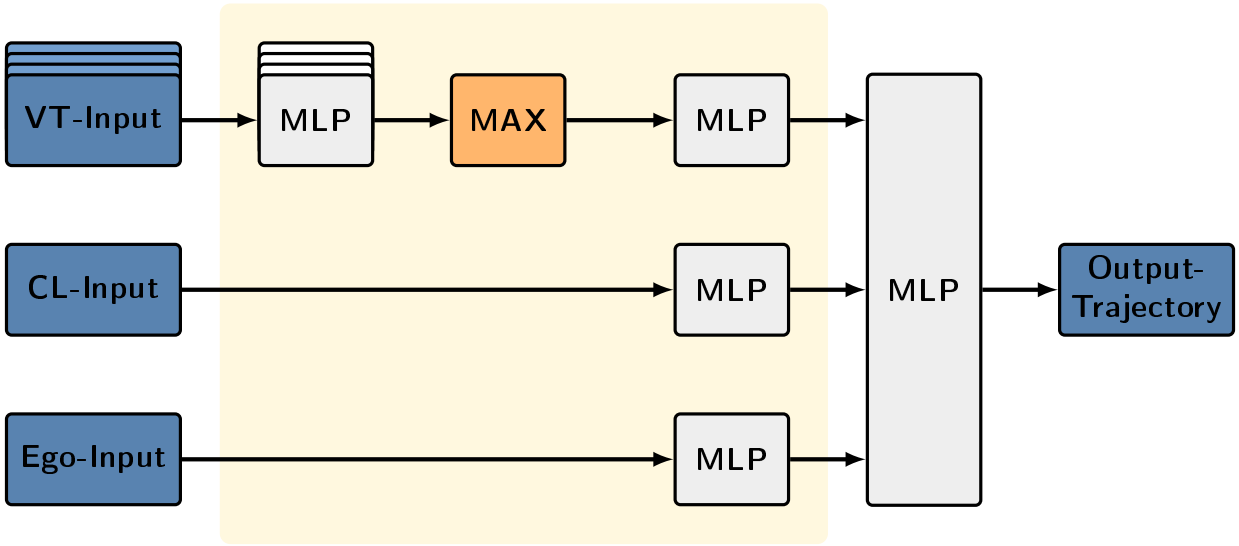
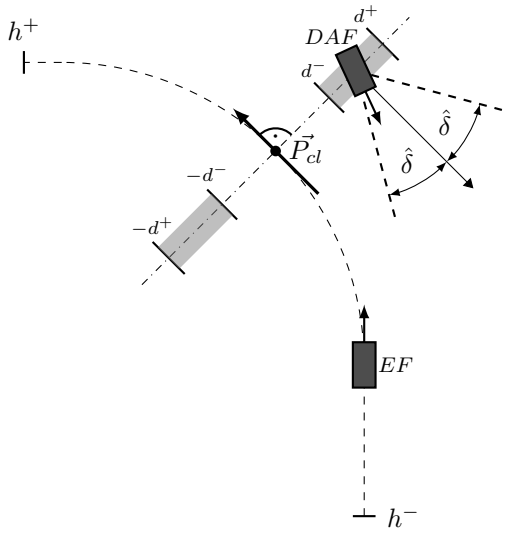


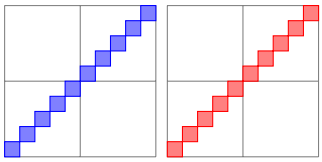
Abbildung 4: Grundlegender Aufbau des Modells und der Encoder. Die Input- verarbeitung ist beige hinterlegt. Die Encoder für Ego-, CL- und VT- Inputs bestehen jeweils aus einem MLP. Die einzelnen VT-Inputs durch- laufen vorher zusätzlich ein MLP und werden in einem Maxpool-Layer zusammengeführt. Alle Inputs werden in das eigentliche Modell, ein MLP, gegeben. Dieses liefert die Output-Trajektorie.

6.13.2 Auto in der Kurve



6.14 Loops

6.14.1 Basics



6.14.2 Nested Loops