Simulator (PREN1) Architektur-Dokumentation

Inhaltsverzeichnis

1. Einführung und Ziele

Das Modul PREN verlangt die Entwicklung eines autonomen Gerätes, das Puzzleteile erkennt, ausrichtet und platziert. Da in der Konzeptphase (PREN 1) noch kein physischer Prototyp des Gesamtgerätes zur Verfügung steht, soll ein Simulator entwickelt werden.

Dieser Simulator dient als Software-Demonstrator, um die Machbarkeit der algorithmischen Lösungsansätze zu beweisen und Risiken für die spätere Umsetzung in PREN 2 zu minimieren.

1.1. Aufgabenstellung

Die Hauptaufgabe dieses Simulators ist der Nachweis der Funktionalität der Objekterkennung, der Platzierungslogik sowie weiterer vom Team erkannten kritischen Teilfunktionen bevor Hardware gebaut wird.

Der Simulator muss folgende Anforderungen erfüllen:

- **Simulation des Puzzles:** Er muss Puzzleteile erkennen und virtuell abbilden können.
- **Algorithmus-Validierung:** Er soll demonstrieren, wie die erkannten Teile korrekt zugeordnet werden ("Greedy Matching" oder ähnliche Ansätze).
- **Standorterkennung:** Er soll jedem erkanntem Puzzleteil einen Standort zuweisen können.
- **Ausrichtung:** Der Simulator soll eine entsprechende Drehung (Rotation) und Standortverschiebung (Translation) durchführen.
- **Randbedingungen:** Die Simulation berücksichtigt die physikalischen Gegebenheiten wie die Startzone (A4) und die Ablagezone (A5 Rahmen).

1.2. Qualitätsziele

Die Qualitätsziele leiten sich aus der Notwendigkeit ab, das technische Risiko für das spätere physische Gerät zu senken.

Priorität	Qualitätsziel	Beschreibung
1	Funktionalität / Korrektheit	Der Simulator muss beweisen, dass der gewählte Algorithmus (z. B. Konturerkennung) fähig ist, das Puzzle autonom und korrekt zu lösen.

Priorität	Qualitätsziel	Beschreibung
2	Verständlichkeit	Die Schritte des Algorithmus müssen für Außenstehende (Stakeholder) nachvollziehbar aufzeigt werden (z. B. "Compare → Match → Place").
3	Übertragbarkeit	Die Logik des Simulators muss so konzipiert sein, dass sie später auf den Mikrocontroller des realen Roboters portiert werden kann (Vorbereitung PREN 2).

1.3. Stakeholder

Die Stakeholder sind die Personen und Gruppen, die ein direktes Interesse an der Funktionsfähigkeit und dem Nachweis durch den Simulator haben.

Rolle	Kontakt	Erwartungshaltung
Dozenten / Experten	Thomas Gisler (Coach der Gruppe 8)	Erwarten einen klaren Beweis (Proof of Concept), dass das Team eine funktionierende Lösung für die Objekterkennung und Platzierung sowie die weiteren kritischen Teilfunktionen erarbeitet hat. Sie bewerten die Risikominimierung.
Software-Entwicklungsteam	Studierende (Informatik)	Benötigen eine Testumgebung, um Algorithmen (z.B. Greedy Matching) ohne Hardware zu entwickeln und zu debuggen.
PREN-Board	HSLU T&A Bewertungsgremium	Achtet auf die Einhaltung der Wettbewerbsregeln (Autonomie, keine Kommunikation nach außen) und prüft, ob das Konzept tragfähig für PREN 2 ist.

2. Architecture Constraints (Randbedingungen)

Dieses Kapitel beschreibt die Einschränkungen und Vorgaben, die bei der Entwicklung des **Simulators** als Teil-Funktionsmuster in der Konzeptphase (PREN 1) zwingend einzuhalten sind. Diese Bedingungen definieren den architektonischen Rahmen für die Software.

2.1. Technische & Funktionale Randbedingungen

Diese Vorgaben definieren, welche Funktionalität der Simulator abbilden und welche physikalischen Gegebenheiten er berücksichtigen muss.

Randbedingung	Erläuterung

Simulationsumfang	Der Simulator muss die kritischen Teilfunktionen der Aufgabe abbilden: **Objekterkennung (Standorterkennung)**, **Ausrichtung (Rotation/Translation)** und **korrekte Platzierung** des Puzzles.
Puzzle-Parameter	Zur Vereinfachung wird die Logik für die Verarbeitung von **4 Puzzleteilen** ausgelegt sein.
Kernfunktionen	Der Simulator muss die Berechnung der notwendigen **Drehung (Rotation)** und **Standortverschiebung (Translation)** durchführen können, um die Teile korrekt zu platzieren.
Autonomie-Nachweis	Die gesamte Simulationssequenz muss **autonom** ablaufen, um die Machbarkeit des Algorithmus ohne menschliches Eingreifen zu beweisen (Vorgabe: kein Netzwerk/Kommunikation während des Vorgangs).
Schnittstelle (Input)	Der Simulator verarbeitet **statische, vordefinierte Daten** des Puzzlebildes, das von der Gruppe festgelegt wurde. Die Aufgabe ist die Verarbeitung dieser Daten (Konturen, Standorte), nicht deren Live-Erfassung durch eine Kamera.

2.2. Qualitäts- & Nachweis-Randbedingungen

Diese Randbedingungen stellen sicher, dass der Simulator seinen Zweck als Risikominderungs-Tool für die spätere Hardware-Entwicklung erfüllt (PREN 2).

Randbedingung	**Erläuterung**
Validierung	Die Simulation muss einen **eindeutigen Beweis** für die Korrektheit **aller als kritisch eingestuften Teilfunktionen** (u.a. Objekterkennung, Platzierungslogik, Matching-Algorithmus) liefern, um die Risikominimierung zu belegen.
Verständlichkeit	Die Simulation muss die **Schritte des Algorithmus** und den **aktuellen Zustand** (z. B. **Compare → Match → Place**, **Placed: x/y**) für Außenstehende nachvollziehbar aufzeigen.

Übertragbarkeit	Der **Kern-Algorithmus** zur Objekterkennung, Platzierungslogik, Standorterkennung und Ausrichtung muss so implementiert werden, dass er weitgehend unabhängig von anderen Bestandteilen des Simulators ist und eine **Portierung** auf die Embedded-Plattform (PREN 2) vereinfacht.
Code-Qualität	Die Prinzipien und Praktiken von **Clean Code** und die **Objektorientierung** sind anzuwenden, um die interne Modularität, Wartbarkeit und die Zusammenarbeit im Team zu sichern.
Architekturdokumentation	Die Architekturbeschreibung des Simulators muss zwingend das **arc42-Format** verwenden.

3. Kontextabgrenzung

Die Kontextabgrenzung definiert die **Systemgrenze** des Simulators und beschreibt, wie dieser mit seiner Umgebung interagiert. Der Simulator ist ein **Teilsystem** des Gesamtprojekts Produktentwicklung (PREN1 und PREN2) und dient als **Proof of Concept** für die kritischen Lösungsalgorithmen.

3.1. Fachlicher Kontext

Der fachliche Kontext beschreibt die Rolle des Simulators innerhalb der Problemdomäne (Puzzle lösen) und seine Interaktion mit dem übergeordneten Zielsystem (dem physischen PREN 2 Gerät).

Das folgende Diagramm zeigt die Einbettung des Simulators in den logischen Produktentwicklungszyklus, der durch die Notwendigkeit des Nachweises der kritischen Teilfunktionen bestimmt wird.

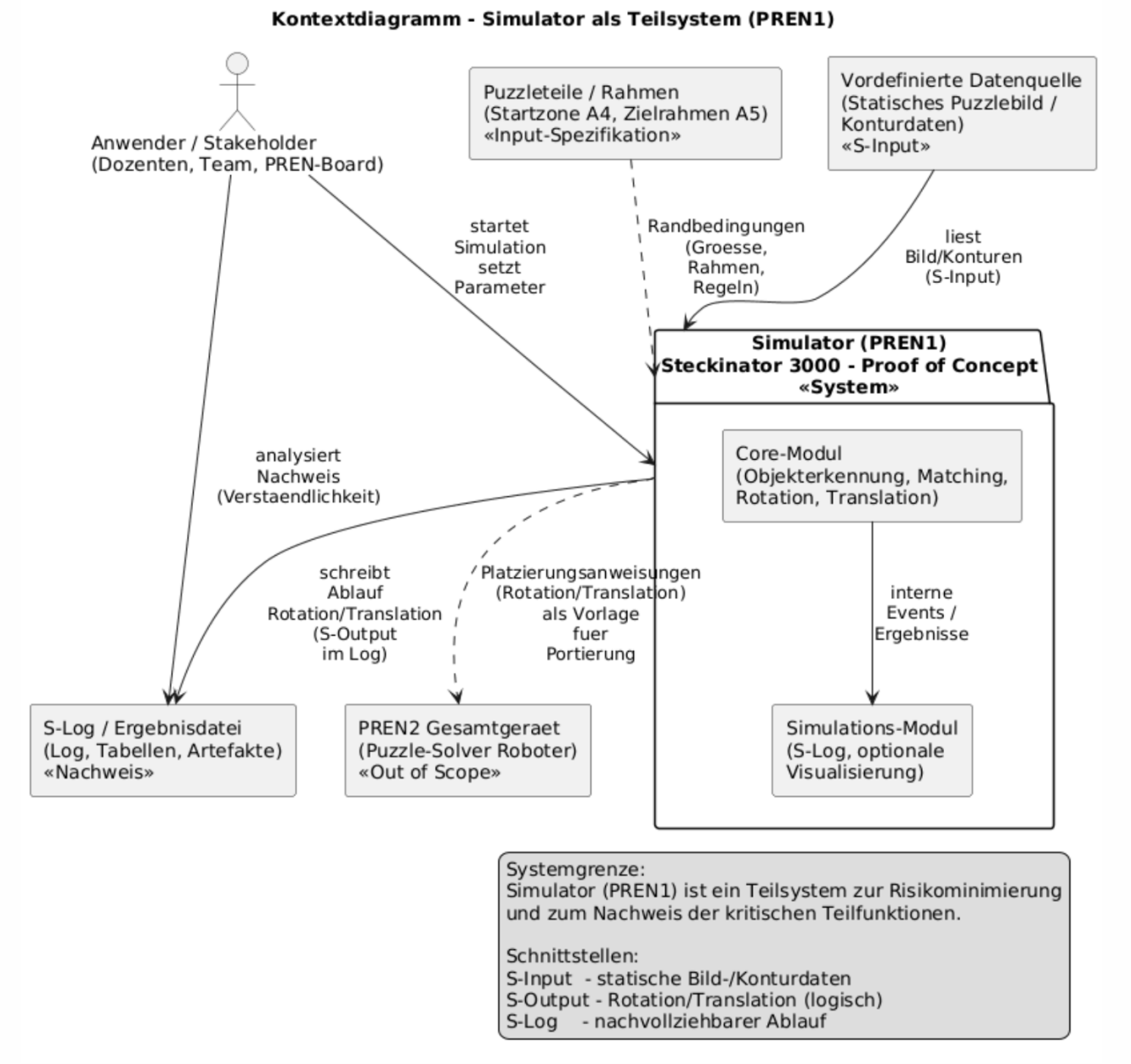

Abbildung: Kontextdiagramm

System (Fachlich)	Beziehung zum Simulator	Status
PREN 2 Gesamtgerät, Puzzle-Solver	Der Simulator ist das logische Vorbild für die Software des realen Roboters.	**Out of Scope** (Ist das Zielsystem, nicht das Simulationssystem.)
Puzzleteile / Rahmen	Das Problem, das gelöst werden muss (4–6 Teile, A5-Zielrahmen, A4-Startzone).	**Input-Spezifikation** (Beeinflusst die Regeln der Simulation.)
Kritische Teilfunktionen	Der Zweck des Simulators ist der Nachweis der Funktion dieser kritischen Algorithmen (z. B. Objekterkennung, Matching-Logik).	**Core Scope** (Liegt im Zentrum der Architektur.)

Erläuterung der externen fachlichen Schnittstellen

Die externen fachlichen Schnittstellen definieren die Art der Informationen, die das System verarbeitet und ausgibt:

- **Puzzleteil-Input:** Daten, die die Geometrie und die simulierte Startposition der Puzzleteile beschreiben.
- **Platzierungsanweisungen:** Die vom Simulator berechneten logischen Befehle (**Rotation** und **Translation**), die notwendig sind, um ein Puzzleteil korrekt abzulegen.

3.2. Technischer Kontext

Der technische Kontext beschreibt die Systemgrenze des Simulators als Softwareanwendung und deren Interaktion mit den technischen Nutzern und Datensystemen.

System (Technisch)	Beziehung zum Simulator	Anforderung
Anwender / Stakeholder	Steuert und überwacht die Simulation über die Benutzerschnittstelle.	Der Simulator muss die Lösung der eingegebenen Puzzle aufzeigen.
Vordefinierte Datenquelle	Liefert die statischen Konturdaten des Puzzlebildes.	Der Simulator muss eine **S-Input** Schnittstelle zur Verarbeitung der statischen Daten implementieren.
Entwicklungsumgebung	Wird zur Entwicklung und zum Debugging des Algorithmus-Kerns genutzt.	Der Simulator-Code muss die Prinzipien von **Clean Code** zur einfachen Wartung erfüllen.

Erläuterung der externen technischen Schnittstellen

- **S-UI (User Interface):** Die primäre Schnittstelle zur Interaktion mit den Stakeholdern. Sie zeigt den Algorithmus-Ablauf auf gibt den Zustand des Puzzles zurück.
- **S-Input (Datenkanal):** Der Datenkanal, der die statischen, vordefinierten Puzzle-Input-Daten einliest (z. B. aus einer Datei).
- **S-Output (Ergebniskanal):** Der Kanal, der die berechneten **Koordinaten (Translation)** und **Winkel (Rotation)** in einem definierten Format ausgibt.

Mapping fachliche auf technische Schnittstellen

Fachliche Schnittstelle	Technische Schnittstelle	Beschreibung
Puzzleteil-Input	S-Input (Datenkanal)	Der technische Kanal, über den die vordefinierten Konturdaten in das System gelangen.
Platzierungsanweisungen	S-Output (Ergebniskanal)	Die Ausgabe der berechneten Rotation und Translation, dargestellt über die S-GUI und protokolliert im S-Log.

Funktionsnachweis	S-UI und S-Log	Das Aufzeigen der Algorithmen (S-UI) und die Protokollierung (S-Log) dienen als direkter Beweis für die Korrektheit der kritischen Teilfunktionen.

4. Lösungsstrategie

Dieses Kapitel beschreibt die grundlegende Lösungsstrategie der Softwarearchitektur für den Steckinator 3000. Die Software ist modular aufgebaut und folgt einer klar strukturierten Verarbeitungskette, welche vom Erfassen der Puzzleteile bis zur Berechnung der Roboterbewegungen reicht. Jeder Teilbereich ist funktional abgegrenzt und kann unabhängig getestet, angepasst oder erweitert werden. Zentrales Element zur Validierung der Lösungsstrategie ist ein Simulator, der die komplette Puzzle-Logik abbildet.

4.1. Objekterkennung

Die Objekterkennung ist der erste Verarbeitungsschritt der Software und dient dazu, die einzelnen Puzzleteile zuverlässig im Kamerabild zu identifizieren. Die Lösung basiert auf klassischer Bildverarbeitung mit OpenCV. Nach der Umwandlung des Kamerabildes in ein Graustufenbild werden Störungen durch Weichzeichnung reduziert und anschliessend mittels Binarisierung Vorder- und Hintergrund getrennt. Über eine Konturerkennung werden die Umrisse der Puzzleteile extrahiert. Kleine Störungen oder Artefakte werden über Flächenfilter verworfen. Das Ergebnis dieses Schrittes sind saubere Aussenkonturen aller erkannten Puzzleteile.

4.2. Kantenseparation

Ziel der Kantenseparation ist es, die Konturen der Puzzleteile in einzelne, vergleichbare Kanten zu unterteilen. Dazu werden aus der Kontur stabile Eckpunkte bestimmt, welche als Trennstellen für die Kanten dienen. Die Kontur wird anschliessend in Segmente zerlegt und den jeweiligen Seiten des Puzzleteils zugeordnet. Dieser Schritt reduziert die Komplexität der Konturdaten und schafft die Grundlage für einen gezielten und strukturierten Kantenvergleich.

4.3. Kantenvergleich (Matching)

Der Kantenvergleich dient dazu, passende Kantenpaare zwischen unterschiedlichen Puzzleteilen zu identifizieren. Hierfür werden alle relevanten Kanten geometrisch normalisiert, sodass sie unabhängig von Position, Rotation und Skalierung vergleichbar sind. Zusätzlich werden die Kanten grob typisiert (z. B. Puzzlenase, Einbuchtung oder flache Kante), um unplausible Vergleiche frühzeitig auszuschliessen. Der eigentliche Vergleich erfolgt über einen Brute-Force-Ansatz, welcher aufgrund der geringen Anzahl von Puzzleteilen performant genug ist. Das Ergebnis ist eine Liste von Kantenpaaren mit zugehörigen Übereinstimmungswerten.

4.4. Puzzle-Zusammensetzung

Basierend auf den ermittelten Kantenmatches werden die Puzzleteile virtuell zusammengesetzt. Ein geeignetes Eckteil dient dabei als Start- oder Ankerpunkt. Weitere Teile werden iterativ anhand der Matching-Beziehungen positioniert und ausgerichtet. Dieser Schritt erzeugt eine konsistente Zielanordnung aller Puzzleteile im gelösten Zustand, unabhängig von deren ursprünglicher Lage.

4.5. Koordinatensystem und Skalierung

Um virtuelle Berechnungen mit der realen Arbeitsfläche des Roboters zu verknüpfen, wird ein einheitliches Koordinatensystem eingeführt. Die aus der Bildverarbeitung gewonnenen Konturen werden skaliert und verschoben, sodass sie den realen Abmessungen des Spielfelds entsprechen. Dadurch können sowohl die Ausgangspositionen der Puzzleteile als auch deren Zielpositionen im selben Referenzsystem beschrieben werden.

4.6. Berechnung von Rotation und Translation (S-Output Generierung)

Dieses Modul führt die kritische Risikominimierung durch, indem es die **exakten Bewegungsbefehle** für das spätere physische System generiert und protokolliert.

4.6.1. Koordinatensystem und Skalierung

Der Simulator arbeitet mit einem virtuellen Koordinatensystem, das an die realen Abmessungen der A4-Startzone und des A5-Zielrahmens gebunden ist.

- Die geladenen Konturdaten werden auf die korrekten **Skalierungsfaktoren** normiert.
- Das System enthält nach diesem Schritt zwei Datensätze: die **Ausgangsposition** (ungelöst) und die **Endposition** (gelöst) im gleichen virtuellen Raum.

4.6.2. Berechnung der Rotation

Die Rotation ist die Differenz im Winkel zwischen der aktuellen Orientierung des Puzzleteils und der benötigten Endorientierung im gelösten Zustand.

- **Methode:** Bestimmung eines Referenzvektors für jedes Teil im ungelösten und gelösten Zustand.
- **Resultat:** Berechnung des Rotationswinkels um den Flächenschwerpunkt.

4.6.3. Berechnung der Translation

Die Translation ist die Distanz, die das Puzzleteil (nach der Rotation) verschoben werden muss, um seine finale Position zu erreichen.

- **Methode:** Berechnung der Differenz der X- und Y-Koordinaten (Verschiebung) zwischen einem Referenzpunkt des ungelösten und des gelösten Puzzleteils.

- **S-Output:** Die **Rotation und Translation** für jedes Teil werden als **S-Output** generiert und in den S-Log protokolliert (z.B. "Place Teil X: Rotate 45°, Translate (424, 40)").

5. Klassendiagramm

Das nachfolgende Diagramm zeigt die Klassen und ihre Methoden des Simulators.

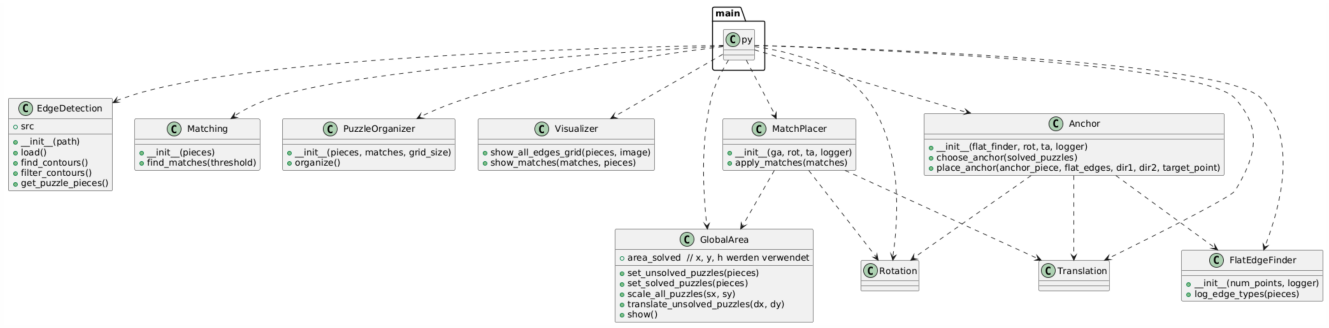

Abbildung: UML-Diagramm

6. Laufzeitsicht

Das nachfolgende Diagramm zeigt die Laufzeitsicht des Simulators.

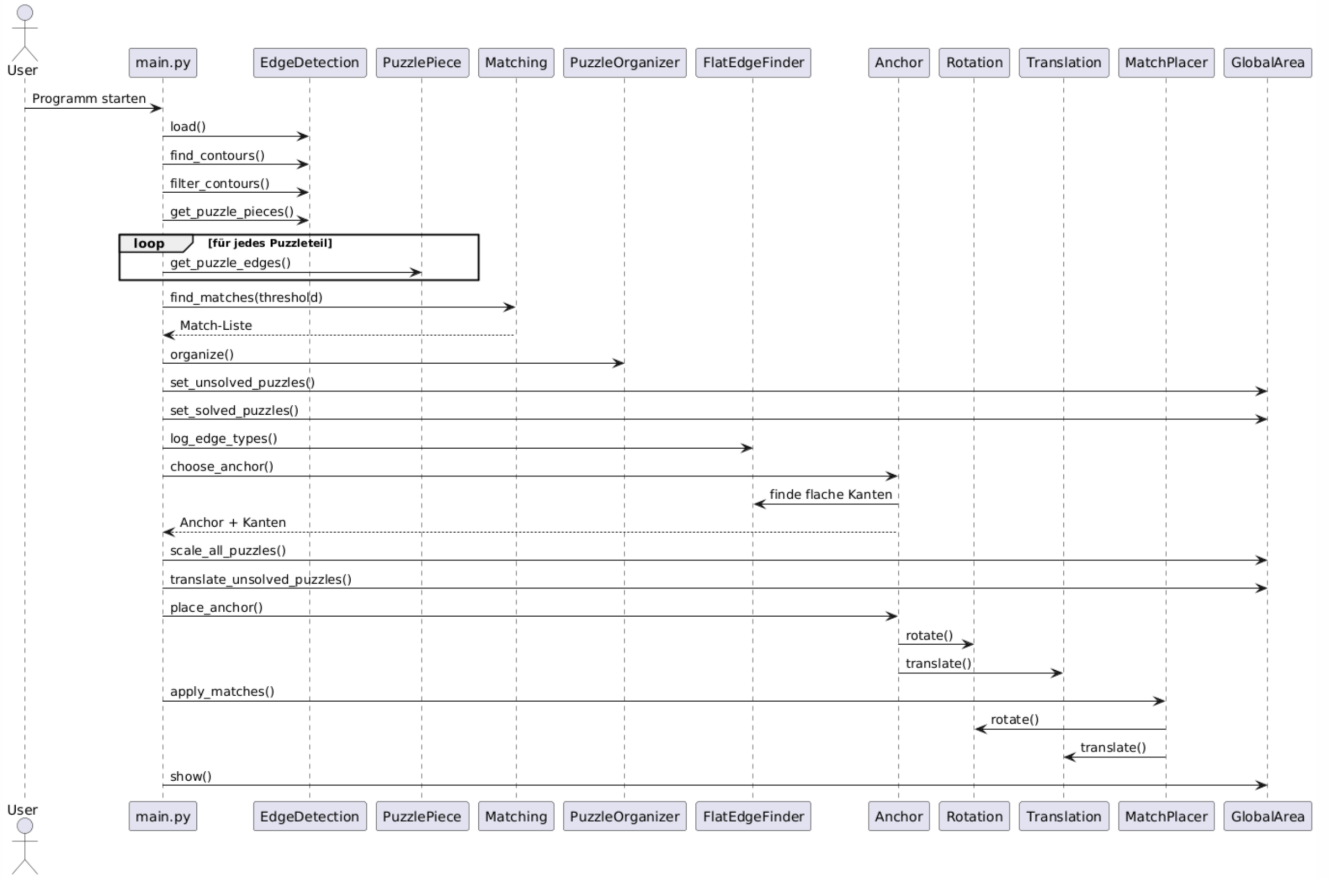

Abbildung: Sequenzdiagramm

7. Verteilungssicht

Der Simulator wird lokal auf einem Entwicklungsrechner ausgeführt. Es gibt keine verteilten Komponenten.

7.1. Laufzeit-Umgebung

Knoten	Beschreibung
Entwicklungsrechner	Windows/WSL oder Linux mit Python 3.x
Dateisystem	Projektverzeichnis mit Quellcode und statischem Puzzlebild

7.2. Abhängigkeiten

Komponente	Zweck
OpenCV	Bildverarbeitung und Konturen
NumPy	Array- und Matrizenoperationen
SciPy	Hilfsfunktionen für Matching/Geometrie
Matplotlib	Visualisierung der Ergebnisse

7.3. Start und Artefakte

- Startpunkt: `src/main.py`
- Input: statisches Bild (z. B. `Data/puzzle_selfmade_black.jpeg`)
- Output: Log-Ausgaben und optional Visualisierungen

8. Architekturkonzepte

Dieses Kapitel beschreibt die wichtigsten architektonischen Entscheidungen, die für die Realisierung des Simulators getroffen wurden, um die kritischen Teilfunktionen zu validieren und die Qualitätsziele zu erfüllen.

8.1. 1. Trennung von Logik und reinen Simulatorbestandteilen (Core-Konzept)

Das übergeordnete architektonische Konzept ist die strikte Trennung des Algorithmus-Kerns von den Ausgabefunktionen des Simulators. Dies adressiert direkt das Qualitätsziel der **Übertragbarkeit**.

- **Konzept:** Separation of Concerns (Trennung der Belange).
- **Ziel:** Der Kern-Algorithmus (**Core-Modul**) muss vollständig unabhängig von der Protokollierungs- und Aufzeichnungs-Logik sein, um eine einfache Portierung auf die

Embedded-Plattform in PREN 2 zu ermöglichen.

- **Umsetzung:**
- **Core-Modul:** Beinhaltet die gesamte Puzzle-Logik (Objekterkennung, Matching, Rotation/Translation-Berechnung).
- **Simulations-Modul:** Übernimmt die statische Datenaufnahme (**S-Input**) und die Bereitstellung des **S-Log**.

8.2. 2. Architekturprinzip (Objektorientierung)

Die Implementierung des Simulators erfolgt strikt nach dem Paradigma der **Objektorientierung (OOP)**. Dies ist ein fundamentales Prinzip zur Einhaltung der Qualitätsziele **Wartbarkeit** und **Code-Qualität**.

- **Ziel:** Verbesserung der internen Modularität, klarere Verantwortlichkeiten (Single Responsibility Principle) und einfachere **Zusammenarbeit im Projektteam**.
- **Umsetzung:** Jede logische Entität (wie Puzzleteil oder Matching Algorithmus) wird durch eine eigene Klasse abgebildet. Dies führt zu einer klaren **Kapselung** von Daten und Verhalten.

8.3. 3. Datenhaltung und Modellierung (Model-Konzept)

Dieses Konzept wird direkt durch die OOP-Entscheidung umgesetzt.

- **Puzzleteil-Objekt:** Jedes Puzzleteil wird als **instanziierte Klasse** abgebildet. Dieses Objekt enthält:
- **Konturdaten:** Eine Liste von Punkten/Vektoren zur Definition der Geometrie.
- **Aktuelle Position:** Aktuelle Koordinaten (x, y) und Winkel (Rotation) im simulierten Raum.

8.4. 4. Kontroll- und Ablaufkonzept

Da keine GUI existiert, wird die Steuerung über Konfiguration und Protokollierung realisiert, um die **Verständlichkeit** und den **Nachweis** zu gewährleisten.

- **Konfigurationsgesteuerte Ausführung:** Die Startparameter werden über das definierte Eingabebild und Kommandozeilenargumente gesteuert.
- **Detaillierte Protokollierung (S-Log):** Die Log-Funktion ersetzt die Echtzeit-Visualisierung. Der Algorithmus muss jeden diskreten, logischen Schritt (**Algorithm steps**) atomar in das Protokoll schreiben.
- **Simulationszeit:** Es wird keine physikalische Echtzeit-Simulation durchgeführt. Die

Ausführung ist **logisch-sequenziell**, wobei der **S-Log** die chronologische Abfolge beweist.

8.5. 5. Test- und Validierungskonzept (Qualitätsnachweis)

Das Testkonzept ist zentral, um das Qualitätsziel der **Korrektheit** zu belegen und die **Risikominimierung** für alle kritischen Teilfunktionen nachzuweisen.

- **Unit Tests für den Algorithmus-Kern:**
- **Fokus:** Der gesamte **Core-Modul** (Matching-Logik, Rotations- und Translationsberechnung) muss durch automatisierte Unit Tests abgedeckt werden.
- **Ziel:** Nachweis der **funktionalen Korrektheit** unabhängig von den Simulatorbestandteilen.
- **Manueller Validierungstest (Log-basiert):**
- **Fokus:** Überprüfung der **Korrektheit** des gesamten Lösungsablaufs durch die Stakeholder.
- **Ablauf:** Der **S-Log** wird analysiert. Die ausgegebenen **Platzierungsanweisungen (S-Output: Rotation/Translation)** müssen manuell mit den Soll-Werten für das vordefinierte Puzzle verglichen werden. Der S-Log dient als primärer **Funktionsnachweis**.

8.6. 6. Protokollierungs- und Aufzeigungskonzept (Funktionsnachweis)

Da keine Live-GUI existiert, ist die Protokollierung das primäre Medium für den Nachweis und die **Verständlichkeit**.

- **S-Log als primäres Output:** Der Simulator muss den **S-Log** (Konsole/Log-Datei) als detailliertes Protokoll verwenden.
- **Inhalt des Protokolls:** Das Protokoll muss alle Schritte enthalten, die zur Beantwortung der **Aufgabenstellung** relevant sind: Standorterkennung, Ergebnisse des Matching-Algorithmus (Match/No Match), und die finalen **Rotation**- und **Translation**-Werte für jedes platzierte Teil (S-Output).
- **Statische Ergebnisaufzeichnung:** Zusätzlich zum S-Log kann das System am Ende der Simulation eine **statische Ergebnisdatei** (z. B. eine Texttabelle) generieren, die den **finalen Zustand** des Puzzles im Zielrahmen tabellarisch festhält.

9. Architekturentscheidungen

Dieses Kapitel dokumentiert die wesentlichen Architektur- und Implementationsentscheide des Simulators.

9.1. ADR-01 Bildverarbeitung mit OpenCV

- Kontext: Die Objekterkennung basiert auf einem statischen Puzzlebild, das Konturen liefern muss.
- Entscheidung: Einsatz von OpenCV für Graustufen, Binarisierung und Konturerkennung.
- Begründung: Bewährte Bibliothek, stabile Ergebnisse, geringe Implementationszeit.
- Konsequenzen: Abhängigkeit von OpenCV, Parameter müssen für das Bildmaterial abgestimmt werden.

9.2. ADR-02 Heuristisches Kanten-Matching mit Schwellwert

- Kontext: Puzzlekanten müssen robust miteinander verglichen werden.
- Entscheidung: Matching mit Score und festem Schwellwert (Greedy-Auswahl der besten Treffer).
- Begründung: Einfach, nachvollziehbar und für 4 Teile ausreichend schnell.
- Konsequenzen: Risiko für Mehrdeutigkeiten, Skalierung für grössere Puzzles begrenzt.

9.3. ADR-03 Trennung von Core-Logik und Simulation

- Kontext: Der Algorithmus soll später auf eine Embedded-Plattform portierbar sein.
- Entscheidung: Kernlogik (Matching, Rotation, Translation) bleibt modular und von UI/Log getrennt.
- Begründung: Erleichtert Portierung in PREN 2 und gezielte Unit Tests.
- Konsequenzen: Mehr Schnittstellen, klare Datenstrukturen nötig.

9.4. ADR-04 Log-basierter Funktionsnachweis

- Kontext: Kein physischer Roboter verfügbar, Funktionsnachweis nur simuliert möglich.
- Entscheidung: Detaillierte Protokollierung als primärer Nachweis der Ablaufkette.
- Begründung: Stakeholder können Ablauf und Ergebnisse nachvollziehen.
- Konsequenzen: Logging muss konsistent und vollständig sein.

10. Qualitätsanforderungen

Dieses Kapitel konkretisiert die definierten Qualitätsziele anhand von Qualitätsszenarien. Da der Simulator primär als **Proof of Concept** dient, stehen die Korrektheit der Algorithmen und die Nachvollziehbarkeit des Ablaufs im Vordergrund.

10.1. Übersicht der Qualitätsanforderungen

Die wichtigsten Qualitätsziele und die zugehörigen Arc42-Kategorien sind:

Qualitätsziel	Kategorie (ISO 25010)	Begründung (Fokus des Simulators)
Korrektheit (Priorität 1)	Funktionalitätsangemessenheit	Der Simulator muss beweisen, dass der Algorithmus die Aufgabe lösen kann.
Verständlichkeit (Priorität 2)	Benutzerfreundlichkeit / Auditierbarkeit	Der gesamte Lösungsweg muss rein durch den **S-Log** nachvollziehbar sein.
Übertragbarkeit (Priorität 3)	Portierbarkeit / Modularität	Die Kernlogik muss vom Simulationscode isoliert und auf die Embedded-Plattform portierbar sein.
Wartbarkeit	Modifizierbarkeit / Analysierbarkeit	Die objektorientierte Struktur und Clean Code müssen Änderungen erleichtern.

10.2. Qualitätsszenarien

Die folgenden Szenarien beschreiben die kritischsten Qualitätsanforderungen aus der Sicht der Stakeholder:

10.2.1. 1. Korrektheit des Matching-Algorithmus (Proof of Concept)

Typ	Test-Szenario
Quelle	Dozenten, Informatikteam der Gruppe 8
Stimulus	Der Testfall mit den **4 vordefinierten Puzzleteilen** wird ausgeführt.
Umgebung	Simulator (Core-Modul) läuft in der Testumgebung (Unit Test).
Reaktion	Der Algorithmus berechnet für alle Puzzleteile die **korrekten Rotations- und Translationswerte** (S-Output).
Messung	Alle Unit Tests für die Matching-Logik laufen erfolgreich durch. Die berechneten S-Output-Werte stimmen exakt mit den manuell ermittelten Soll-Werten überein (Toleranz **0 mm** im logischen Modell).

10.2.2. 2. Nachvollziehbarkeit des Lösungsablaufs (Verständlichkeit)

Typ	Szenario zur Betriebssicherheit
Quelle	Dozenten
Stimulus	Ein Stakeholder (Experte) analysiert den vollständigen Lösungsablauf des Simulators.
Umgebung	Analyse des generierten **S-Log-Protokolls**.
Reaktion	Der S-Log muss alle relevanten Schritte detailliert protokollieren (Input-Daten, Match-Entscheidungen, finale Platzierungsbefehle).
Messung	Die Nachvollziehbarkeit des gesamten Ablaufs bis zur Platzierung aller Teile ist **in maximal 4 Minuten** allein durch die Analyse des S-Logs möglich.

10.2.3. 3. Trennung der Logik zur Portierung (Übertragbarkeit)

Typ	Entwicklungs-Szenario
Quelle	Informatikteam der Gruppe 8
Stimulus	Die Kernlogik-Dateien sollen aus dem Simulator-Projekt isoliert werden, um sie für das Embedded System in PREN 2 vorzubereiten.
Umgebung	Quellcode des **Core-Moduls** wird in eine leere Projektstruktur kopiert.
Reaktion	Das Core-Modul ist direkt kompilierbar, ohne dass Code der reinen Simulatorbestandteile (S-Log, S-Input) entfernt oder modifiziert werden muss.
Messung	Die Zeit, die benötigt wird, um den Kern-Algorithmus aus dem Simulatorkontext zu isolieren und ihn erfolgreich mit **Dummy-Elementen** in der neuen Zielumgebung zu kompilieren und auszuführen, beträgt **maximal eine Stunde**.

10.2.4. 4. Wartbarkeit (Erweiterung der Kernlogik)

Typ	Erweiterungs-Szenario
Quelle	Software-Entwicklungsteam

Stimulus	**Hypothetisch:** Es wird angenommen, die **simulierte Objekterkennung** muss durch eine komplexere, zuverlässigere Methode ersetzt werden, z.B. durch die Implementierung eines **Deep Learning**-Ansatzes, der aus erlernten Bilddaten die Konturen generiert.
Umgebung	Entwicklungsumgebung unter Anwendung des objektorientierten Designs (OOP).
Reaktion	Der neue Objekterkennungs-Ansatz kann als separate Klasse implementiert werden. Er muss lediglich das **definierte Output-Interface** (Ausgabe von Konturen, Standorten) einhalten. Das **Matching-Modul (Core-Logik)** und das **Simulations-Modul (S-Log)** dürfen keine Modifikationen erfordern.
Messung	Die Zeit, die benötigt wird, um den neuen Objekterkennungs-Ansatz zu integrieren und das System erfolgreich auszuführen, ohne Modifikationen an den **Core-Logik-Klassen** oder den **Simulations-Klassen** vornehmen zu müssen, beträgt **maximal 2 Stunden**.

11. Risiken und technische Schulden

In diesem Abschnitt werden die wesentlichen technischen Risiken der Softwarelösung beschrieben, welche die Funktionalität, Genauigkeit oder Robustheit des Gesamtsystems beeinträchtigen könnten.

ID	Risiko	Beschreibung
TR-01	Unzuverlässige Objekterkennung	Die bildbasierte Objekterkennung ist abhängig von Beleuchtung, Kontrast und Bildqualität. Ungünstige Umgebungsbedingungen können dazu führen, dass Puzzleteile nicht oder nur teilweise erkannt werden.
TR-02	Fehlerhafte Kantenseparation	Die geometrische Trennung der Konturen in einzelne Kanten kann bei komplexen oder unregelmässigen Puzzleformen fehlerhaft sein, was zu falsch definierten Kanten führt.

TR-03	Mehrdeutiges Kanten-Matching	Ähnliche oder symmetrische Kantenformen können zu mehreren gleichwertigen Matching-Ergebnissen führen, wodurch das Puzzle falsch oder inkonsistent zusammengesetzt wird.
TR-04	Falsche Puzzle-Zusammensetzung	Fehler in der Matching-Logik oder in der Einordnung der Puzzleteile können dazu führen, dass eine inkorrekte Zielanordnung berechnet wird.
TR-05	Ungenaue Rotationsberechnung	Ungenauigkeiten in der Konturerkennung oder Eckpunktbestimmung können sich in einer fehlerhaften Berechnung der Rotationswinkel widerspiegeln.
TR-06	Ungenaue Translationsberechnung	Abweichungen im Koordinatensystem oder in der Skalierung können dazu führen, dass berechnete Verschiebungen nicht exakt mit der realen Position übereinstimmen.
TR-07	Abweichung zwischen Simulation und Realität	Der Simulator bildet das reale System vereinfacht ab. Mechanische Toleranzen oder Kalibrierfehler können dazu führen, dass korrekt simulierte Bewegungen physisch nicht exakt umgesetzt werden.
TR-08	Performance-Engpässe	Bildverarbeitung und Kantenvergleich können bei ungünstigen Parametern oder ineffizienter Implementierung zu erhöhten Rechenzeiten führen.
TR-09	Schnittstelleninkonsistenzen	Unklare oder inkonsistente Definitionen von Koordinaten, Einheiten oder Referenzpunkten zwischen Software und Hardware können zu fehlerhaften Bewegungen führen.

TR-10	Eingeschränkte Skalierbarkeit	Die gewählte Brute-Force-Strategie ist für kleine Puzzlegrössen geeignet, skaliert jedoch schlecht bei steigender Anzahl von Puzzleteilen oder höherer Komplexität.

12. Glossar

Begriff	**Definition**
OpenCV	*Open Source Bildverarbeitungsframework für Konturerkennung und Filterung.*
PuzzlePiece	*Repr?sentiert ein einzelnes Puzzleteil mit Kontur und Kanten.*
PuzzleEdge	*Eine Kante eines Puzzleteils mit Punkten und Typ (innen, aussen, flach).*
Matching	*Algorithmus, der kompatible Kantenpaare anhand eines Scores bestimmt.*
Rotation	*Berechnung des Rotationswinkels zur korrekten Ausrichtung eines Teils.*
Translation	*Berechnung der Verschiebung im Koordinatensystem.*
Anchor	*Referenzteil mit flachen Kanten zur initialen Ausrichtung im Zielrahmen.*
FlatEdgeFinder	*Hilfskomponente zur Identifikation flacher Kanten.*
GlobalArea	*Container für gelöste/ungelöste Teile und Zielbereich im Koordinatensystem.*
S-Log	*Protokollierung des Simulationsablaufs als Nachweis der Funktionskette.*