Karlsruhe Institute of Technology

Robotik I: Einführung in die Robotik
Übung 8: Programmierung

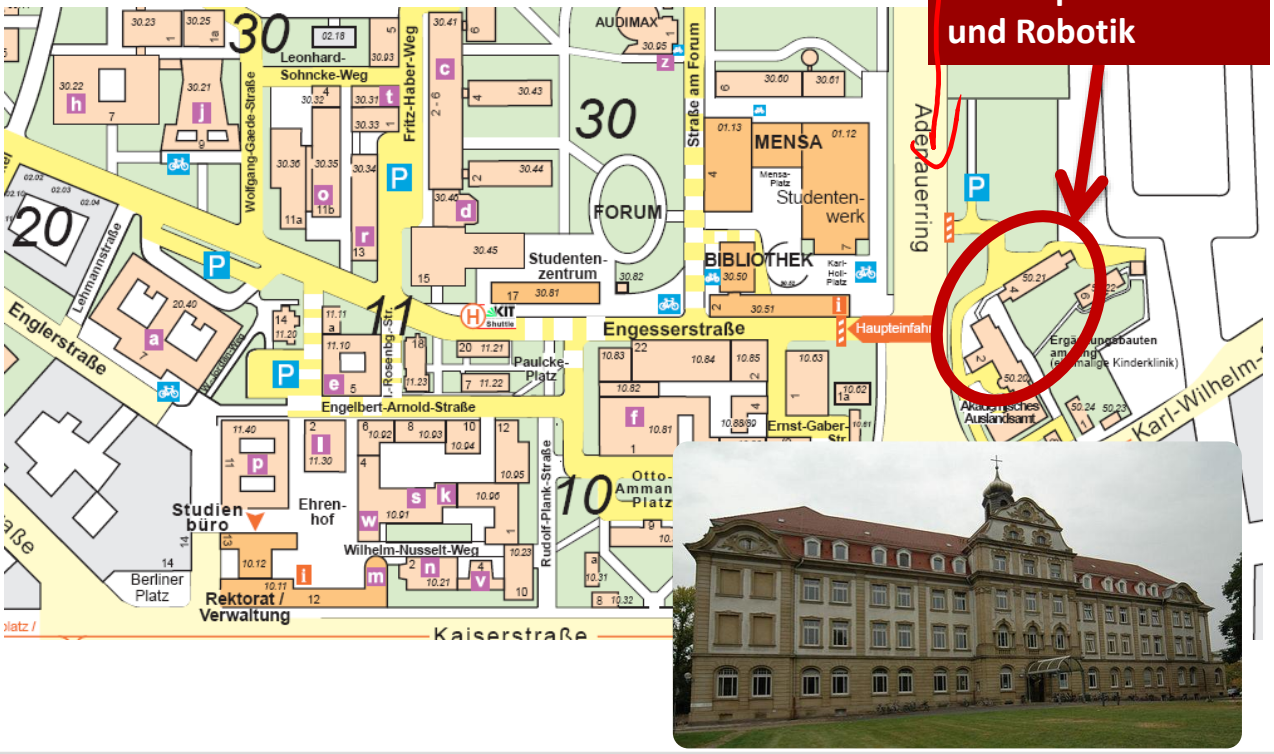
Fabian Paus, Tamim Asfour
Institut für Anthropomatik und Robotik

KIT-Fakultät für Informatik, Institut für Anthropomatik und Robotik (IAR)
Hochperformante Humanoide Technologien (H^2T)

Institutsführung

- Am 08.02.2018 findet die Robotik-1 Vorlesung **nicht** statt

- Stattdessen: Einladung zur **Institutsführung**
  - Donnerstag, 08.02, 9:45 – 11:15
  - Adenauerring 2,
    Geb. 50.20
    76131 Karlsruhe



Klausur

- Klausur:
  - Freitag, 16.03.2018
  - 14:00 - 15:00 Uhr
  - **Keine** Hilfsmittel (Taschenrechner, Unterlagen, etc.) erlaubt
  - Achtung: Nicht mit Bleistift schreiben!

- Hörsäle: Audi, Gerthsen
  - Verteilung wird kurz vor der Klausur bekannt gegeben

- Anmeldung im Campus-System:
  - Prüfungsnummern: 902 oder 209
  - Bei Fragen zur Anmeldung: sekretariat.asfour@anthropomatik.kit.edu

Übersicht

- Aufgabe 1: Symbolische Planung mit STRIPS
  - 1.1: Aktionssequenz angeben
  - 1.2: Planungsoperator erstellen
  - 1.3: Planungsoperator anwenden
- Aufgabe 2: Implementierung von A*
  - 2.1: Simox installieren
  - 2.2: Skeleton-Code kompilieren
  - 2.3: Lösung in Pseudocode schreiben
  - 2.4: Lösung in C++ schreiben
- Aufgabe 3: Implementierung von RRT

Aufgabe 1: Symbolische Planung mit STRIPS

- Initial state: Table(T), Stove(S), Location(T), Location(S), Gripper(G), Hand(G), Robot(R), Agent(R), Pan(P), Empty(G), On(P, T), At(R, S)
- Goal state: On(P, S)

Aufgabe 1: Symbolische Planung mit STRIPS

- Actions:
  - Agent A nimmt Objekt O mit Hand H von Ort L
    pickup(A, O, H, L)

    Preconditions: Agent(A), Hand(H), Location(L), Empty(H),
    On(O, L), At(A, L)

    Effects: !On(O, L), !Empty(H), InHand(O, H)

  - Agent A platziert Objekt O mit Hand H auf Ort L
    putdown(A, O, H, L)

    Preconditions: Agent(A), Hand(H), Location(L),
    InHand(O, H), At(A, L)

    Effects: On(O, L), Empty(H), !InHand(O, H)

Aufgabe 1: Symbolische Planung mit STRIPS

- Actions:
  - Agent A bewegt sich von Ort L zu Ort M
    move(A, L, M)

    Preconditions: Agent(A), Location(L), Location(M),
    At(A, L) $L \neq M$

    Effects: !At(A, L), At(A, M)

move(R, S, S)

Effects: !At(R, S), At(R, S)

↳ Widerspruch

Aufgabe 1: Symbolische Planung mit STRIPS

- Actions:
  - Agent A bewegt sich von Ort L zu Ort M
    move(A, L, M)

    Preconditions: Agent(A), Location(L), Location(M),
    At(A, L), $L \neq M$

    Effects: !At(A, L), At(A, M)

  - $L \neq M$
    Notwendig, da sonst Widersprüche entstehen

Aufgabe 1: Symbolische Planung mit STRIPS

1. Geben sie die kürzeste **Aktionssequenz** an:

   Initialer Weltzustand ➔ Zielzustand

2. Erstellen Sie einen **Planungsoperator**

   ```
   moveAndPickup(A, O, H, L, M),
   ```

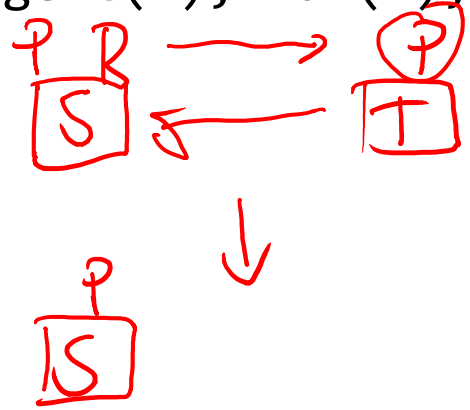
   der den Agenten A von Ort L nach M bewegt und von dort das Objekt O mit der Hand H aufnimmt.

3. Bestimmen Sie den **Weltzustand** nach

   ```
   moveAndPickup(R, P, G, S, T)
   ```

Aufgabe 1.1: Aktionssequenz

- Initial state:
  Table(T), Stove(S), Location(T), Location(S),
  Gripper(G), Hand(G), Robot(R), Agent(R), Pan(P),
  Empty(G), On(P, T), At(R, S)



- Goal state: On(P, S)

Aufgabe 1.1: Aktionssequenz

- Initial state:
  Table(T), Stove(S), Location(T), Location(S),
  Gripper(G), Hand(G), Robot(R), Agent(R), Pan(P),
  Empty(G), On(P, T), At(R, S)

- Goal state: On(P, S)

- Zum Table T gehen:
  - Vorbedingungen prüfen
  - Effekte anwenden

```
move(R, S, T)
Pre:   Agent(R), Location(S),
       Location(T), At(R, S), S ≠ T
Eff:   !At(R, S), At(R, T)
```

Aufgabe 1.1: Aktionssequenz

- Initial state:
  Table(T), Stove(S), Location(T), Location(S),
  Gripper(G), Hand(G), Robot(R), Agent(R), Pan(P),
  Empty(G), On(P, T), At(R, S)

- Goal state: On(P, S)

- Zum Table T gehen:
  - **Vorbedingungen prüfen**
  - Effekte anwenden

```
move(R, S, T)
Pre:   Agent(R), Location(S),
       Location(T), At(R, S), S ≠ T
Eff:   !At(R, S), At(R, T)
```

Aufgabe 1.1: Aktionssequenz

- Current state:
  Table(T), Stove(S), Location(T), Location(S),
  Gripper(G), Hand(G), Robot(R), Agent(R), Pan(P),
  Empty(G), On(P, T), ~~At(R, S)~~, At(R, T)
- Goal state: On(P, S)
- Zum Table T gehen:
  - Vorbedingungen prüfen
  - **Effekte anwenden**

move(R, S, T)
Pre: Agent(R), Location(S),
Location(T), At(R, S), S ≠ T
Eff: !At(R, S), At(R, T)

Aufgabe 1.1: Aktionssequenz

- Current state:
  `Table(T), Stove(S), Location(T), Location(S), Gripper(G), Hand(G), Robot(R), Agent(R), Pan(P), Empty(G), On(P, T), At(R, T)`
- Goal state: `On(P, S)`
- Pan P vom Table T aufnehmen:
  - Vorbedingungen prüfen
  - Effekte anwenden

```
pickup(R, P, G, T)
Pre:   Agent(R), Hand(G), Location(T),
       Empty(G), On(P, T), At(R, T)
Eff:   !On(P, T), !Empty(G), InHand(P, G)
```

Aufgabe 1.1: Aktionssequenz

- Current state:
  Table(T), Stove(S), Location(T), Location(S),
  Gripper(G), Hand(G), Robot(R), Agent(R), Pan(P),
  Empty(G), On(P, T), At(R, T)

- Goal state: On(P, S)

- Pan P vom Table T aufnehmen:
  - **Vorbedingungen prüfen**
  - Effekte anwenden

statisch

```
pickup(R, P, G, T)
Pre:   Agent(R), Hand(G), Location(T),
       Empty(G), On(P, T), At(R, T)
Eff:   !On(P, T), !Empty(G), InHand(P, G)
```

Aufgabe 1.1: Aktionssequenz

- Current state:
  Table(T), Stove(S), Location(T), Location(S),
  Gripper(G), Hand(G), Robot(R), Agent(R), Pan(P),
  ~~Empty(G)~~, ~~On(P, T)~~, At(R, T), InHand(P, G)

- Goal state: On(P, S)

- Pan P vom Table T aufnehmen:
  - Vorbedingungen prüfen
  - **Effekte anwenden**

pickup(R, P, G, T)
Pre: Agent(R), Hand(G), Location(T),
Empty(G), On(P, T), At(R, T)
Eff: !On(P, T), !Empty(G), InHand(P, G)

Aufgabe 1.1: Aktionssequenz

- Current state:
  `Table(T), Stove(S), Location(T), Location(S), Gripper(G), Hand(G), Robot(R), Agent(R), Pan(P), At(R, T), InHand(P, G)`
- Goal state: `On(P, S)`
- Vom Table T zum Stove S bewegen:
  - Vorbedingungen prüfen
  - Effekte anwenden

```
move(R, T, S)
Pre:   Agent(R), Location(T),
       Location(S), At(R, T), T ≠ S
Eff:   !At(R, T), At(R, S)
```

Aufgabe 1.1: Aktionssequenz

- Current state:
  Table(T), Stove(S), Location(T), Location(S),
  Gripper(G), Hand(G), Robot(R), Agent(R), Pan(P),
  At(R, T), InHand(P, G)

- Goal state: On(P, S)

- Vom Table T zum Stove S bewegen:
  - **Vorbedingungen prüfen**
  - Effekte anwenden

```
move(R, T, S)
Pre:   Agent(R), Location(T),
       Location(S), At(R, T), T ≠ S
Eff:   !At(R, T), At(R, S)
```

Aufgabe 1.1: Aktionssequenz

- Current state:
  Table(T), Stove(S), Location(T), Location(S),
  Gripper(G), Hand(G), Robot(R), Agent(R), Pan(P),
  ~~At(R, T)~~, InHand(P, G), At(R, S)

- Goal state: On(P, S)

- Vom Table T zum Stove S bewegen:
  - Vorbedingungen prüfen
  - **Effekte anwenden**

move(R, T, S)
Pre: Agent(R), Location(T),
Location(S), At(R, T), T ≠ S
Eff: !At(R, T), At(R, S)

Aufgabe 1.1: Aktionssequenz

- Current state:
  `Table(T), Stove(S), Location(T), Location(S),`
  `Gripper(G), Hand(G), Robot(R), Agent(R), Pan(P),`
  `InHand(P, G), At(R, S)`
- Goal state: `On(P, S)`
- Pan P auf Stove S absetzen:
  - Vorbedingungen prüfen
  - Effekte anwenden

```
putdown(R, P, G, S)
Pre:   Agent(R), Hand(G), Location(S),
       InHand(P, G), At(R, S)
Eff:   On(P, S), Empty(G), !InHand(P, G)
```

Aufgabe 1.1: Aktionssequenz

- Current state:
  Table(T), Stove(S), Location(T), Location(S),
  Gripper(G), Hand(G), Robot(R), Agent(R), Pan(P),
  InHand(P, G), At(R, S)

- Goal state: On(P, S)

- Pan P auf Stove S absetzen:
  - **Vorbedingungen prüfen**
  - Effekte anwenden

```
putdown(R, P, G, S)
Pre:   Agent(R), Hand(G), Location(S),
       InHand(P, G), At(R, S)
Eff:   On(P, S), Empty(G), !InHand(P, G)
```

Aufgabe 1.1: Aktionssequenz

- Current state:
  Table(T), Stove(S), Location(T), Location(S),
  Gripper(G), Hand(G), Robot(R), Agent(R), Pan(P),
  ~~InHand(P, G)~~, At(R, S), On(P, S), Empty(G)

- Goal state: On(P, S)

- Pan P auf Stove S absetzen:
  - Vorbedingungen prüfen
  - **Effekte anwenden**

```
putdown(R, P, G, S)
Pre:   Agent(R), Hand(G), Location(S),
       InHand(P, G), At(R, S)
Eff:   On(P, S), Empty(G), !InHand(P, G)
```

Aufgabe 1.1: Aktionssequenz

- Current state:
  Table(T), Stove(S), Location(T), Location(S),
  Gripper(G), Hand(G), Robot(R), Agent(R), Pan(P),
  At(R, S), On(P, S), Empty(G)
- Goal state: On(P, S)
- Action sequence:
  1. move(R, S, T)
  2. pickup(R, P, G, T)
  3. move(R, T, S)
  4. putdown(R, P, G, S)

Aufgabe 1.2: Planungsoperator erstellen

- Erstellen Sie einen **Planungsoperator**

  ```
  moveAndPickup(A, O, H, L, M),
  ```

  der den Agenten A von Ort L nach M bewegt und von dort das Objekt O mit der Hand H aufnimmt.

- STRIPS-Notation:
  - Vorbedingungen
  - Effekte

Aufgabe 1.2: Planungsoperator erstellen

- **moveAndPickup(A, O, H, L, M)**

  Preconditions:

  Agent(A), Hand(H), Location(L), Location(M), L ≠ M
  At(A, L), On(O, M),
  Empty(H)

  Effects:

Aufgabe 1.2: Planungsoperator erstellen

- **moveAndPickup(A, O, H, L, M)**

  Preconditions: Agent(A), Hand(H),
  Location(L), Location(M), L ≠ M,

  At(A, L), Empty(H), On(O, M)

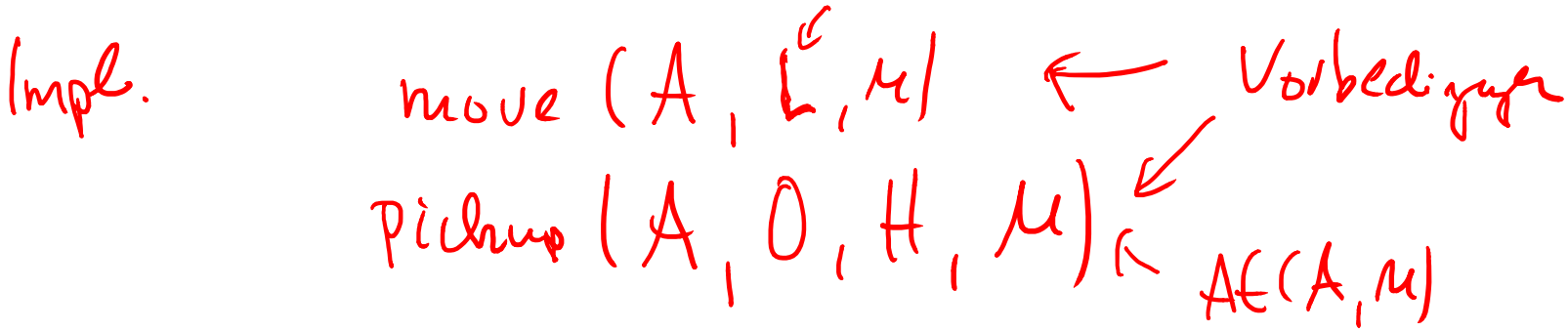
  Effects:

  !At(A, L), At(A, M)
  !On(O, M), InHand(H, O), !Empty(H)

Aufgabe 1.2: Planungsoperator erstellen

- **moveAndPickup(A, O, H, L, M)**

```
Preconditions:   Agent(A), Hand(H),
                 Location(L), Location(M), L ≠ M,

                 At(A, L), Empty(H), On(O, M)

Effects:         !Empty(H), !At(A, L), At(A, M),
                 !On(O, M), InHand(O, H)
```



Aufgabe 1.3 Planungsoperator anwenden

- Bestimmen Sie den **Weltzustand** nach

```
moveAndPickup(R, P, G, S, T)
```

- Initial state:
  `Table(T), Stove(S), Location(T), Location(S),`
  `Gripper(G), Hand(G), Robot(R), Agent(R), Pan(P),`
  `Empty(G), On(P, T), At(R, S)`

Aufgabe 1.3 Planungsoperator anwenden

- Initial state:
  Table(T), Stove(S), Location(T), Location(S),
  Gripper(G), Hand(G), Robot(R), Agent(R), Pan(P),
  Empty(G), On(P, T), At(R, S)

- Vorbedingungen prüfen
- Effekte anwenden

moveAndPickup(R, P, G, S, T)

Pre: Agent(R), Hand(G), Location(S),
Location(T), S ≠ T, At(R, S),
Empty(G), On(P, T)

Eff: !Empty(G), !At(R, S), At(R, T),
!On(P, T), InHand(P, G)

Aufgabe 1.3 Planungsoperator anwenden

- Initial state:
  Table(T), Stove(S), Location(T), Location(S),
  Gripper(G), Hand(G), Robot(R), Agent(R), Pan(P),
  Empty(G), On(P, T), At(R, S)

- **Vorbedingungen prüfen**
- Effekte anwenden

moveAndPickup(R, P, G, S, T)

Pre: Agent(R), Hand(G), Location(S),
Location(T), S ≠ T, At(R, S),
Empty(G), On(P, T)

Eff: !Empty(G), !At(R, S), At(R, T),
!On(P, T), InHand(P, G)

Aufgabe 1.3 Planungsoperator anwenden

- Current state:
  Table(T), Stove(S), Location(T), Location(S),
  Gripper(G), Hand(G), Robot(R), Agent(R), Pan(P),
  ~~Empty(G)~~, ~~On(P, T)~~, ~~At(R, S)~~, At(R, T), InHand(P, G)

- Vorbedingungen prüfen
- **Effekte anwenden**

moveAndPickup(R, P, G, S, T)

Pre: Agent(R), Hand(G), Location(S),
Location(T), S ≠ T, At(R, S),
Empty(G), On(P, T)

Eff: !Empty(G), !At(R, S), At(R, T),
!On(P, T), InHand(P, G)

Aufgabe 1.3 Planungsoperator anwenden

- Current state:
  Table(T), Stove(S), Location(T), Location(S),
  Gripper(G), Hand(G), Robot(R), Agent(R), Pan(P),
  At(R, T), InHand(P, G)

Δ Weltzustand

```
moveAndPickup(R, P, G, S, T)
Pre:    Agent(R), Hand(G), Location(S),
        Location(T), S ≠ T, At(R, S),
        Empty(G), On(P, T)
Eff:    !Empty(G), !At(R, S), At(R, T),
        !On(P, T), InHand(P, G)
```

- Antwort beinhaltet den **gesamten** Weltzustand, nicht nur das **Delta**!

Aufgabe 2: Implementierung von A*

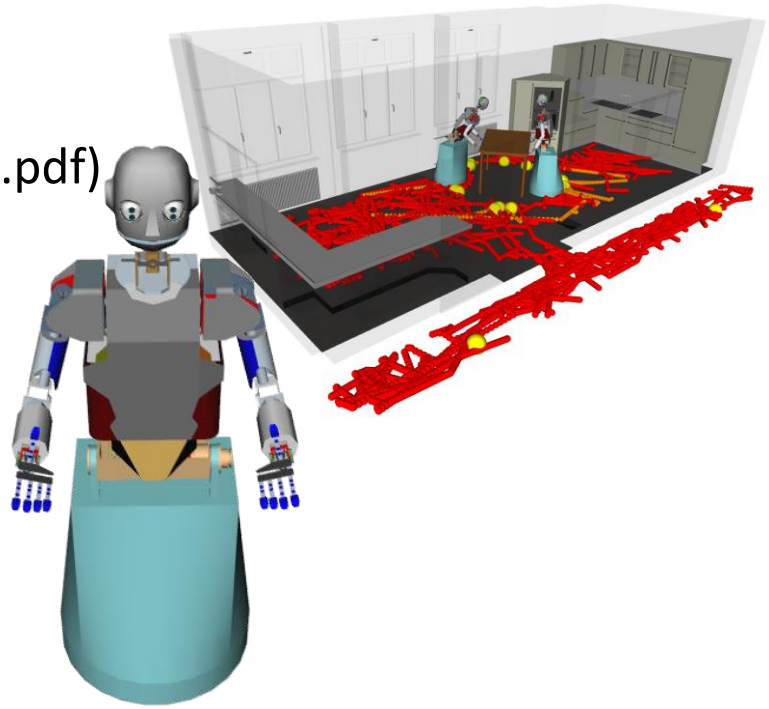
1. Simox installieren
2. Skeleton-Code kompilieren
3. Lösung in Pseudocode schreiben
4. Lösung in C++ schreiben

Aufgabe 2.1: Simox installieren

- Installation:
  https://gitlab.com/Simox/simox/wikis/Installation-Source-Ubuntu

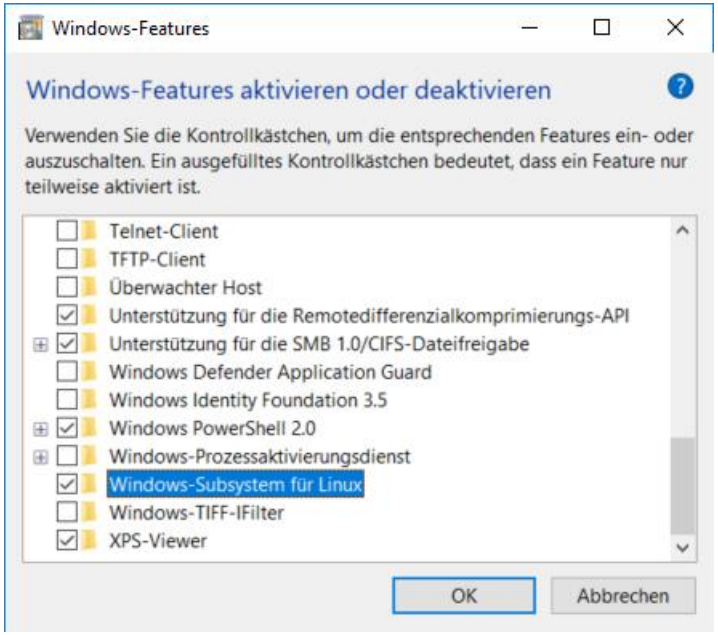
- Ubuntu: Am besten getestet

- Windows: Ubuntu-Subsystem empfohlen
  ➔ Siehe Anleitung (Simox-unter-Windows.pdf)



Aktiviere das Windows Subsystem für Linux (WSL)

- **Start -> Suche** „windows-features aktivieren oder deaktivieren“

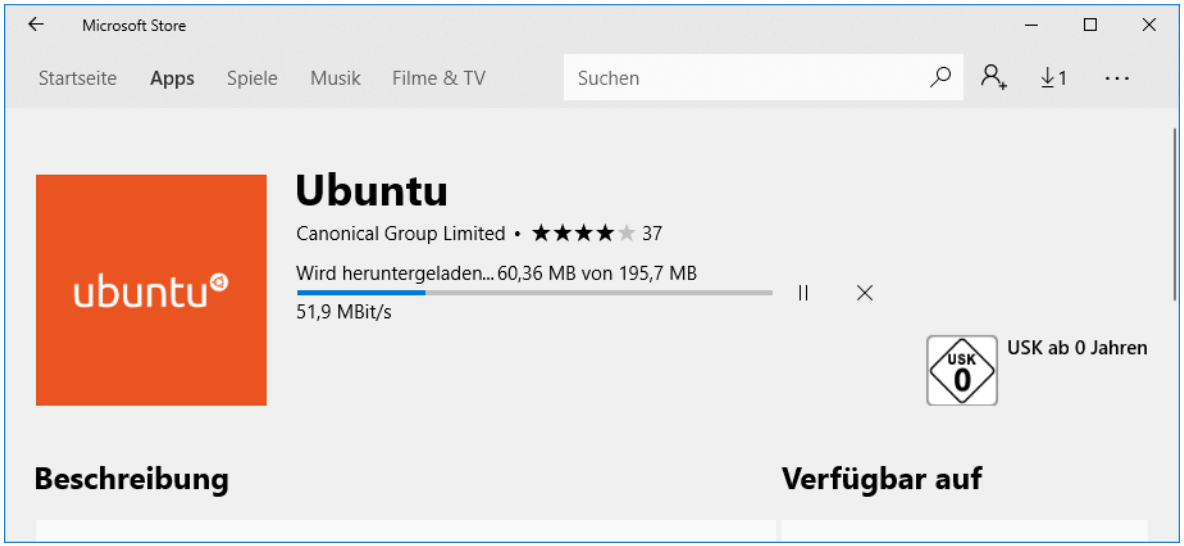


- Nach dem Aktivieren ist ein Neustart nötig

Hierfür ist ein aktuelles Windows 10 nötig!

Installation Ubuntu und X-Server

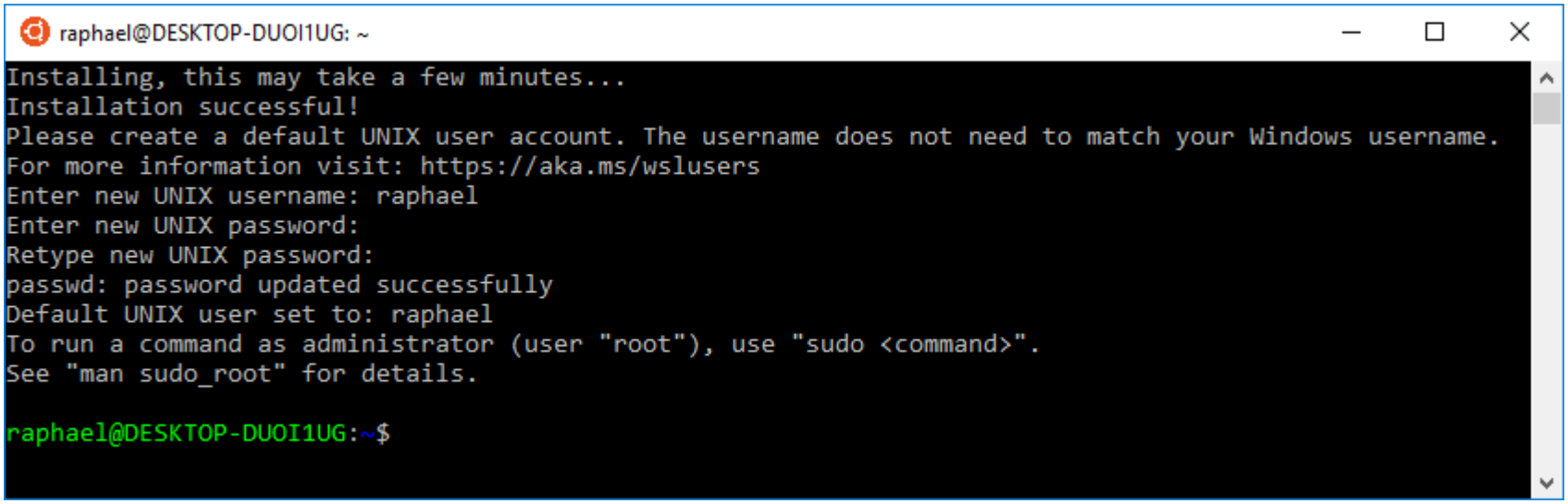
- **Start -> Store -> Ubuntu**



- Installiere einen X-Server für Windows für graphische Anwendungen
  - Getestet: https://sourceforge.net/projects/xming/

Initiales Setup

- **Start -> Ubuntu**
- Nutzer erstellen



- Update das System und Installiere Basispakete
  - sudo apt-get update && sudo apt-get upgrade
  - sudo apt-get install build-essential make cmake freeglut3-dev
- Konfiguriere den X-Server in Ubuntu
  - export DISPLAY=:0
  - echo "export DISPLAY=:0" >> ~/.bashrc

Installation Simox

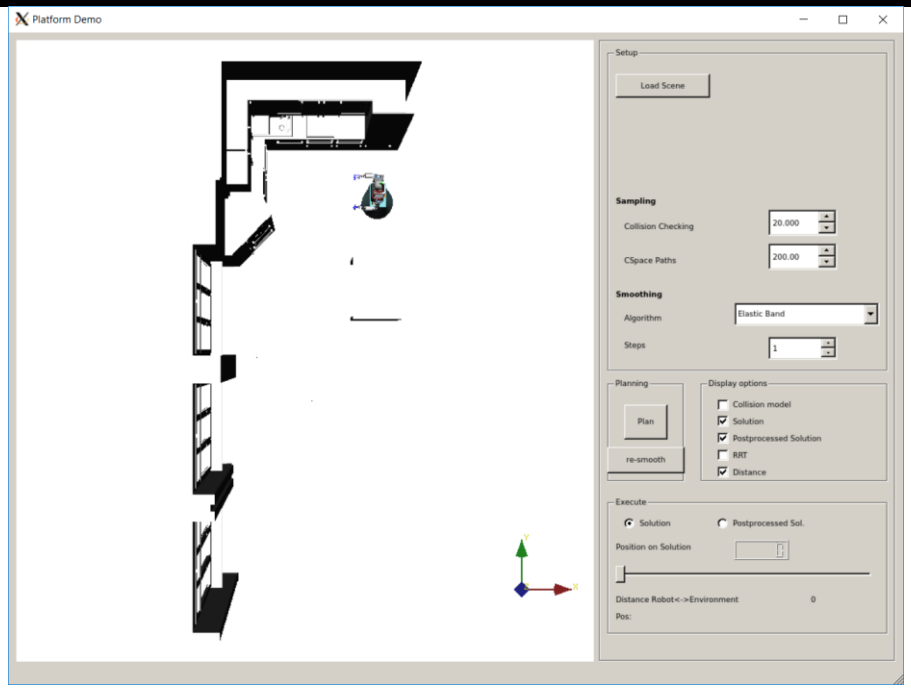
- Folge der Simox Installationsanleitung für Ubuntu
  - https://gitlab.com/Simox/simox/wikis/Installation-Source-Ubuntu

```
cd ~/
sudo apt-get install libboost-all-dev libeigen3-dev libsoqt4-dev libcoin80-dev libqt4-dev libnlopt-dev

wget https://github.com/bulletphysics/bullet3/archive/2.83.7.tar.gz
tar xf 2.83.7.tar.gz
mkdir -p bullet3-2.83.7/build
cd bullet3-2.83.7/build
cmake .. -DBUILD_SHARED_LIBS=on -DCMAKE_BUILD_TYPE=Release -DUSE_DOUBLE_PRECISION=on
make -j8

cd ../..

git clone https://gitlab.com/Simox/simox.git
mkdir -p simox/build
cd simox/build
cmake .. -DCMAKE_BUILD_TYPE=Release -DSimox_BUILD_SimDynamics=on \
         -DSimDynamics_USE_BULLET_DOUBLE_PRECISION=on \
         -DBULLET_ROOT=../../bullet3-2.83.7/build -DBULLET_INCLUDE_DIR=../../bullet3-2.83.7/src
make -j8
```

Benutze die Simox-Beispiele

- Starte den X-Server (falls nicht bereits gestartet)
- Führe ein Beispiel aus **simox/build/bin** aus

```
raphael@DESKTOP-DUOI1UG: ~
raphael@DESKTOP-DUOI1UG:~$ ./simox/build/bin/PlatformDemo
 --- START ---
 ********** Simox RuntimeEnvironment **********
Data paths:
 * /home/raphael/simox/VirtualRobot/data
Known keys:
 * scene
```



Aufgabe 2.2: Skeleton-Code kompilieren

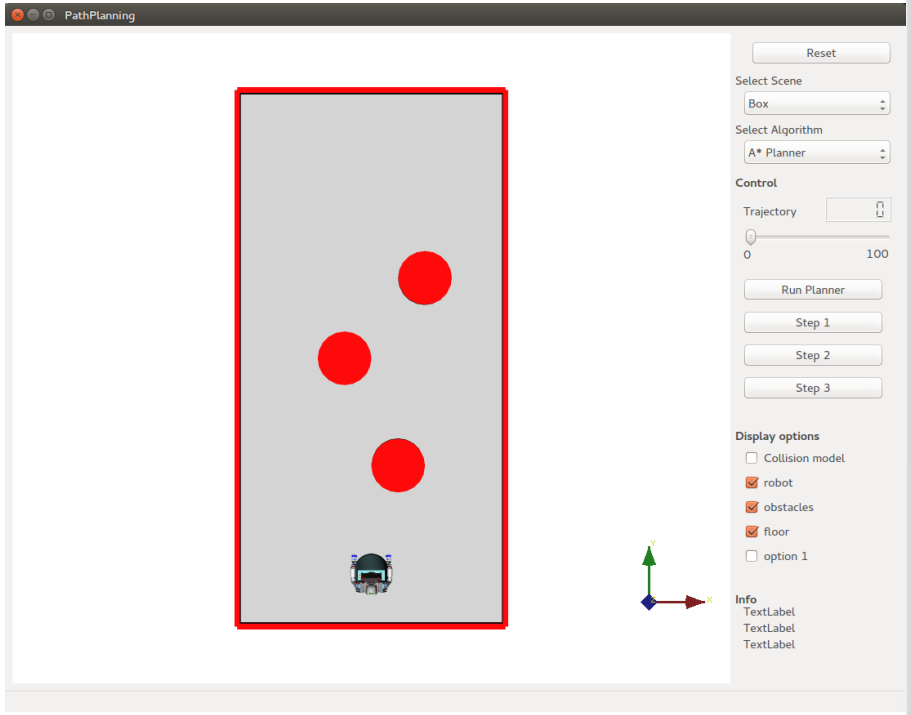
- Folgende Dateien wurden im ILIAS bereit gestellt:
  - `motion-planning-simox.zip`: Projektdateien zur Bearbeitung
  - `motion-planning-simox-solution.zip`: Musterlösung

- Vorgehen:
  - Entpacken von `motion-planning-simox.zip`
  - Folgende Befehle verwenden

```
cd build
cmake ..
make
```

  - Die gebaute Anwendnug kann jetzt gestartet werden:

```
./PathPlanning
```



Aufgabe 2.2: Skeleton-Code kompilieren

- Aufbau des Projekts:

```
CMakeLists.txt
PathPlanning.cpp
PathPlanning.ui
PathPlanningWindow.cpp
PathPlanningWindow.h
Planner/
|- AStarPlanner.cpp   ← In dieser Datei arbeiten wir
|- AStarPlanner.h
|- Node.cpp
|- Node.h
|- …
build/
|- PathPlanning   ← Hier liegt gebaute ausführbare Datei
…
```

Aufgabe 2.3: Lösung in Pseudocode

- Wiederholung: A*-Algorithmus
- Implementierung in Pseudocode ➔ Kommentare

A*-Algorithmus

- Iterativer Ansatz
- Es werden zwei Knotenlisten verwaltet
  - **Open Set** O: Noch zu besuchende Knoten
  - **Closed Set** C: Bereits besuchte Knoten

- **Update**: Für einen besuchten Knoten v_n:
  - **Vorgängerknoten** $pred(v_n)$
  - **Akkumulierte Kosten**, um v_n zu erreichen: $g(v_n)$
  - **Heuristik** für die erwarteten Kosten zum Ziel: $h(v_n)$

- **Initialisierung**
  - $O = \{v_s\}$
  - $C = \{\}$
  - $g(v_i) = \infty, \quad 1 \leq i \leq K$
  - $g(v_s) = 0$

A*-Algorithmus

- **Algorithmus**
  Solange $O \neq \emptyset$
  - Bestimme den zu erweiternden Knoten
    - Finde $v_i \in O$ mit minimalem $f(v_i) = g(v_i) + h(v_i)$
  - Wenn $v_i = v_{ziel}$
    Lösung gefunden: Traversiere Vorgänger von v_i bis v_{start} erreicht ist.
  - $O.remove(v_i)$
  - $C.add(v_i)$
  - **Update** für alle Nachfolger v_j von v_i durchführen
    - Wenn $v_j \in C$, dann überspringe v_j
    - Wenn $v_j \notin O$, dann $O.add(v_j)$
    - Wenn $g(v_i) + cost(v_i, v_j) < g(v_j)$
      - $g(v_j) = g(v_i) + cost(v_i, v_j)$
      - $h(v_j) = heuristic(v_j, v_{ziel})$
      - $pred(v_j) = v_i$

Aufgabe 2.4: Lösung in C++

- Siehe Musterlösung

Aufgabe 3: Implementierung von RRT*

- Aufbau des Projekts:

```
CMakeLists.txt
PathPlanning.cpp
PathPlanning.ui
PathPlanningWindow.cpp
PathPlanningWindow.h
Planner/
|- AStarPlanner.cpp
|- AStarPlanner.h
|- RRTPlanner.cpp   ← In dieser Datei arbeiten wir
|- RRTPlanner.h
|- …
build/
|- PathPlanning     ← Hier liegt gebaute ausführbare Datei
…
```

Aufgabe 3: Implementierung von RRT*

- Aufbau des Projekts:

```
CMakeLists.txt
PathPlanning.cpp
PathPlanning.ui
PathPlanningWindow.cpp
PathPlanningWindow.h
Planner/
|- AStarPlanner.cpp
|- AStarPlanner.h
|- RRTPlanner.cpp   ← In dieser Datei arbeiten wir
|- RRTPlanner.h
|- …
build/
|- PathPlanning     ← Hier liegt gebaute ausführbare Datei
…
```

Rapidly-exploring Random Trees (RRTs)

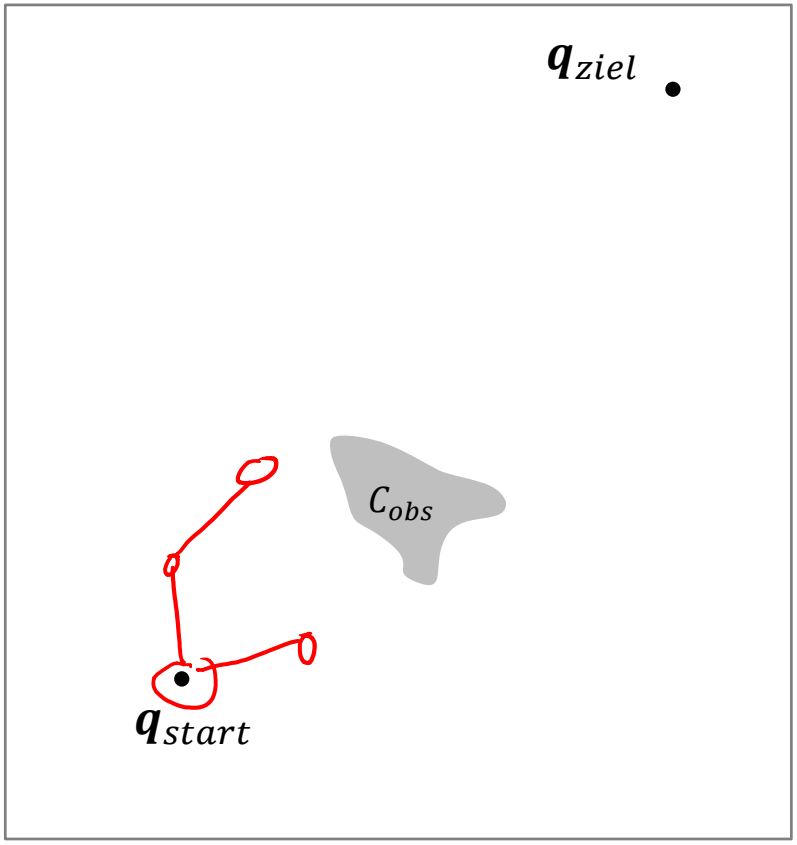
- Im Gegensatz zu PRMs
  - Algorithmus zur Einmalanfrage
  - Keine Vorverarbeitung nötig
  - Keine Probleme mit sich verändernden Umgebungen / kinematischen Ketten
- Probabilistisch vollständiger, randomisierter Algorithmus
  - Keine Garantie, dass eine Lösung innerhalb eines Zeitlimits gefunden wird
  - Wenn eine Lösung existiert, wird sie gefunden (Laufzeit geht gegen Unendlich)
  - Terminiert nicht, wenn keine Lösung existiert
- Effizient für hochdimensionale Problemstellungen
- Erweiterungen der klassischen RRT für spezifische Problemstellungen
  z.B. enge Durchgänge

RRT: Prinzip I

- Die Form von C_{obs} im Konfigurationsraum ist unbekannt

- Initialisierung des RRT
  - Erzeuge leeren Baum T
  - Füge $\boldsymbol{q}_{start}$ in T ein

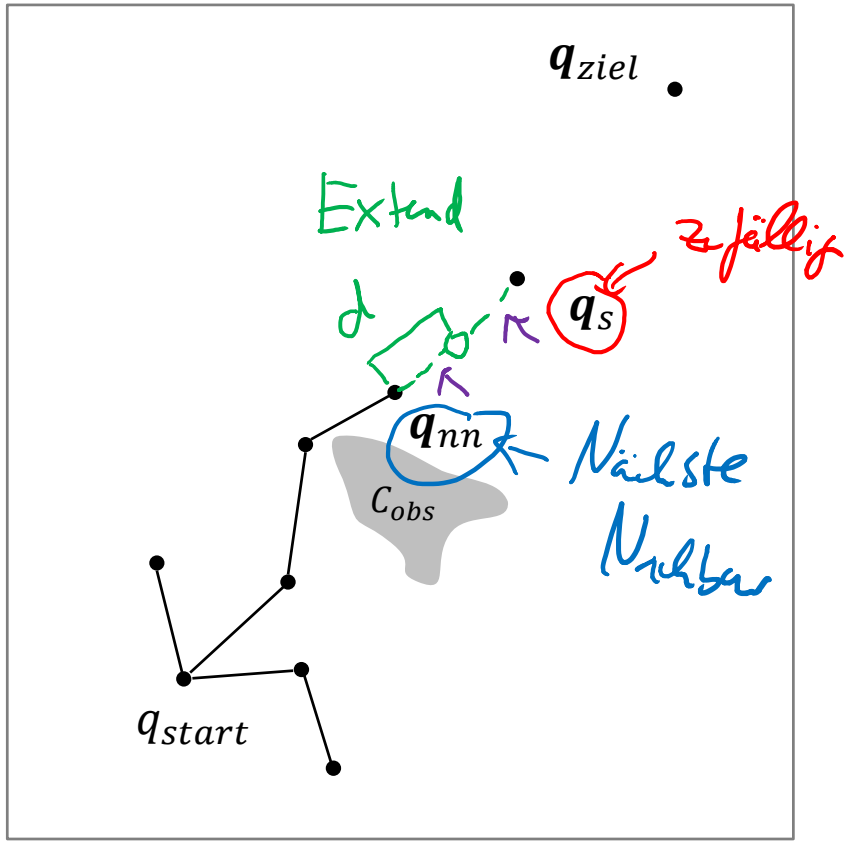
Beispiel mit 2D Konfigurationsraum



RRT: Prinzip II

- Iteration
  1. Erzeuge einen zufälligen Punkt $\boldsymbol{q}_s$
  2. Bestimme den nächsten Nachbarn $\boldsymbol{q}_{nn}$ in T
  3. Füge Punkte auf der Verbindung zwischen $\boldsymbol{q}_s$ und $\boldsymbol{q}_{nn}$ in T ein
     - Mit der Schrittweite d
     - Prüfe jeden der Teilpfade auf Kollision mit C_{obs}.
     - Stoppe, wenn eine Kollision erkannt wurde.
  4. Gehe zu 1.

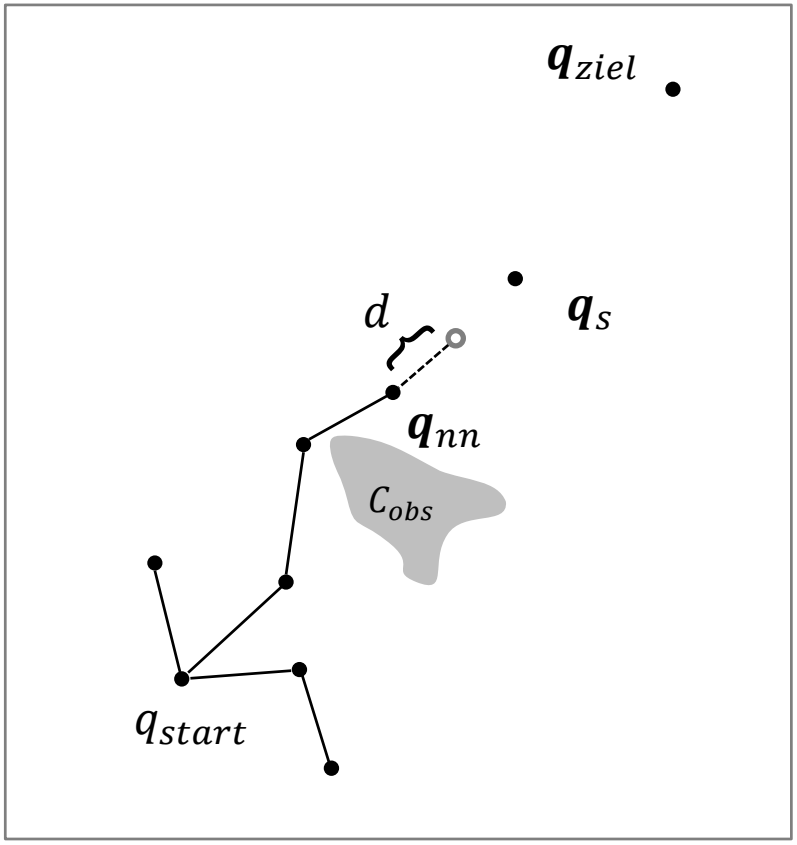
Beispiel mit 2D Konfigurationsraum



RRT: Prinzip II

- Iteration
  1. Erzeuge einen zufälligen Punkt $\boldsymbol{q}_s$
  2. Bestimme den nächsten Nachbarn $\boldsymbol{q}_{nn}$ in T
  3. Füge Punkte auf der Verbindung zwischen $\boldsymbol{q}_s$ und $\boldsymbol{q}_{nn}$ in T ein
     - Mit der Schrittweite d
     - Prüfe jeden der Teilpfade auf Kollision mit C_{obs}.
     - Stoppe, wenn eine Kollision erkannt wurde.
  4. Gehe zu 1.

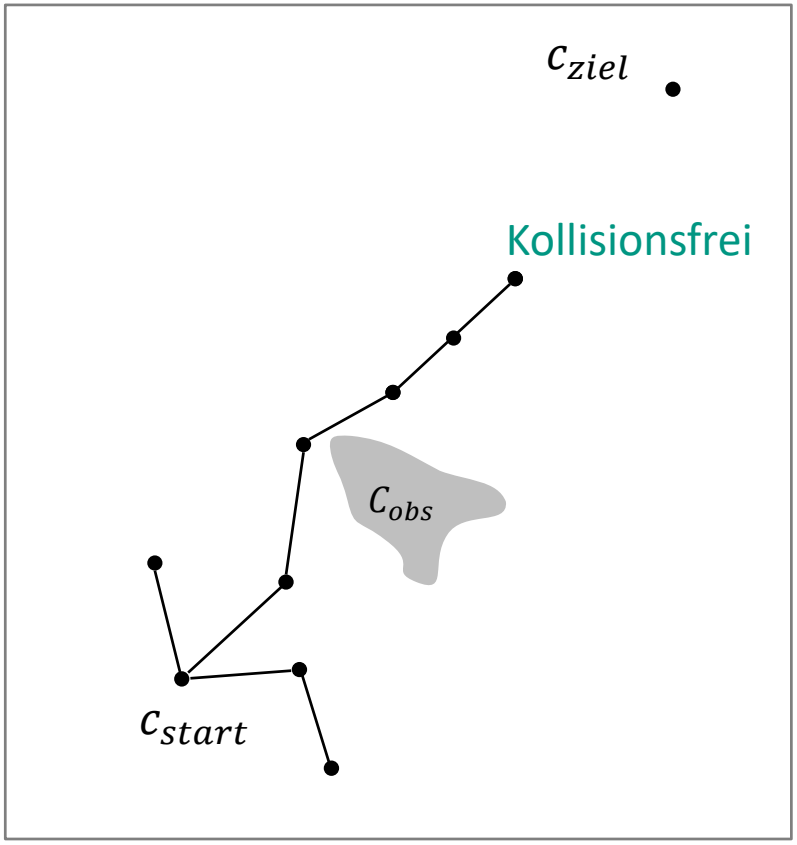
Beispiel mit 2D Konfigurationsraum



RRT: Prinzip III

- Iteration
  1. Erzeuge einen zufälligen Punkt $\boldsymbol{q}_s$
  2. Bestimme den nächsten Nachbarn $\boldsymbol{q}_{nn}$ in T
  3. Füge Punkte auf der Verbindung zwischen $\boldsymbol{q}_s$ und $\boldsymbol{q}_{nn}$ in T ein
     - Mit der Schrittweite d
     - Prüfe jeden der Teilpfade auf Kollision mit C_{obs}. Stoppe, wenn eine Kollision erkannt wurde.
  4. Gehe zu 1.

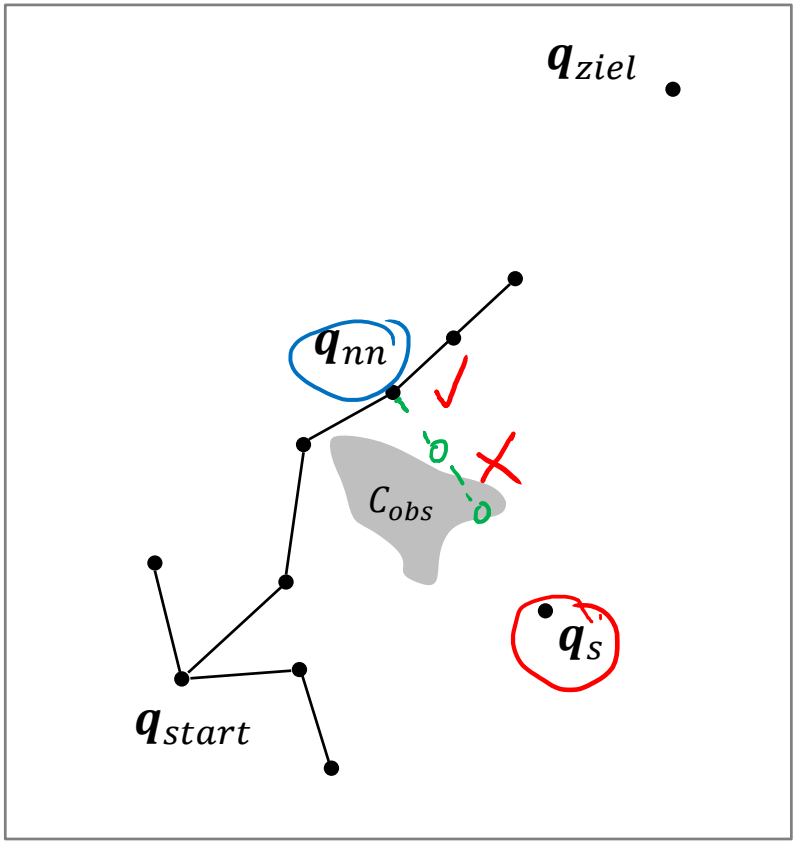
Beispiel mit 2D Konfigurationsraum



RRT: Prinzip IV

- Iteration
  1. Erzeuge einen zufälligen Punkt $\boldsymbol{q}_s$
  2. Bestimme den nächsten Nachbarn $\boldsymbol{q}_{nn}$ in T
  3. Füge Punkte auf der Verbindung zwischen $\boldsymbol{q}_s$ und $\boldsymbol{q}_{nn}$ in T ein
     - Mit der Schrittweite d
     - Prüfe jeden der Teilpfade auf Kollision mit C_{obs}. Stoppe, wenn eine Kollision erkannt wurde.
  4. Gehe zu 1.

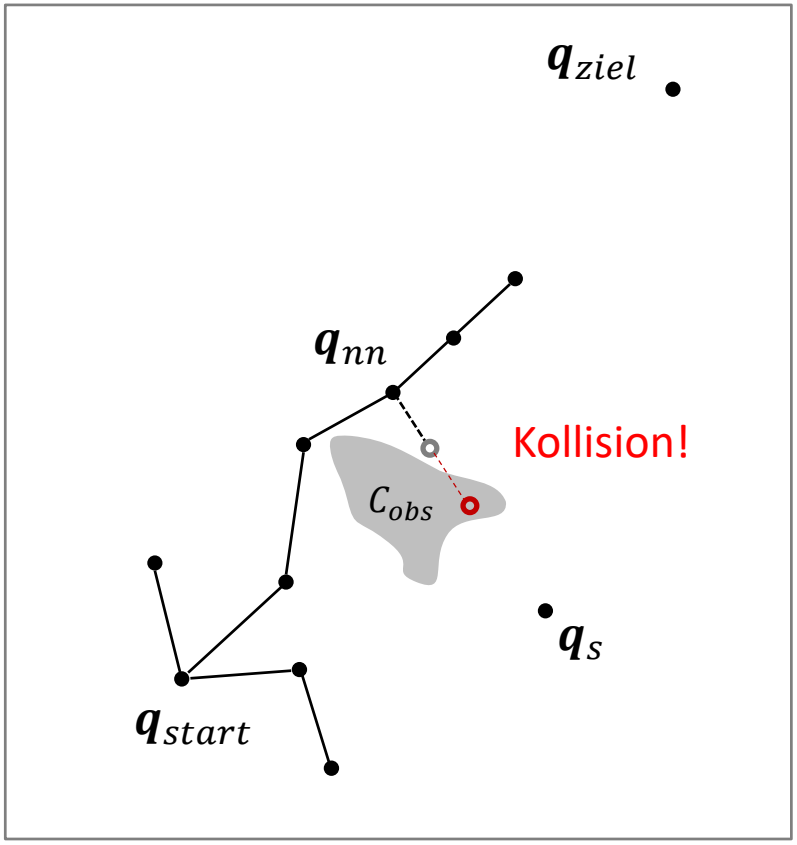
Beispiel mit 2D Konfigurationsraum



RRT: Prinzip IV

- Iteration
  1. Erzeuge einen zufälligen Punkt $\boldsymbol{q}_s$
  2. Bestimme den nächsten Nachbarn $\boldsymbol{q}_{nn}$ in T
  3. Füge Punkte auf der Verbindung zwischen $\boldsymbol{q}_s$ und $\boldsymbol{q}_{nn}$ in T ein
     - Mit der Schrittweite d
     - Prüfe jeden der Teilpfade auf Kollision mit C_{obs}. Stoppe, wenn eine Kollision erkannt wurde.
  4. Gehe zu 1.

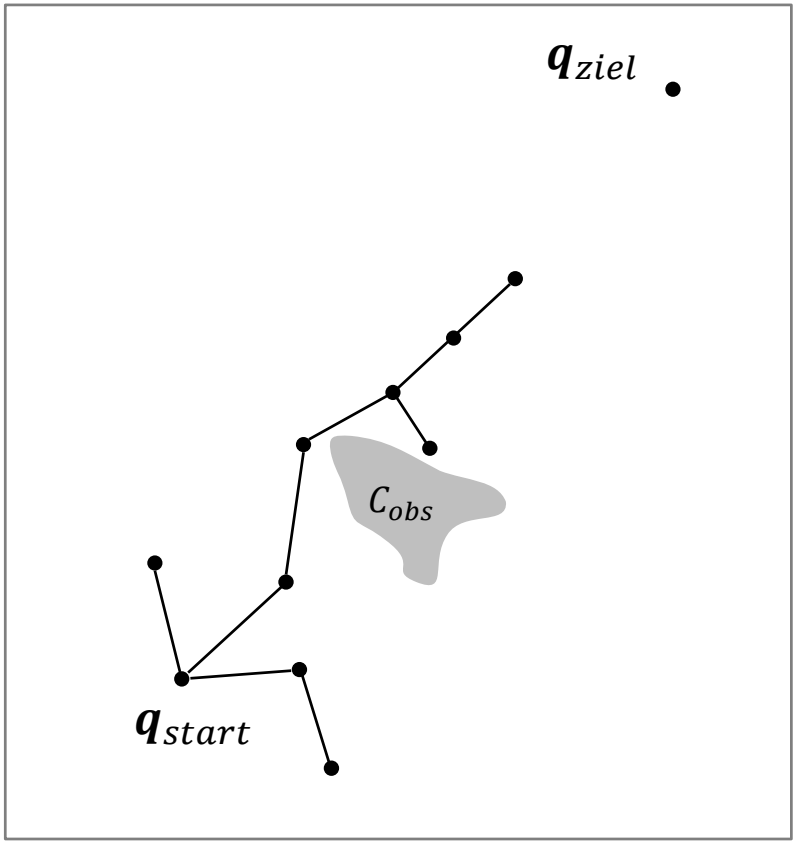
Beispiel mit 2D Konfigurationsraum



RRT: Prinzip V

- Iteration
  1. Erzeuge einen zufälligen Punkt $\boldsymbol{q}_s$
  2. Bestimme den nächsten Nachbarn $\boldsymbol{q}_{nn}$ in T
  3. Füge Punkte auf der Verbindung zwischen $\boldsymbol{q}_s$ und $\boldsymbol{q}_{nn}$ in T ein
     - Mit der Schrittweite d
     - Prüfe jeden der Teilpfade auf Kollision mit C_{obs}. Stoppe, wenn eine Kollision erkannt wurde.
  4. Gehe zu 1.

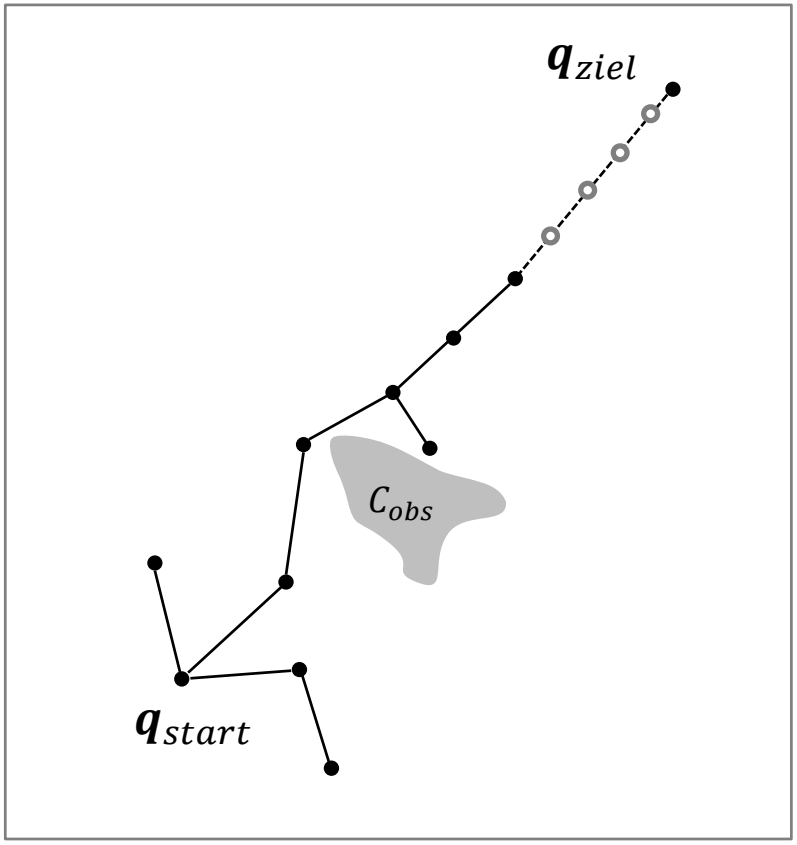
Beispiel mit 2D Konfigurationsraum



RRT: Prinzip VI

- Iteration
  1. Erzeuge einen zufälligen Punkt $\boldsymbol{q}_s$
  2. Bestimme den nächsten Nachbarn $\boldsymbol{q}_{nn}$ in T
  3. Füge Punkte auf der Verbindung zwischen $\boldsymbol{q}_s$ und $\boldsymbol{q}_{nn}$ in T ein
     - Mit der Schrittweite d
     - Prüfe jeden der Teilpfade auf Kollision mit C_{obs}. Stoppe, wenn eine Kollision erkannt wurde.
  4. Gehe zu 1.
- Prüfe in jedem k-ten Schritt, ob $\boldsymbol{q}_{ziel}$ mit T verbunden werden kann

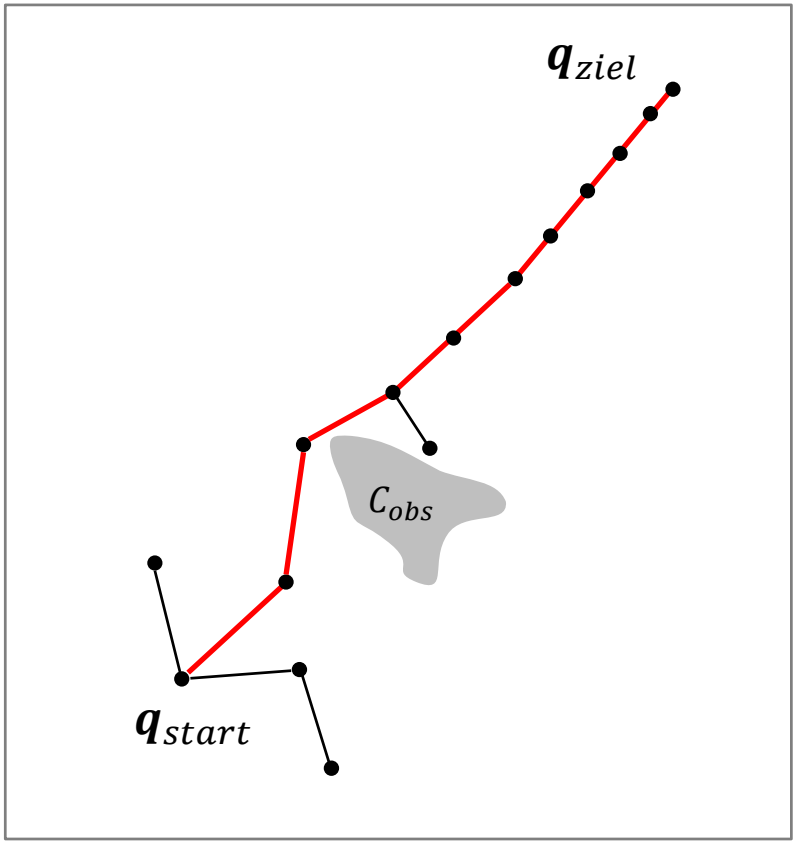
Beispiel mit 2D Konfigurationsraum



RRT: Prinzip VII

- Iteration
  1. Erzeuge einen zufälligen Punkt $\boldsymbol{q}_s$
  2. Bestimme den nächsten Nachbarn $\boldsymbol{q}_{nn}$ in T
  3. Füge Punkte auf der Verbindung zwischen $\boldsymbol{q}_s$ und $\boldsymbol{q}_{nn}$ in T ein
     - Mit der Schrittweite d
     - Prüfe jeden der Teilpfade auf Kollision mit C_{obs}. Stoppe, wenn eine Kollision erkannt wurde.
  4. Gehe zu 1.
- Prüfe in jedem k-ten Schritt, ob $\boldsymbol{q}_{ziel}$ mit T verbunden werden kann

Beispiel mit 2D Konfigurationsraum



Lösung gefunden