HOCHSCHULE KONSTANZ
TECHNIK, WIRTSCHAFT UND GESTALTUNG

Autonome Roboter (SS2015)

Prof. Dr. Oliver Bittel

Präsentation

Daniel Eckstein (290668)

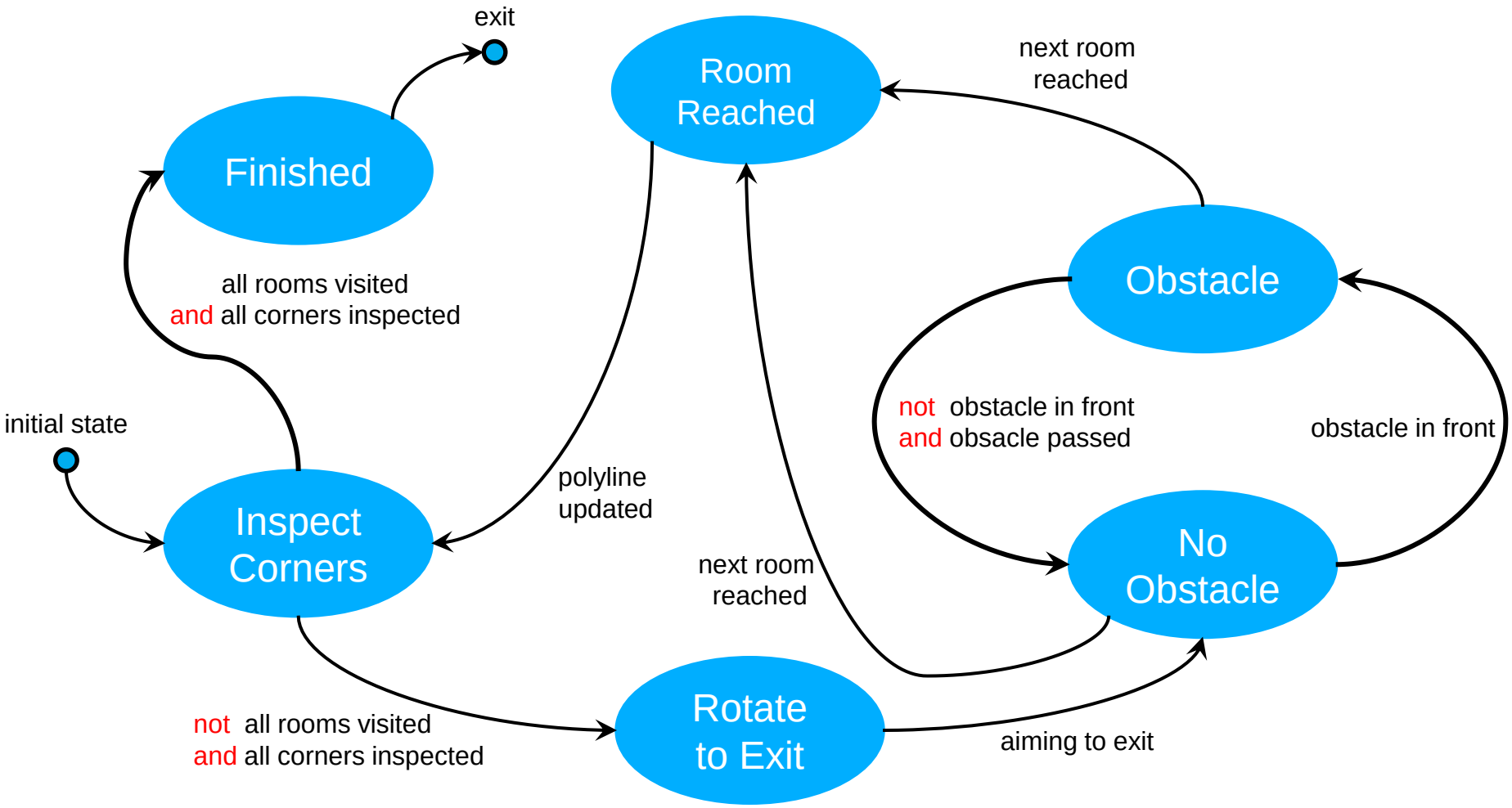
Philipp Lohrer ()

21.07.2015

Inhalt

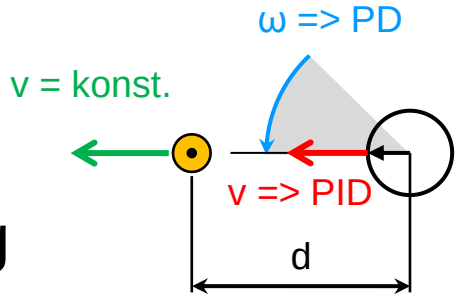
- Text

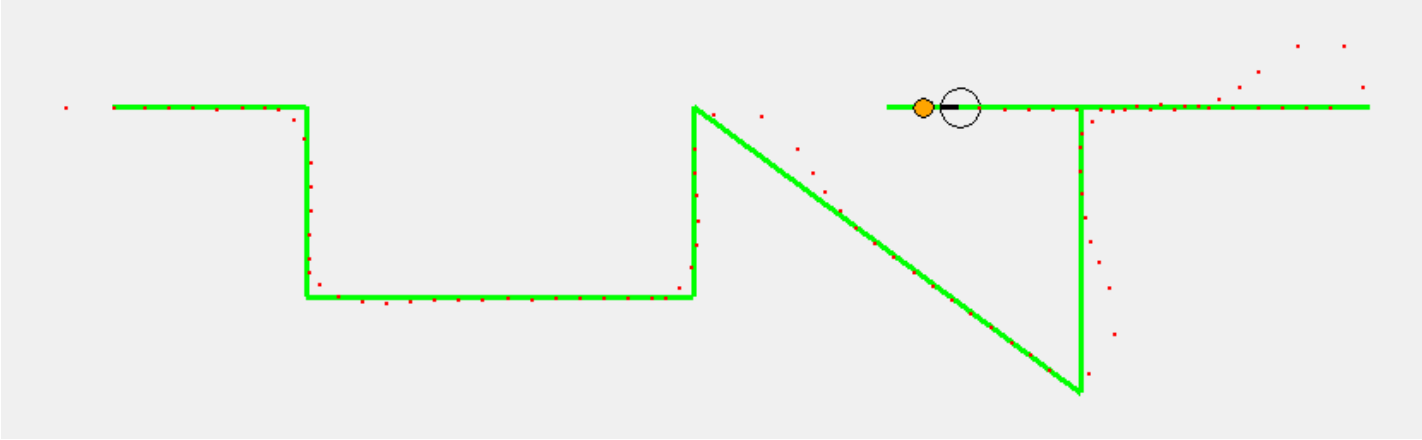
Zustandsautomat



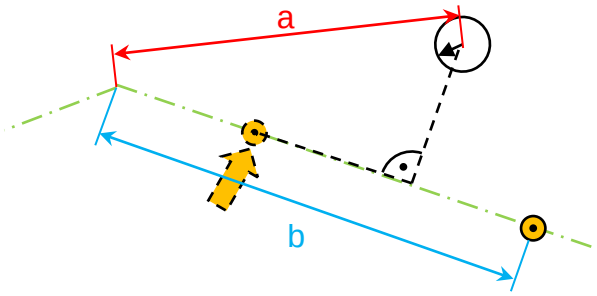
Carrot-Donkey-Verfahren (1)

- Carrot in jedem Zustand Aktiv
- Robot verfolgt Carrot durch Regelung
- Situationsabhängige Spezialfunktionen

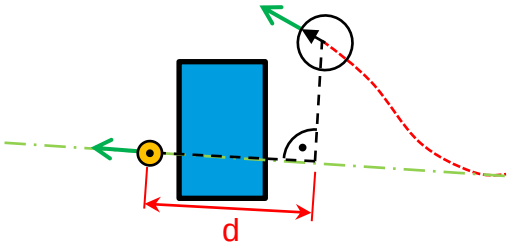




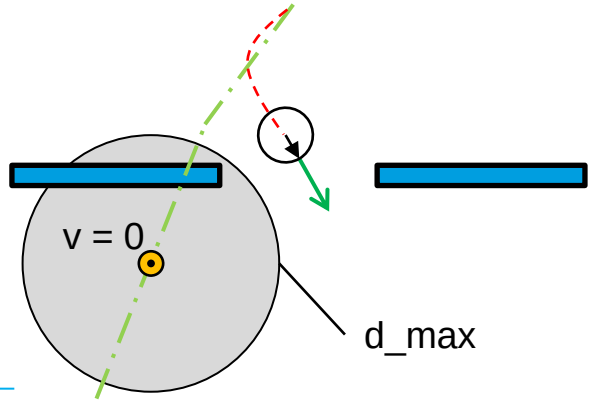
Carrot-Donkey-Verfahren (2)



- Wenn Robot näher am Ziel als Carrot (a < b)
  - Schiebe Carrot auf Polyline vor den Robot



- Wenn Robot einem Hindernis ausweicht
  - Bewege Carrot mit Abstand d mit
- Carrot dient als Zielrichtung für Hindernisvermeidung



- Wenn sich Robot zu weit entfernt
  - Warte ab (Stoppe Carrot)

A*-Algorithmus (1)

```
def dijkstra_algorithm(start_point, end_point, adjacency=8)


    open_list.push(start_point)

    while open_list.not_empty():
        v_priority, v_point, d_v, p_v = open_list.pop()
        closed_list[v_point_index] = p_v

        if v_point == end_point:
            polyline = create_polyline(v_point, closed_list)

        for neighbour in get_neighbours(v_point, adjacency):
            w_point_index = self.match_in_grid(w_point)

            if closed_list[w_point_index] is None:
                d_w = d_v + c_v_w
                h_w_z = calc_heuristic(w_point)
                p_w = v_point
                # Scale priority according to occupancy ([0..1]) at grid[point]
                  -> priority * [1.0 .. 2.0]
                priority_w = (d_w + h_w_z) * (w_point_occupancy + 1)

                if w_point not in open_list:
                    open_list.push(w_point, priority_w, d_w, p_w)
                else:
                    if d_w < open_list.get_old_dist(w_point):
                        open_list.update_entry(w_point, priority_w, d_w, p_w)
```

Nachbarschaft (4 oder 8)

Prioritätsliste

Closed List wird als Map adressiert

Erzeugung der Polyline durch Pfadrückverfolgung in der Map

A*-Algorithmus (2)

- Kürzeste Wege zwischen den Räumen berechnen
- Open List als heapq:

Kleinste Priorität ↑

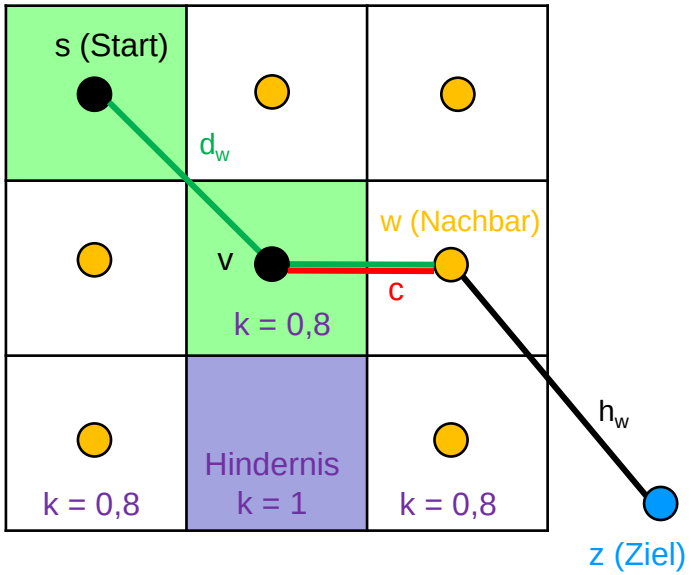
1	Priorität	Punkt w [x, y]	Distanz zum Start d_w	Vorgänger p
2	…	…	…	…

- Closed List als 2-D Array:

Index	0	1	2	…
0	→	•		
1		↖	↖	
…		↑	↖	←

- Es wird jeweils der Vorgänger p gespeichert
- Schneller Zugriff durch Indexing

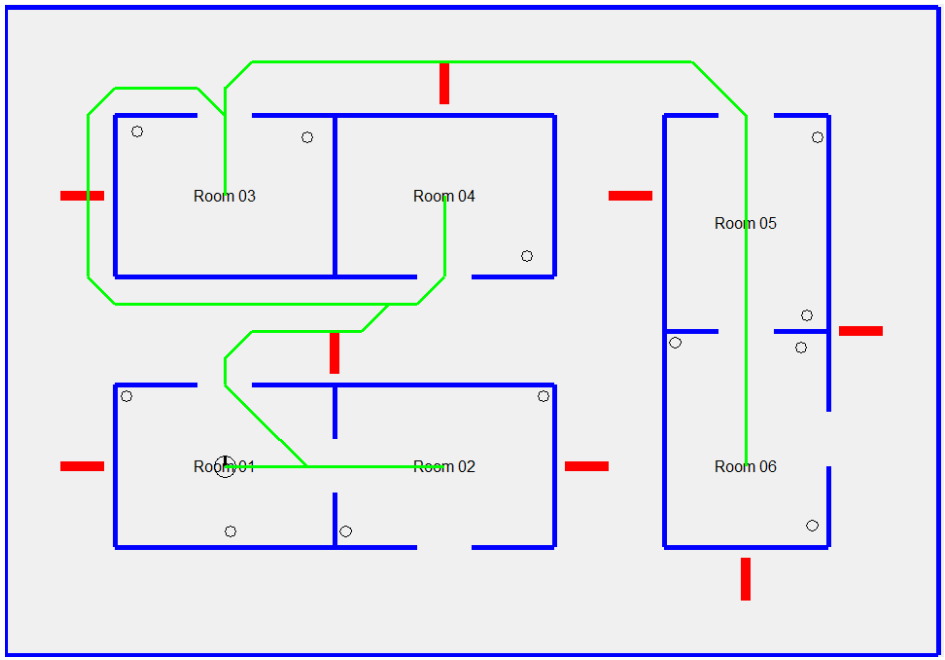
- Heuristik $h_w = \sqrt{(x_z - x_w)^2 + (y_z - y_w)^2}$
- Kosten $c = \sqrt{(x_w - x_v)^2 + (y_w - y_v)^2}$
- Belegtheitswert k durch Brushfire
- Priorität $= (d_w + h) \cdot (k + 1)$



Path Scheduler

- Ermittelt den kürzesten Weg durch alle Räume
- Wege von Raum zu Raum mit A*
- Bei **n** Räumen gilt:
  - Mögliche Kombinationen durch Permutation = (n-1)!
    - (6-1)! = 120
  - Anzahl A* Iterationen:
    - n · (n-1) / 2 = 15

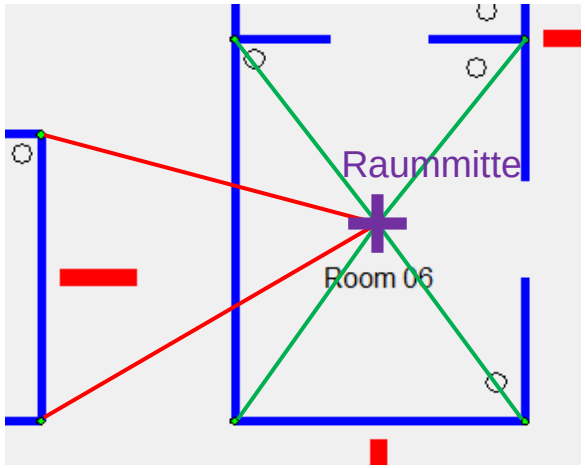
Raum	1	2	3	4	5	6
1	-	A*	A*	A*	A*	A*
2	inv	-	A*	A*	A*	A*
3	inv	inv	-	A*	A*	A*
4	inv	inv	inv	-	A*	A*
5	inv	inv	inv	inv	-	A*
6	inv	inv	inv	inv	inv	-

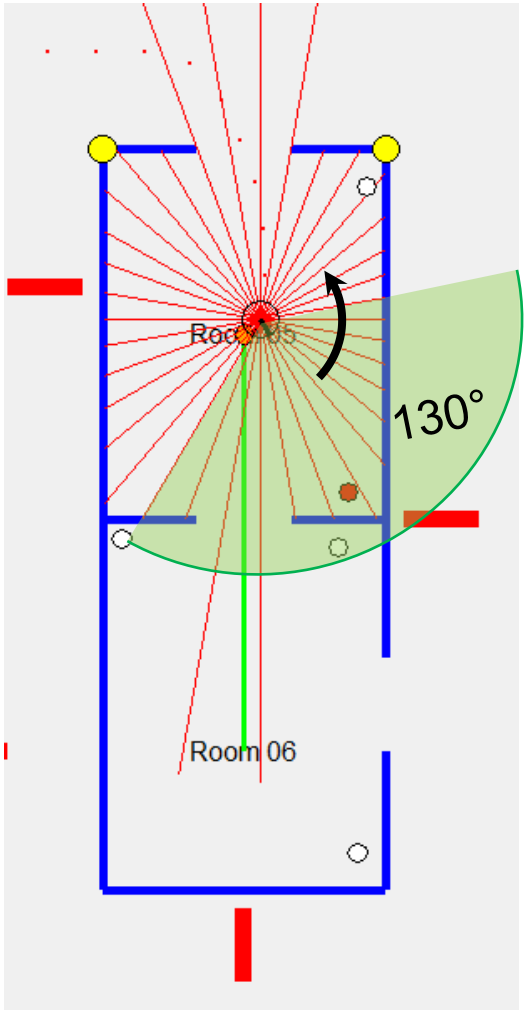


Room Scanner

- Nutzt das Belegtheitsgitter um alle Ecken zu finden
- Ordnet jedem Raum die Ecken zu
- Prüft, ob Robot in aktuellem Raum alle Ecken gesehen hat

0	0	0	0	0	0	0	0	0
1	1	1	1	0	0	1	1	0
0	1	0	0	0	0	0	1	0
0	1	0	0	0	0	0	1	0
0	1	0	0	0	0	0	1	0
1	1	1	1	1	0	1	1	0
0	0	0	0	0	0	0	0	0



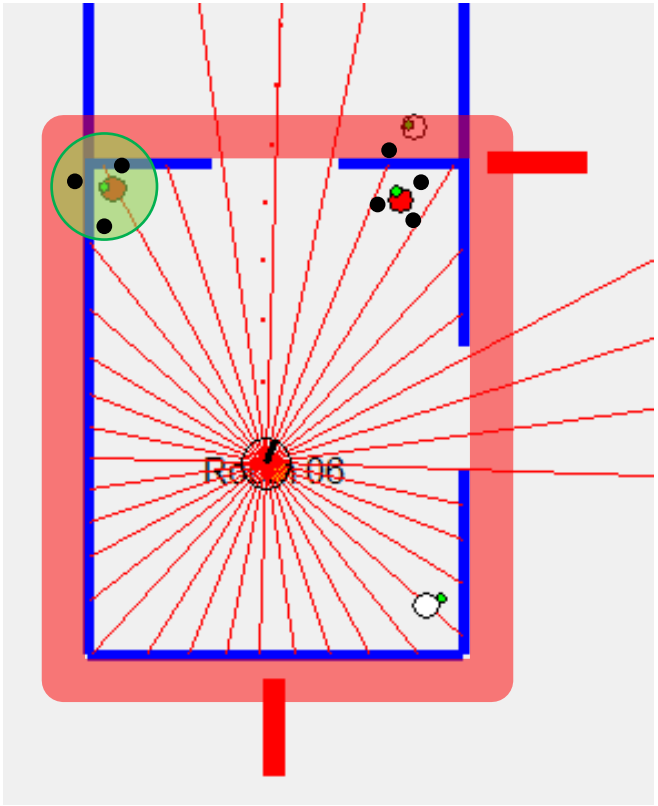


Bereich Eckenerkennung

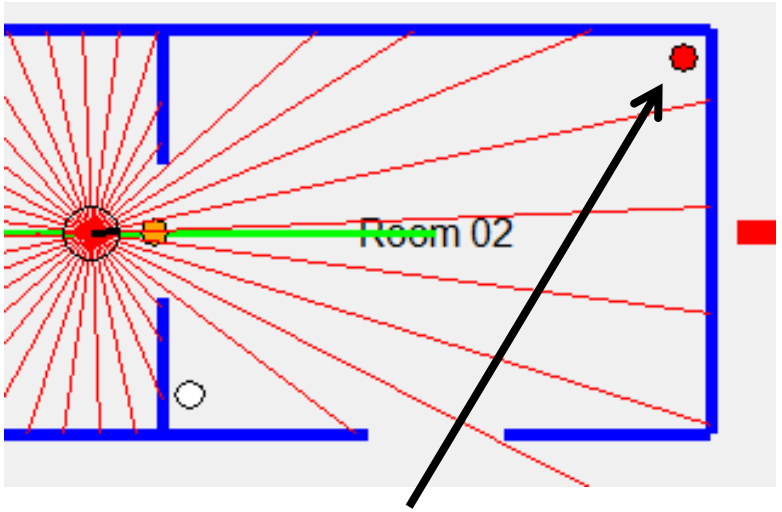
Nicht gesehene Ecken

Box Locator

- Scannt beim betreten eines Raumes nach Boxen
- Gruppiert Messpunkte



- Geschätzte Positionen der Boxen
- Wandtoleranz für Messpunkte
- Einzugsbereich für Messpunkte
- Messpunkte



Erkannte Box in anderem Raum:

➢ Messung wird verworfen

4. Zusammenfassung

- Text

Herzlichen Dank
für Ihre Aufmerksamkeit