

Kapitel 03 - Vertiefung

Testen von `find_intersection()`

Ein visueller Leitfaden für mehrdimensionales Testdesign

Software Engineering | SoSe 2026

Hochschule Aalen

Warum dies ein eigenes Deck verdient

Dies ist deine erste Begegnung mit:

-  Mehrdimensionalem Äquivalenzklassentesten
-  Kombinatorischer Komplexität (600 potentielle Tests!)
-  Strategischer Testreduktion mittels Domänenwissen
-  Visuellem Verständnis geometrischer Testfälle

Ziel: Am Ende wirst du verstehen, wie man systematisch Tests für komplexe Funktionen entwirft!

 **Vollständige Details:** Siehe den [Vorlesungsnotizen-Abschnitt zu `find\_intersection\(\)`](#)

Die Funktion, die wir testen

Aus deinem Road Profile Viewer Projekt:

```
def find_intersection(
    x_road: np.ndarray,      # Straßenprofil x-Koordinaten
    y_road: np.ndarray,      # Straßenprofil y-Koordinaten
    angle_degrees: float,    # Strahlrichtung (-90° bis 90°)
    camera_x: float = 0,     # Kamera x-Position
    camera_y: float = 1.5    # Kamera y-Position
) -> tuple[float, float] | None:
    """
    Findet wo ein Strahl von der Kamera das Straßenprofil schneidet.
    Gibt (x, y) des Schnittpunkts zurück, oder None falls kein Schnitt.
    """
    # ... 70 Zeilen geometrischer Logik ...
```

Die Herausforderung: Fünf Dimensionen der Komplexität

Wir haben 5 unabhängige Dimensionen identifiziert:

1. **Array-Struktur** (5 Klassen): Leer, ein Punkt, zwei Punkte, drei Punkte, viele Punkte
2. **Winkel** (5 Klassen): Abwärts, horizontal, aufwärts, vertikal nach unten, vertikal nach oben
3. **Kameraposition** (6 Klassen): Vor Straße, auf Straße, nach Straße, über, auf Höhe, unter
4. **Ergebnis** (4 Klassen): Kein Schnitt, erstes Segment, mittleres Segment, letztes Segment
5. **Straßenform** (3 Klassen): Flach, geneigt, gekrümmt

Mathematik: $5 \times 5 \times 6 \times 4 \times 3 = 1800$ potentielle Kombinationen! 🤖

Unsere Strategie: 1800 → 22 Tests

Wie wir die Testzahl reduziert haben:

1. **Jede Dimension abdecken** mit 2-4 Repräsentanten
2. **Domänenwissen nutzen** um unmögliche/redundante Kombinationen zu eliminieren
3. **Fokus auf distinkte Verhaltensweisen** statt auf alle Parameterkombinationen
4. **Kritische Grenzfälle hinzufügen** die nicht in ordentliche Kategorien passen

Ergebnis: 22 strategische Tests die die wesentlichen Äquivalenzklassen abdecken!

Lass uns jeden Test visuell ansehen... 🎨

Visualisierungen lesen

Legende für alle Diagramme:

-  **Blaue Linie mit Punkten** = Straßenprofil
-  **Roter Kreis** = Kameraposition
-  **Rote gestrichelte Linie** = Strahl von der Kamera (definiert durch Winkel)
-  **Grüner Kreis** = Schnittpunkt (falls gefunden)
-  **Grüne gepunktete Linie** = Distanz von Kamera zu Schnittpunkt

Hinweis: Alle Schnittpunkte sind **geometrisch berechnet**, nicht approximiert!

Kategorie 1: Strukturelle Tests

Dimension: Wie viele Punkte definieren die Straße?

Warum das wichtig ist:

- Leere Arrays → Fehlerbehandlung
- Ein Punkt → kein Liniensegment (ungültig)
- Zwei Punkte → minimale gültige Straße (1 Segment)
- Viele Punkte → typischer Fall

Tests 1-4 decken diese Dimension ab...

Test 1: Leere Arrays

Eingabe: `x_road = []`, `y_road = []`

Erwartet: `None` (keine Straße zu schneiden)

Was es testet: Fehlerbehandlung für ungültige Eingabestruktur

Test 2: Ein Punkt

Eingabe: `x_road = [5.0]` , `y_road = [2.0]`

Erwartet: `None` (ein Punkt definiert kein Liniensegment)

Was es testet: Minimale Strukturvalidierung (mindestens 2 Punkte nötig)

Test 3: Zwei Punkte - Minimal gültige Straße

Eingabe: `x_road = [0, 10]` , `y_road = [0, 2]` , Winkel = -10°

Erwartet: `None` (Strahl verfehlt die kurze Straße)

Was es testet: Minimal gültige Struktur (1 Segment), Strahlgeometrie

Test 4: Viele Punkte (50 Punkte)

Eingabe: 50 Punkte die eine lange geneigte Straße erzeugen

Erwartet: Schnittpunkt gefunden bei ($x \approx 13.9$, $y \approx 1.4$)

Was es testet: Komplexe Straße mit vielen Segmenten, typischer Anwendungsfall

Kategorie 2: Winkel-Äquivalenzklassen

Dimension: Richtung des Strahls

Warum das wichtig ist:

- Verschiedene Winkel trainieren verschiedene Trigonometrie
- Vertikale Strahlen ($\pm 90^\circ$) haben spezielle Behandlung (keine Steigung)
- Horizontale Strahlen (0°) sind Grenzfälle
- Richtung beeinflusst welche Segmente getroffen werden können

Tests 5-9 decken diese Dimension ab...

Test 5: Abwärtswinkel (-10°)

Eingabe: Winkel = -10° (nach unten schauend), Kamera über Straße

Erwartet: Schnittpunkt gefunden bei ($x \approx 33.1$, $y \approx 4.1$)

Was es testet: Typischer Abwärtsstrahl, positive Steigungsmathematik

Test 6: Horizontaler Strahl (0°)

Eingabe: Winkel = 0° (horizontal), Kamera auf Straßenhöhe

Erwartet: Schnittpunkt bei (x=0, y=2)

Was es testet: Nullsteigungsfall, Tangente zu flacher Straße

Test 7: Aufwärtswinkel (45°)

Eingabe: Winkel = 45° (nach oben schauend), Kamera auf Straßenhöhe

Erwartet: Schnittpunkt bei (x=1, y=1)

Was es testet: Positiver Winkel, negative Steigungsmathematik

Test 8: Vertikal abwärts Strahl (-90°)

Eingabe: Winkel = -90° (geradeaus nach unten), Kamera über Straße

Erwartet: None (vertikaler Strahl, kein Schnitt erkannt)

Was es testet: Sonderfall - undefinierte Steigung, vertikale Strahlbehandlung

Test 9: Vertikal aufwärts Strahl (90°)

Eingabe: Winkel = 90° (geradeaus nach oben), Kamera am Boden

Erwartet: None (vertikaler Strahl, kein Schnitt)

Was es testet: Weiterer vertikaler Grenzfall, Symmetrie mit -90°

Kategorie 3: Positions-Äquivalenzklassen

Dimension: Wo ist die Kamera relativ zur Straße?

Warum das wichtig ist:

- Position beeinflusst welche Segmente "vor" der Kamera sind
- Randpositionen testen Grenzlogik
- Höhe beeinflusst Schnittwinkel
- Testet camera_x und camera_y Parameter

Tests 10-15 decken diese Dimension ab...

Test 10: Kamera vor Straßenbeginn

Eingabe: camera_x = 0 (vor Straßenbeginn bei x=10)

Erwartet: Schnittpunkt bei (x≈41.5, y≈3.8)

Was es testet: Kamera sieht volle Straße voraus, trifft mittleres Segment

Test 11: Kamera auf Straße (x=15)

Eingabe: camera_x = 15 (zwischen Straßenpunkten)

Erwartet: Schnittpunkt bei (x≈41.8, y≈4.2)

Was es testet: Kamera innerhalb der Straßengrenzen positioniert

Test 12: Kamera nach Straßenende

Eingabe: camera_x = 30 (nach Straßenende bei x=20)

Erwartet: None (Strahl schaut vorwärts, Straße ist hinten)

Was es testet: Kamera nach allen Segmenten, kein Vorwärtsschnitt

Test 13: Kamera über Straße - Nach unten schauend

Eingabe: camera_y = 20 (hoch über Straße), Winkel = -10°

Erwartet: None (Strahl verfehlt niedrige Straße)

Was es testet: Vertikale Position beeinflusst Erreichbarkeit

Test 14: Kamera auf Straßenhöhe - Tangente

Eingabe: camera_y = 2.0 (gleiche Höhe wie flache Straße), Winkel = 0°

Erwartet: Schnittpunkt bei (x=0, y=2) - Strahl berührt Straße sofort

Was es testet: Tangentenfall, Strahl entlang Oberfläche

Test 15: Kamera unter Straße - Nach oben schauend

Eingabe: camera_y = 0 (unter Straße), Winkel = 10° (aufwärts)

Erwartet: Schnittpunkt bei ($x \approx 28.3$, $y=5$)

Was es testet: Kamera unter Oberfläche nach oben schauend um Straße zu treffen

Kategorie 4: Ergebnis-basierte Tests

Dimension: Was gibt die Funktion zurück?

Warum das wichtig ist:

- Verschiedene Ergebnisse trainieren verschiedene Code-Pfade
- "Wo" Schnitt erfolgt beeinflusst Rückgabewert-Berechnung
- Kein Schnitt ist ein gültiges zu testendes Ergebnis

Tests 16-19 decken diese Dimension ab...

Test 16: Schnitt am ersten Segment

Eingabe: camera_x = -5 (vor Straße), steiler Winkel = -30°

Erwartet: Schnittpunkt bei $(x \approx 8.2, y \approx 3.6)$ - trifft erstes Segment

Was es testet: Strahl trifft das frühestmögliche Segment

Test 17: Schnitt am mittleren Segment

Eingabe: Kamera bei $x=7$, sanfter Winkel = -5°

Erwartet: Schnittpunkt bei $(x \approx 13, y \approx 2.5)$ - Mitte der Straße

Was es testet: Strahl passiert erste Segmente, trifft mittleres

Test 18: Schnitt am letzten Segment

Eingabe: Kamera bei $x=15$, flacher Winkel = -2°

Erwartet: Schnittpunkt bei $(x \approx 27.5, y \approx 0.55)$ - letztes Segment

Was es testet: Strahl erreicht fernes Ende der Straße

Test 19: Kein Schnitt - Strahl verfehlt Straße

Eingabe: Kamera am Ursprung, Aufwärtswinkel = 45° , Straße weit voraus

Erwartet: None (Strahl schießt aufwärts, trifft Straße nie)

Was es testet: Strahltrajektorie schneidet Straßengeometrie nicht

Kategorie 5: Straßenform-Variationen

Dimension: Geometrische Eigenschaften der Straße

Warum das wichtig ist:

- Flache Straßen vereinfachen Geometrie (alle y-Werte gleich)
- Geneigte Straßen testen konsistenten Gradienten
- Gekrümmte Straßen testen variierende Steigungen

Tests 20-22 decken diese Dimension ab...

Test 20: Flache horizontale Straße - Tangentenfall

Eingabe: Alle y-Werte = 3.0 (perfekt flach), Winkel = 0°

Erwartet: Schnittpunkt bei (x=0, y=3) - Strahl entlang Oberfläche

Was es testet: Degenerierter Fall, horizontaler Strahl auf horizontaler Straße

Test 21: Konsistent geneigte Straße

Eingabe: $y = 0.2x$ (lineare Steigung), Winkel = -15°

Erwartet: Schnittpunkt bei $(x \approx 22.7, y \approx 4.5)$

Was es testet: Gleichmäßiger Gradient, alle Segmente haben gleiche Steigung

Test 22: Gekrümmte/wellige Straße (Sinusförmig)

Eingabe: $y = 2 \cdot \sin(x/5) + 3$ (wellige Straße)

Erwartet: Schnittpunkt bei ($x \approx 27.8$, $y \approx 4.9$)

Was es testet: Nicht-lineare Straße, variierende Steigungen, realistische Komplexität

Wichtige Erkenntnis: Gleiches Ergebnis, unterschiedliche Dimensionen

Beachte, dass Test 3 und Test 13 beide `None` zurückgeben (kein Schnitt).

Aber sie testen völlig unterschiedliche Dinge!

Test 3: Zwei-Punkte-Straße
(*strukturelle Dimension*)

- Kamera bei (0, 5), Strahl bei -10°
- Straße: (0, 0) $\rightarrow$ (10, 2)
- Strahl würde bei $x \approx 13.3$ schneiden, aber **Straße endet bei $x=10$**

Test 13: Kamera über Straße
(*Positions-Dimension*)

- Kamera bei (0, 20), Strahl bei -10°
- Straße: (0, 0) $\rightarrow$ (10, 1) $\rightarrow$ (20, 2)
- Strahl würde bei $x \approx 72.5$ schneiden, aber **Straße endet bei $x=20$**

Warum beide Tests notwendig sind

Test 3 validiert: "Kann die Funktion eine minimale Straßenstruktur handhaben (nur 1 Segment)?"

- Testet strukturelle Robustheit
- Stellt sicher, dass Algorithmus mit minimal gültiger Eingabe funktioniert

Test 13 validiert: "Kann die Funktion eine weit über der Straße positionierte Kamera handhaben?"

- Testet Positionslogik
- Stellt sicher, dass vertikale Verschiebung Schnittpunktmathematik nicht bricht

*Verschiedene Eingabe-**Dimensionen** abdecken, selbst wenn sie dasselbe*

Die vollständige Testsuite-Struktur

```
class TestFindIntersection:
    # Strukturell (4 Tests)
    def test_empty_arrays(self): ...
    def test_single_point(self): ...
    def test_two_point_road(self): ...
    def test_many_point_road(self): ...

    # Winkel (5 Tests)
    def test_downward_angle(self): ...
    def test_horizontal_angle(self): ...
    def test_upward_angle(self): ...
    def test_vertical_downward(self): ...
    def test_vertical_upward(self): ...
```

Die vollständige Testsuite-Struktur (Fortsetzung)

```
# Position (6 Tests)
def test_camera_before_road(self): ...
def test_camera_on_road(self): ...
def test_camera_after_road(self): ...
def test_camera_above_road(self): ...
def test_camera_at_road_level(self): ...
def test_camera_below_road(self): ...

# Ergebnis (4 Tests)
def test_first_segment(self): ...
def test_middle_segment(self): ...
def test_last_segment(self): ...
def test_no_intersection(self): ...
```

Die vollständige Testsuite-Struktur (Fortsetzung)

```
# Form (3 Tests)
def test_flat_road(self): ...
def test_sloped_road(self): ...
def test_curved_road(self): ...
```

```
# Gesamt: 22 strategische Tests decken 5 Dimensionen ab
# Statt: 1800 möglichen Kombinationen!
```

Was du gelernt hast

1. Mehrdimensionales

Äquivalenzklassentesten

- Echte Funktionen haben mehrere unabhängige Eingabedimensionen
- Jede Dimension kann einigermaßen unabhängig getestet werden

2. Strategische Testreduktion

- 1800 Kombinationen → 22 Tests mittels Domänenwissen
- Jede Dimension mit 2-4

3. Visuelles Verständnis

- Geometrische Probleme profitieren von visuellem Testdesign
- Illustrationen helfen Testkorrektheit zu verifizieren
- Macht abstrakte Konzepte konkret

Was du gelernt hast (Fortsetzung)

4. Gleiches Ergebnis $\neq$ gleicher Test

- Tests können identische Werte zurückgeben aber verschiedene Dimensionen abdecken
- Test 3 (strukturell) vs Test 13 (positional) geben beide `None` zurück
- Vollständige Abdeckung erfordert Testen aller Dimensionen

5. Domänenwissen ist unersetzbar

- Menschliches Verständnis von Geometrie ermöglicht intelligente Reduktion
- LLMs können nicht visualisieren oder priorisieren wie Domänenexperten
- Du lieferst Einsicht, LLMs liefern Implementierungshilfe

Zusammenfassung: Der Testprozess

Für komplexe mehrdimensionale Funktionen:

1. **Dimensionen identifizieren** - Was sind die unabhängigen Eingabefaktoren?
2. **Jede Dimension partitionieren** - Was sind die Äquivalenzklassen?
3. **Kombinationen berechnen** - Wie viele Tests theoretisch?
4. **Domänenwissen anwenden** - Welche Kombinationen sind unmöglich/redundant?
5. **Repräsentanten wählen** - Spezifische Werte für jede Klasse auswählen
6. **Falls möglich visualisieren** - Diagramme helfen Korrektheit zu verifizieren
7. **Strategisch implementieren** - Fokus auf distinkte Verhaltensweisen

Ergebnis: Handhabbare Testsuite mit starker Abdeckung! 

Fragen?

Was wir behandelt haben:

- ✓ Alle 22 Testfälle für `find_intersection()` mit Visualisierungen
- ✓ Fünf Dimensionen: Strukturell, Winkel, Position, Ergebnis, Form
 - ✓ Strategische Reduktion: 1800 → 22 Tests
- ✓ Wichtige Erkenntnis: Gleiches Ergebnis, unterschiedliche Dimensionen
 - ✓ Vollständige Testsuite-Struktur

Zurück zu: Haupt-Äquivalenzklassen-Folien für mehr Theorie

Weiter: Grenzwertanalyse (Testen an Grenzen)

Fragen? Diskussion?