03-05-H
-709.53

Echtzeitbildverarbeitung (5)

Prof. Dr. Udo Frese

k-D Baum für Farbsegmentierung

Farbsegmentierung durch Look-up-tables

Applikation: RoboCup

Was bisher geschah

- **Szenarien können viel einfacher aussehen, als sie sind!**
- **Differenzbilder zur Bewegungserkennung**
  - Differenz zu gleitend nachgeführtes Referenzbild
- **Farbe**
  - physikalisch das Lichtspektrum (Energie vs. Wellenlänge)
  - Objektfarbe abhängig von Beleuchtung und Betrachtungswinkel
  - 3 menschliche Farbrezeptoren ⇒ Farbeindrücke kombinierbar aus Rot, Grün, Blau
  - Im Rechner als RGB32: 1 Byte Rot, 1 Byte Grün, 1 Byte Blau, 1 Byte frei
  - Farbkameras nutzen Mosaik - Farbfilter (Bayer Filter) vor dem CCD Chip
- **Farbsegmentierung**
  - Klassifikation (Rot, Grün, Blau)→ Klasse
  - Handsegmentierte Bilder für große Menge an Trainingsvektoren
  - m-nearest Neighbour Klassifikator: Liefert die Klasse, die der Mehrzahl der m nächstgelegenen Trainingsvektoren entsprechen

k-D Baum für Farbsegmentierung

m-Nearest Neighbour Klassifikation

- **Für Testvektor *x* suche *m* nächsten Trainingsvektoren.**
- **Wurde Mehrzahl (>*m*/2) zur selben Klasse zugeordnet?**
  - Ja: Klasse ist Ergebnis für x
  - Sonst: weise *x* zurück
- **Ist Entfernung über einem Schwellwert: weise *x* zurück**
- **Übliche Werte für *m*: 1..13**
- **Vor- & Nachteile**
  - einfach zu implementieren
  - konvergiert gegen den optimalen Klassifikator bei unendlich vielen Trainingsdaten
  - braucht meist viele Trainingsdaten
  - muss alle Trainingsdaten speichern

k-D Baum für Farbsegmentierung

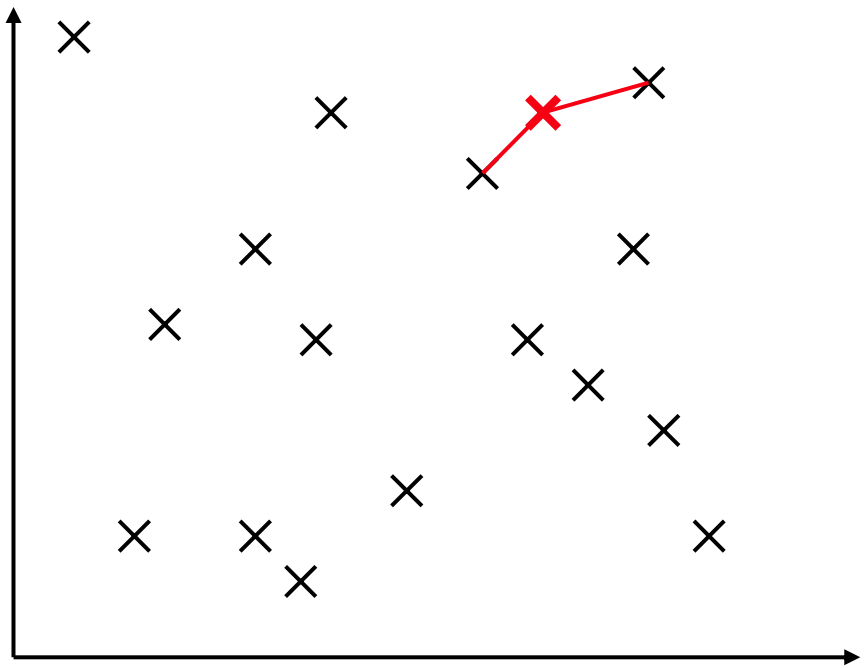
m-Nearest Neighbour Klassifikation

- **Für Testvektor *x* suche *m* nächsten Trainingsvektoren.**
- **Wurde Mehrzahl (>*m*/2) zur selben Klasse zugeordnet?**
  - Ja: Klasse ist Ergebnis für x
  - Sonst: weise *x* zurück
- **Ist Entfernung über einem Schwellwert: weise *x* zurück**
- **Übliche Werte für *m*: 1..13**
- **Vor- & Nachteile**
  - einfach zu implementieren
  - konvergiert gegen den optimalen Klassifikator bei unendlich vielen Trainingsdaten
  - braucht meist viele Trainingsdaten
  - muss alle Trainingsdaten speichern

k-D Baum für Farbsegmentierung

m-Nearest Neighbour Klassifikation

- **Beispiel:**
  - k=2D Merkmalsvektoren
  - m=2 nächsten Nachbarn
- **Farbsegmentierung**
  - k=3D Merkmalsvektoren (RGB)
  - m=1..13
- **Einfach, aber langsam: Alle Trainingsvektoren durchlaufen und die m nächsten bestimmen**
- **Effiziente Alternative: k-D Baum**



k-D Baum für Farbsegmentierung

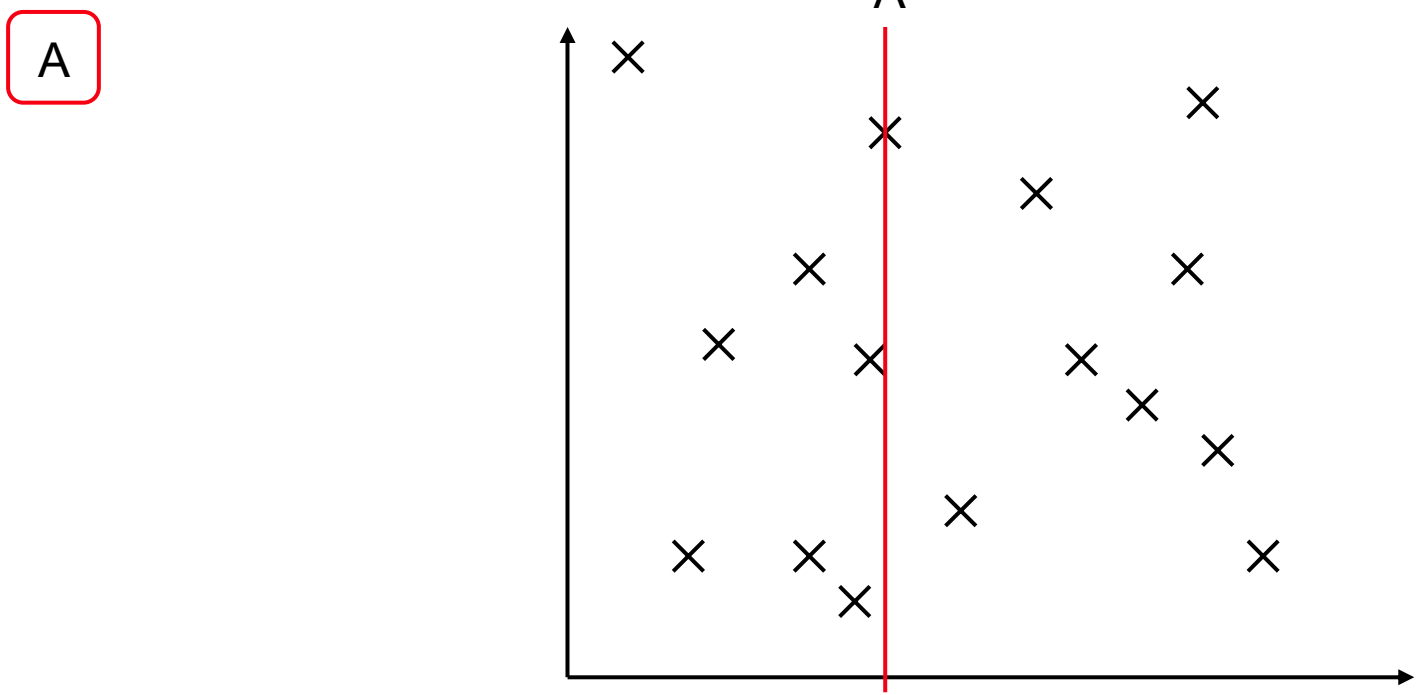
k-D Baum

- **schnelle Suche der nächsten Nachbarn**
- **Generalisierung von (1-D) Bäumen**
- **zwei Parameter**
  - k („k-D Baum“) ist Dimension des Merkmalsvektors
  - m („m-nearest Neighbour“) ist Anzahl der berücksichtigsten Nachbarn
- **Idee: Reihum in den Dimensionen rekursiv halbieren**
- **Aufbau in einem Rutsch**
  - sortieren
  - Median wird Wurzel
  - Rekursion linke/rechte Hälfte

k-D Baum für Farbsegmentierung

k-D Baum: Aufbau

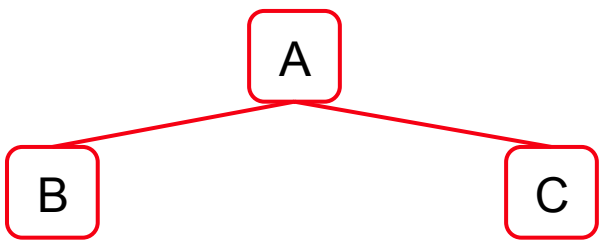
- **Sortieren nach x und Teilen am Median**

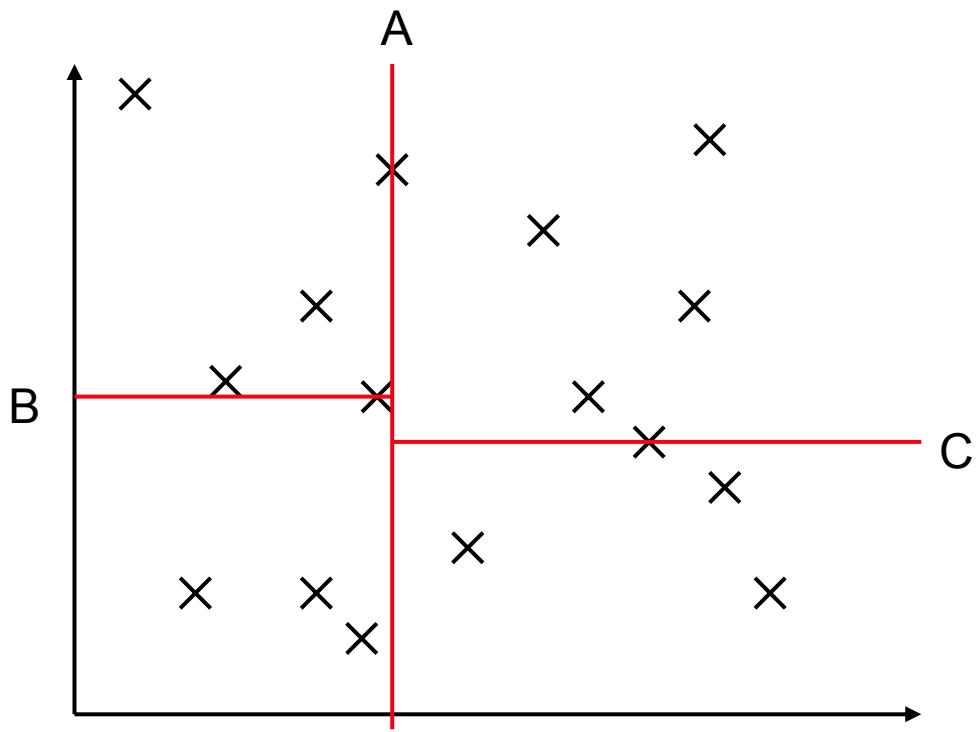


k-D Baum für Farbsegmentierung

k-D Baum: Aufbau

- **Sortieren der Hälften nach y und Teilen am Median**

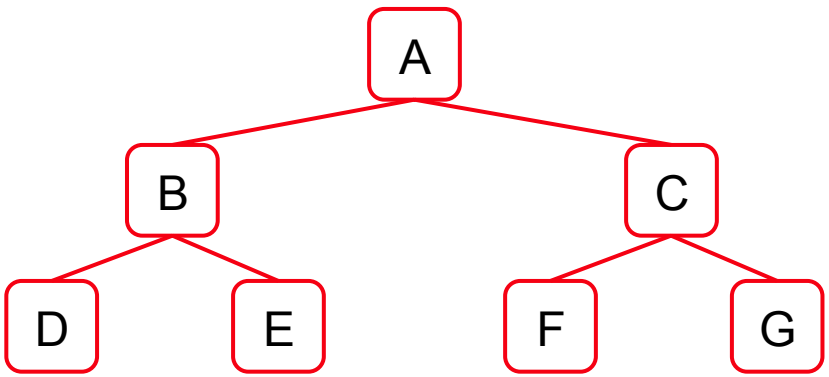


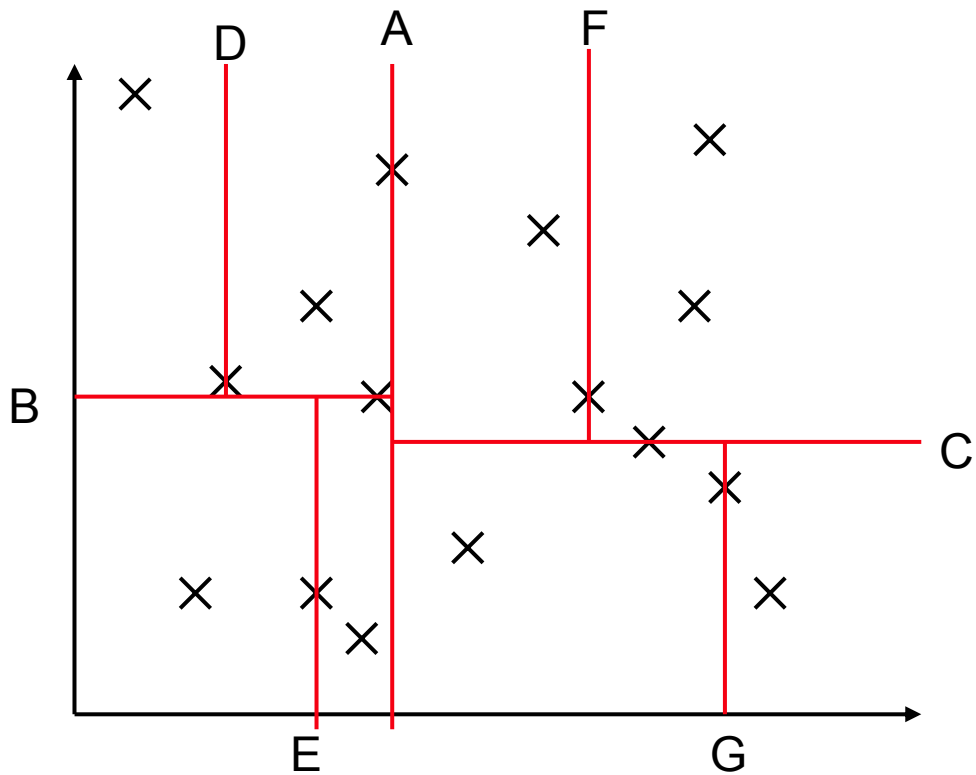


k-D Baum für Farbsegmentierung

k-D Baum: Aufbau

- **Sortieren der Hälften nach x und Teilen am Median**

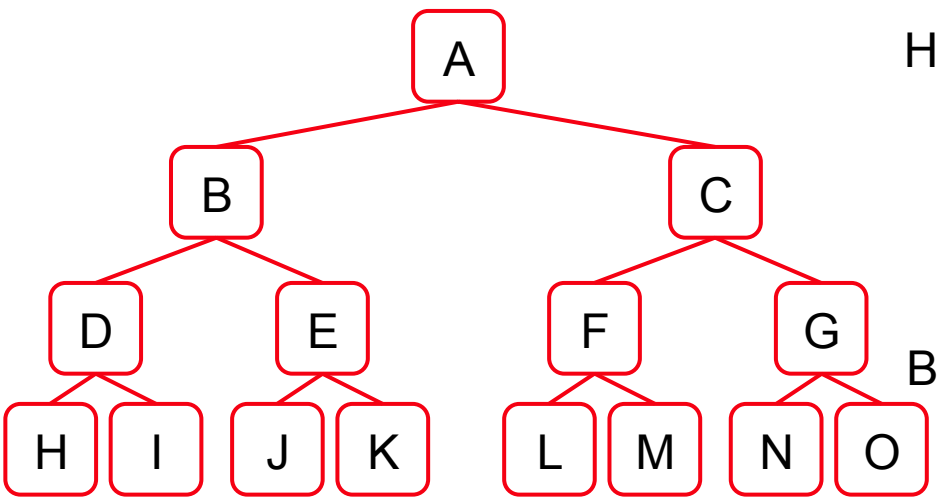


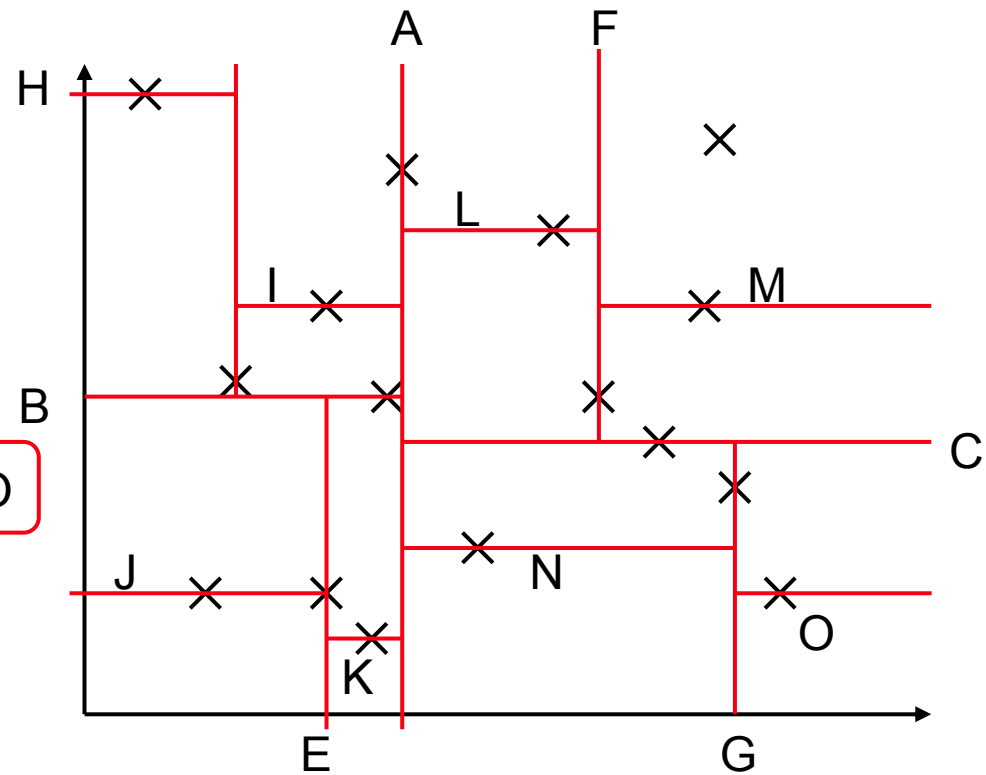


k-D Baum für Farbsegmentierung

k-D Baum: Aufbau

- **Sortieren der Hälften nach y und Teilen am Median**

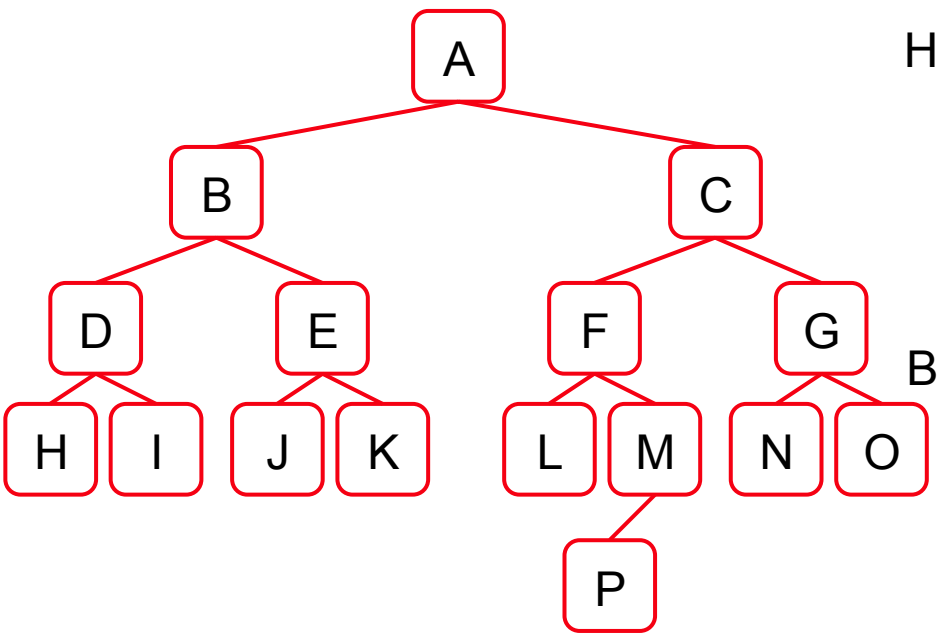


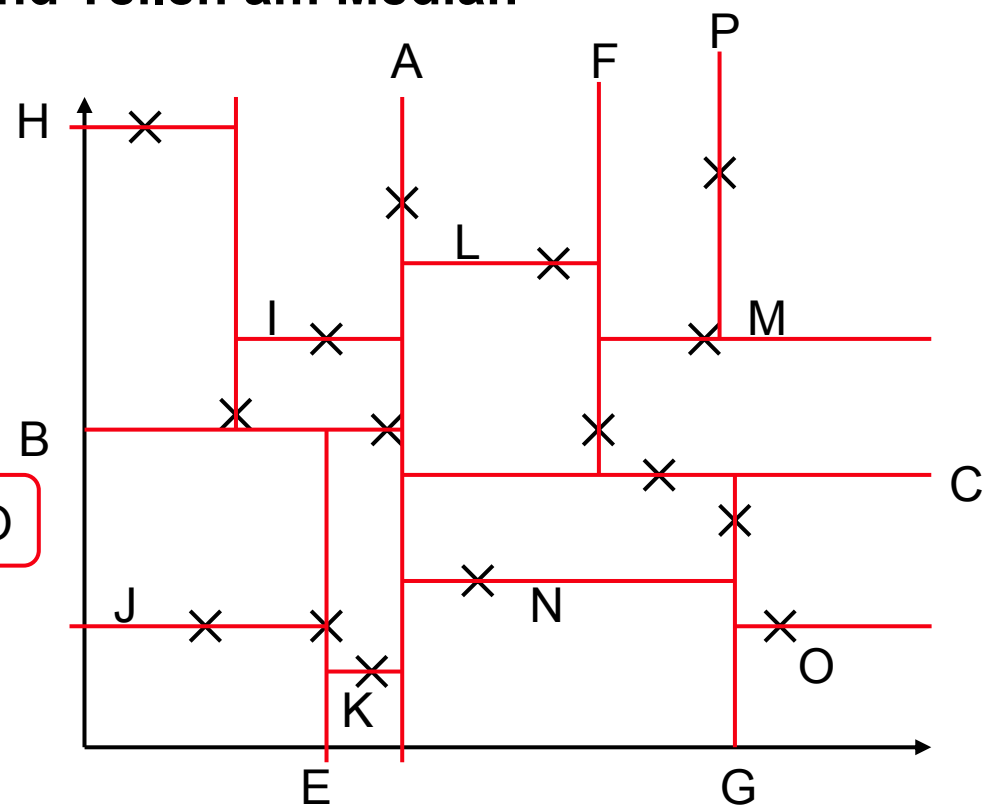


k-D Baum für Farbsegmentierung

k-D Baum: Aufbau

- **Sortieren der Hälften nach x und Teilen am Median**





k-D Baum für Farbsegmentierung

k-D Baum: Implementierung Aufbau

- rekursiv nicht ebenenweise
- sortieren mit Standard Template Library
- nur einmal vorab
- noch schneller möglich, ist aber nicht wichtig.

```
KDNode* buildKDTree (vector<ColorAndIndex>& data, level, lo, hi) {
   if (lo>hi) return NULL;
   sort data[lo..hi] according to component (level%k)
   median = (lo+hi)/2;
   Node* n = new Node(data[median]);
   n->below = buildKDBaum (data, level+1, lo, median-1);
   n->above = buildKDBaum (data, level+1, median+1, hi);
   return n;
}
```

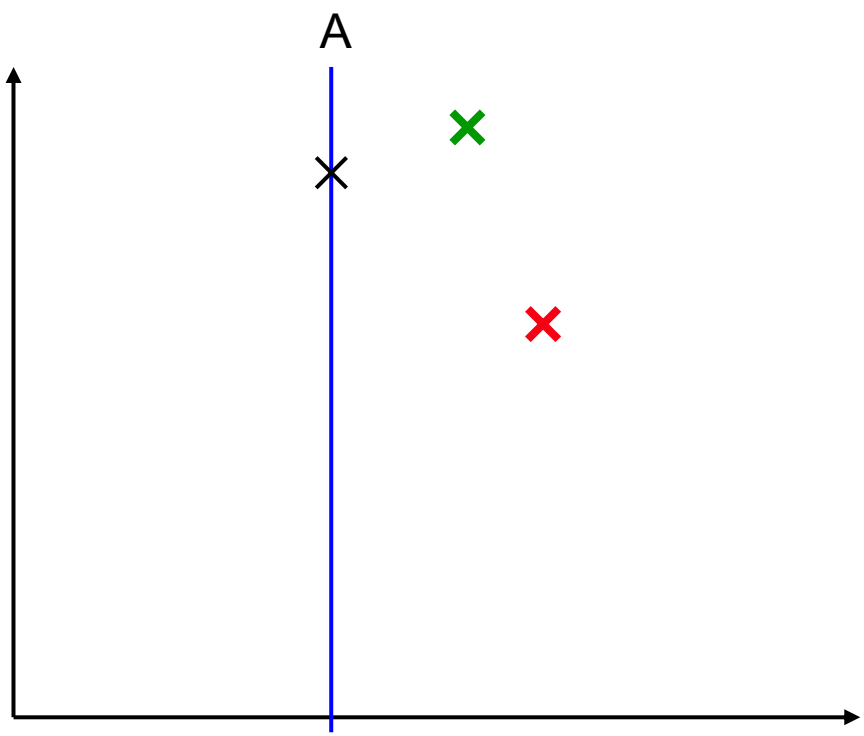
k-D Baum für Farbsegmentierung

k-D Baum: Nächster Nachbar

- **Zuerst nur ein Nachbar**
- **Suche des nächsten Nachbarn**
  - Ist betrachteter Vektor näher? Dann neuer nächster Nachbar.
  - Rekursion nähere Hälfte
  - Rekursion fernere Hälfte, es sei denn…
  - ...Entfernung zur ferneren Hälfte ist größer als zum nächsten bisher gefundenen Nachbarn

k-D Baum für Farbsegmentierung

<u>Frage an das Auditorium</u>: Was heißt „Entfernung zur ferneren Hälfte“?

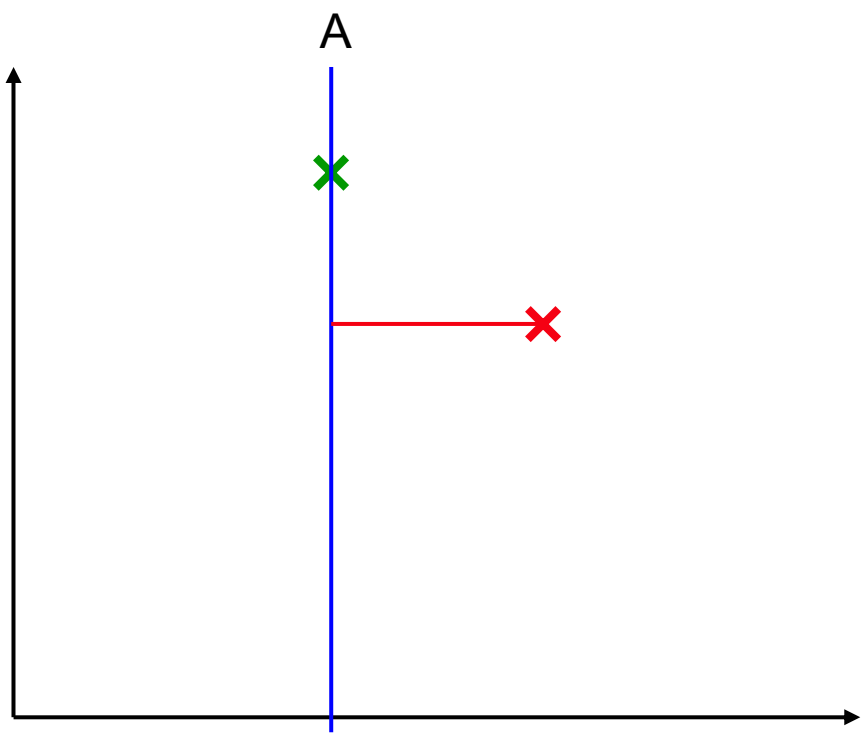


- betrachteter Vektor / im k-D Baum
- Trennlinie bzw. Knoten
- Anfragevektor
- bisher nächster Nachbar

k-D Baum für Farbsegmentierung

<u>Frage an das Auditorium</u>: Was heißt „Entfernung zur ferneren Hälfte“?

- **Länge des Lotes auf die vertikale Trennlinie**
- **Differenz der X-Koordinaten**
- **Nicht: Entfernung zum Vektor im Wurzelknoten**



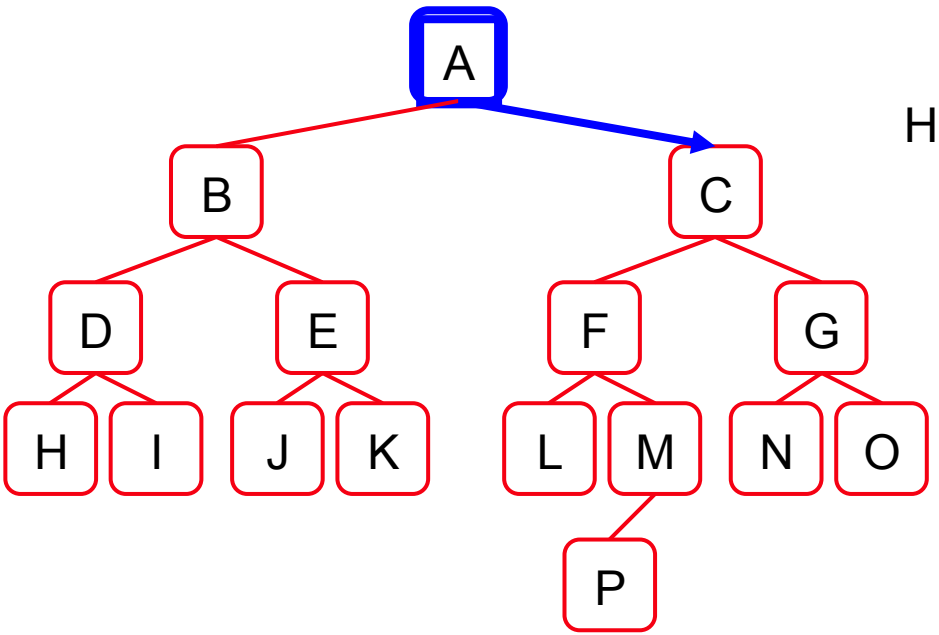
× betrachteter Vektor / im k-D Baum

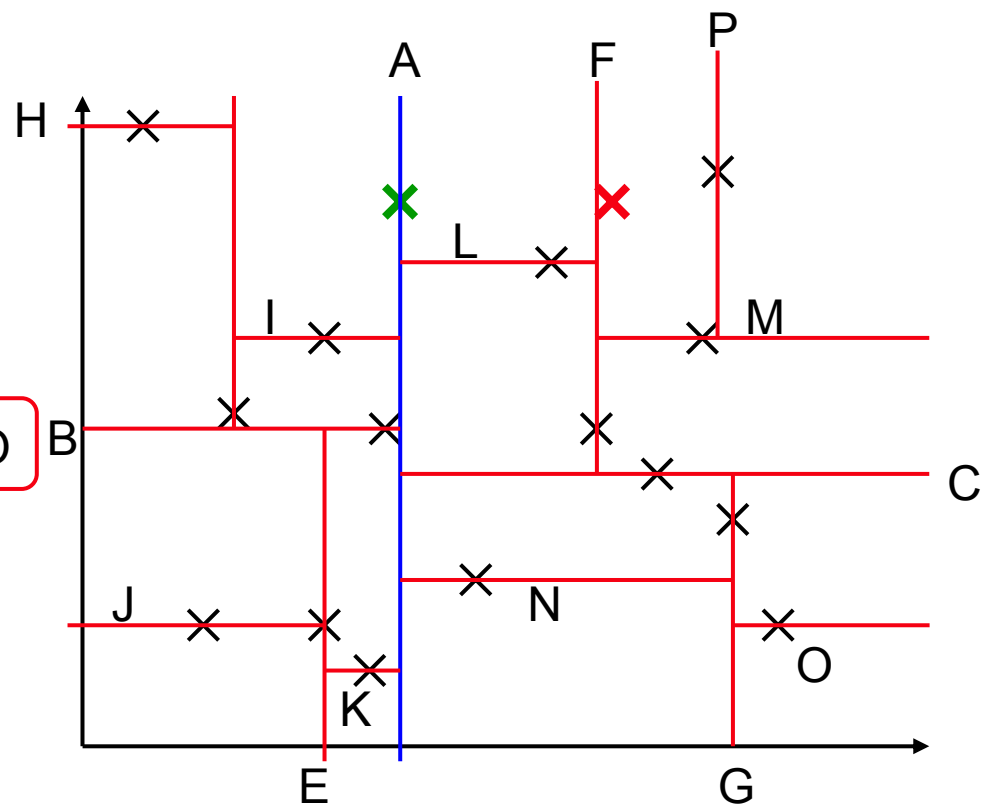
| Trennlinie bzw. Knoten

× Anfragevektor

× bisher nächster Nachbar

k-D Baum für Farbsegmentierung





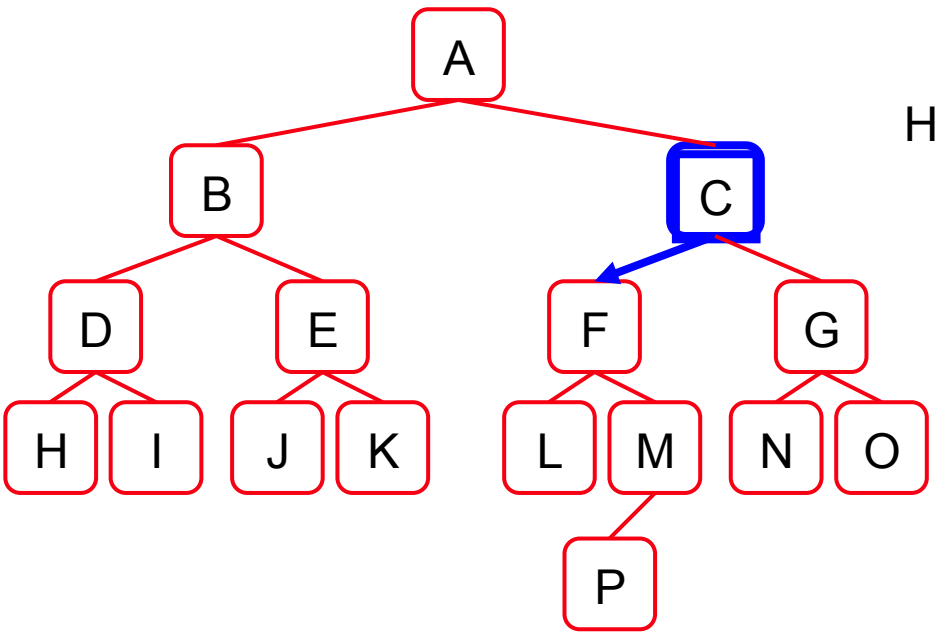
× betrachteter Vektor / im k-D Baum

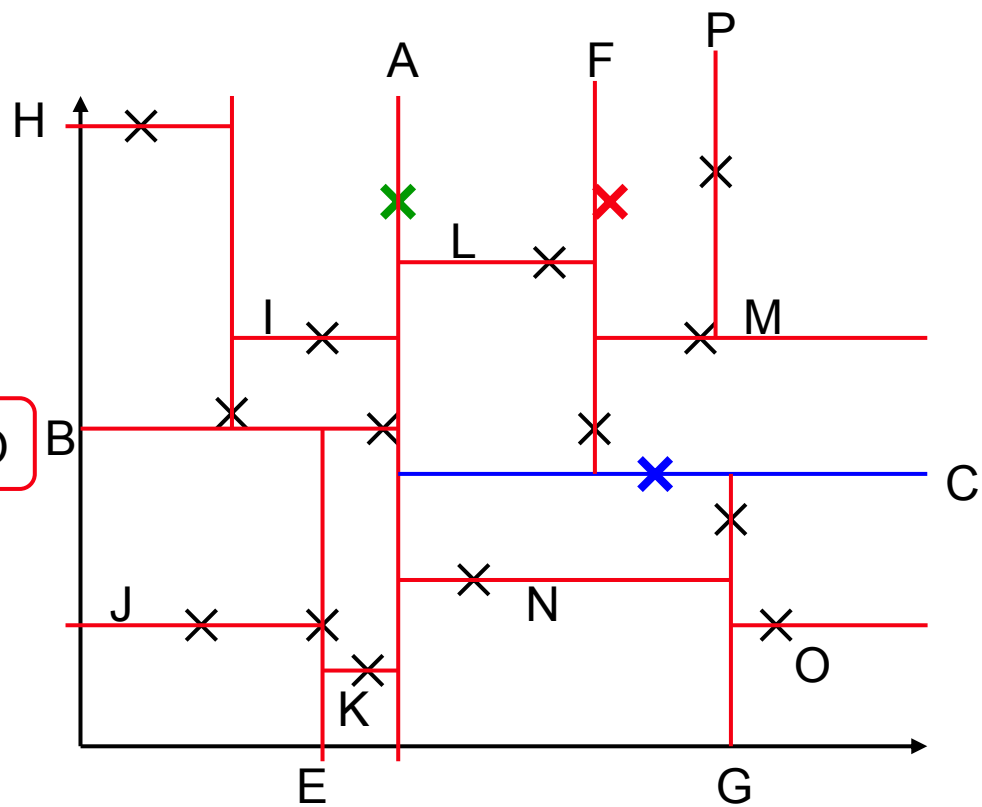
| Trennlinie bzw. Knoten

× Anfragevektor

× bisher nächster Nachbar

k-D Baum für Farbsegmentierung





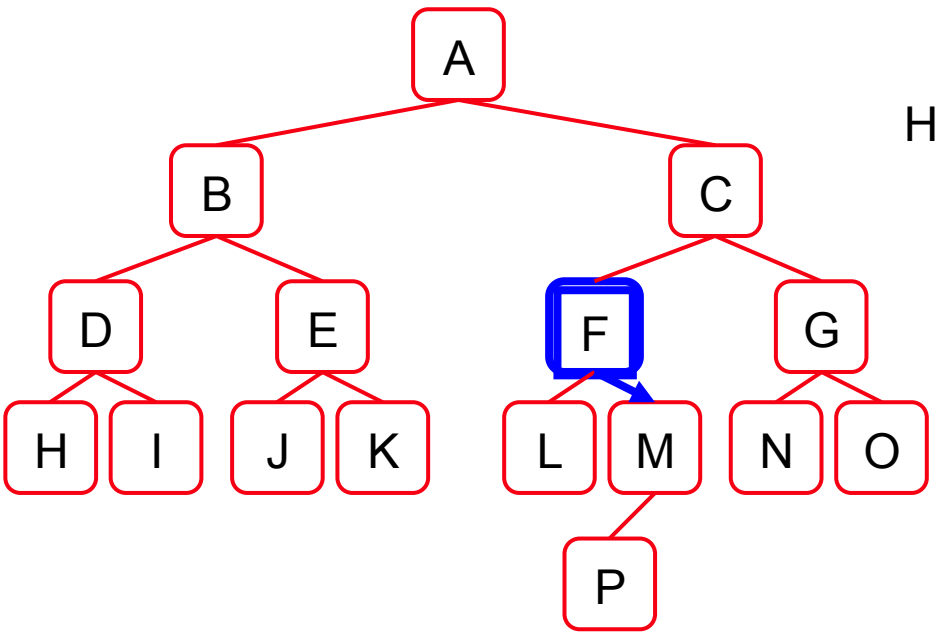
× betrachteter Vektor / im k-D Baum

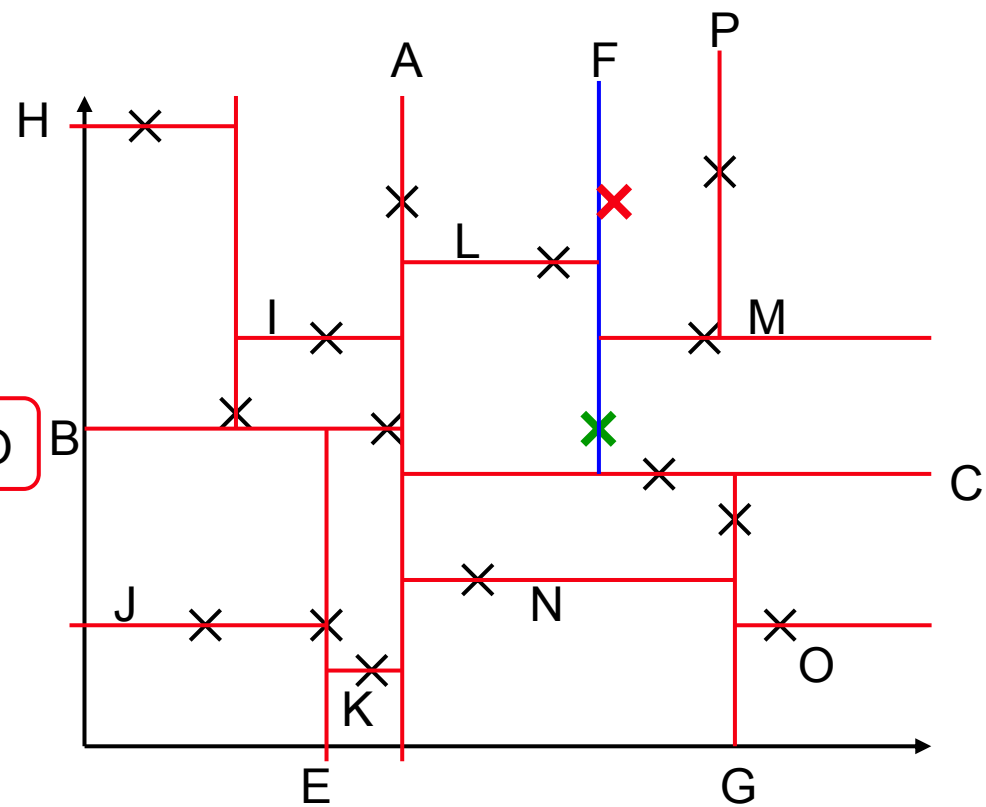
| Trennlinie bzw. Knoten

× Anfragevektor

× bisher nächster Nachbar

k-D Baum für Farbsegmentierung





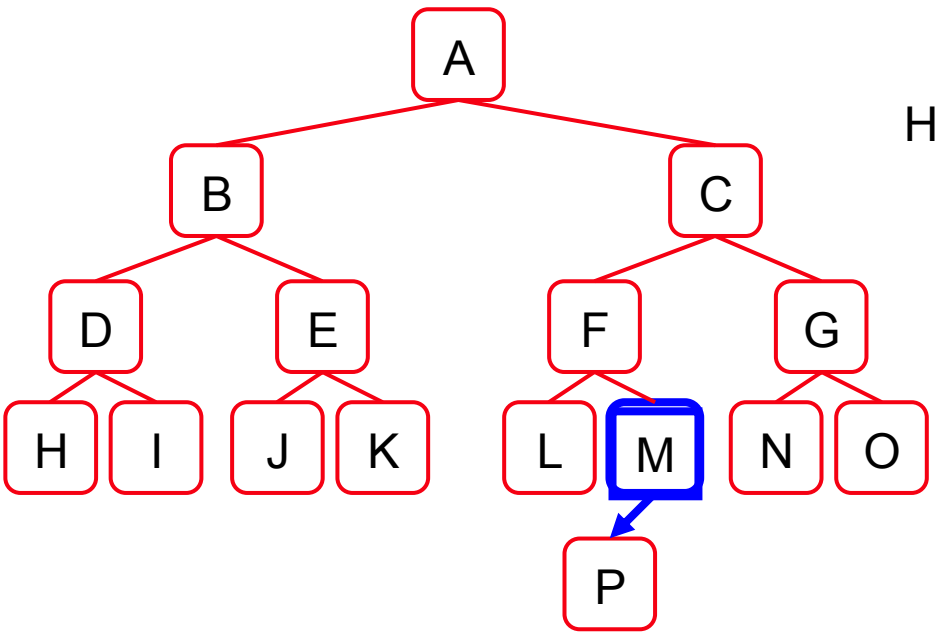
× betrachteter Vektor / im k-D Baum

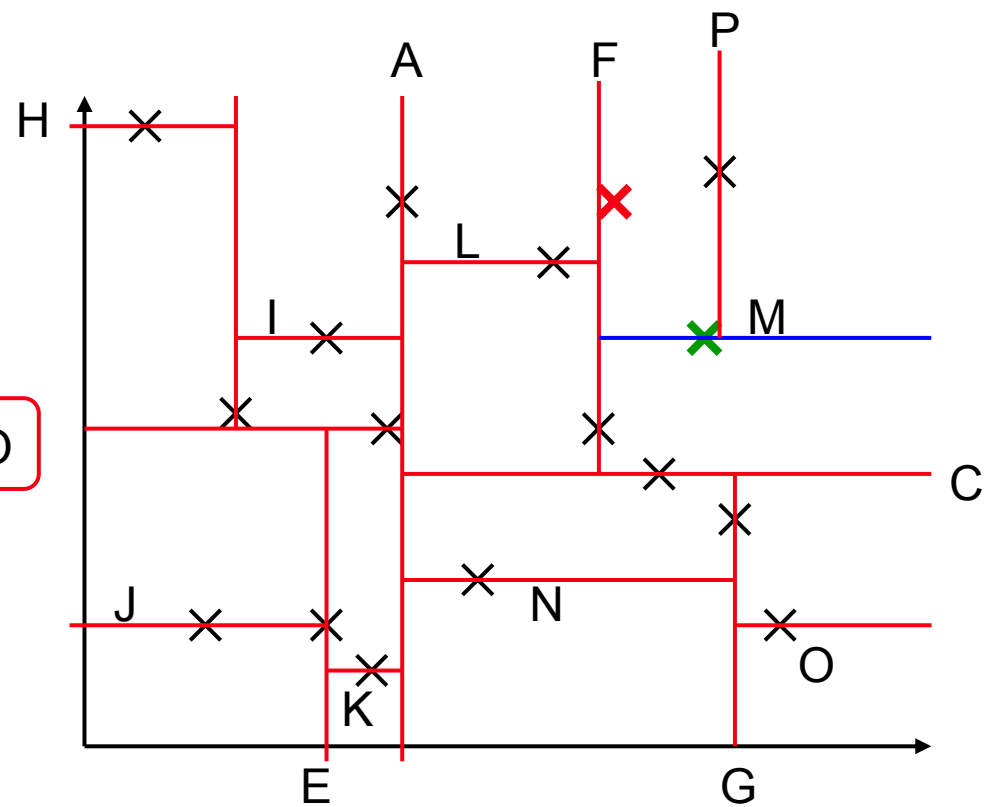
| Trennlinie bzw. Knoten

× Anfragevektor

× bisher nächster Nachbar

k-D Baum für Farbsegmentierung





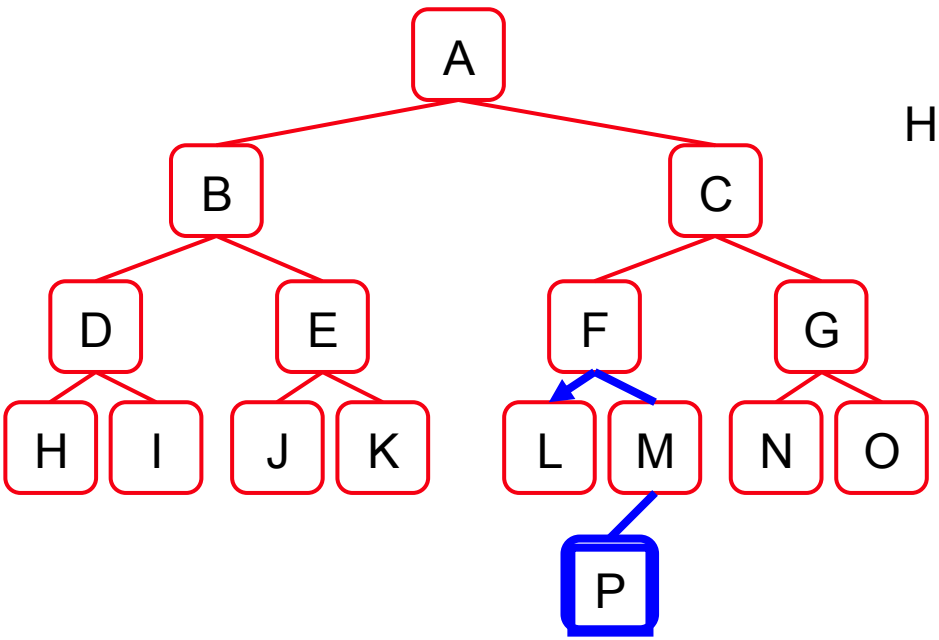
× betrachteter Vektor / im k-D Baum

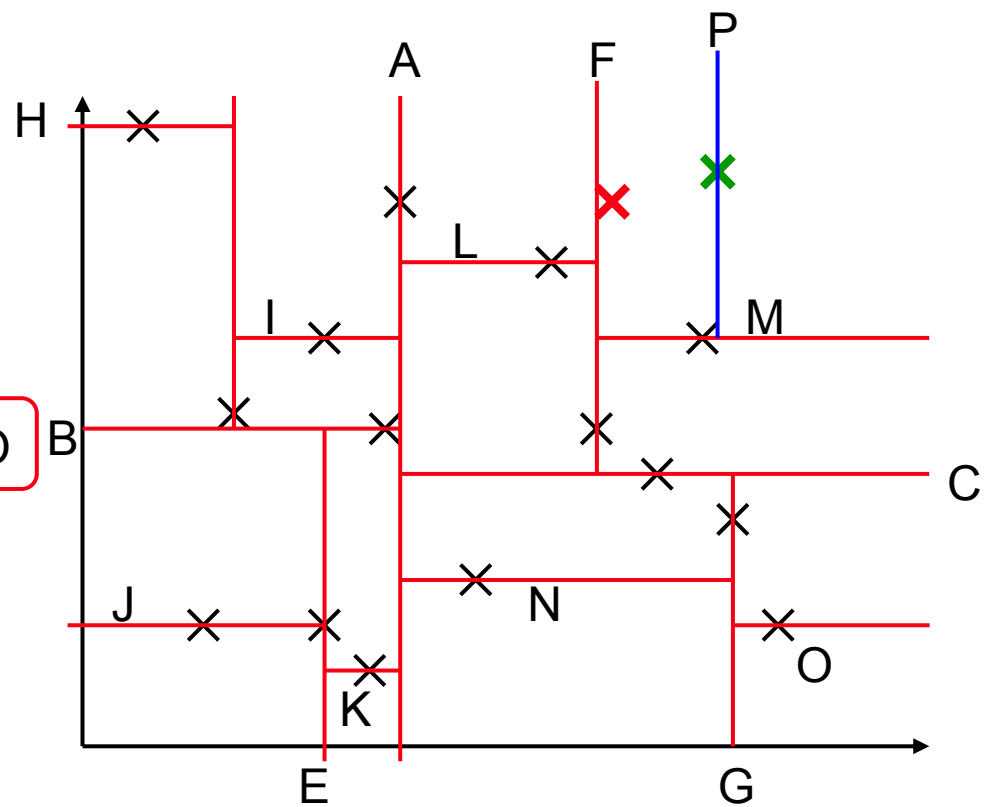
| Trennlinie bzw. Knoten

× Anfragevektor

× bisher nächster Nachbar

k-D Baum für Farbsegmentierung





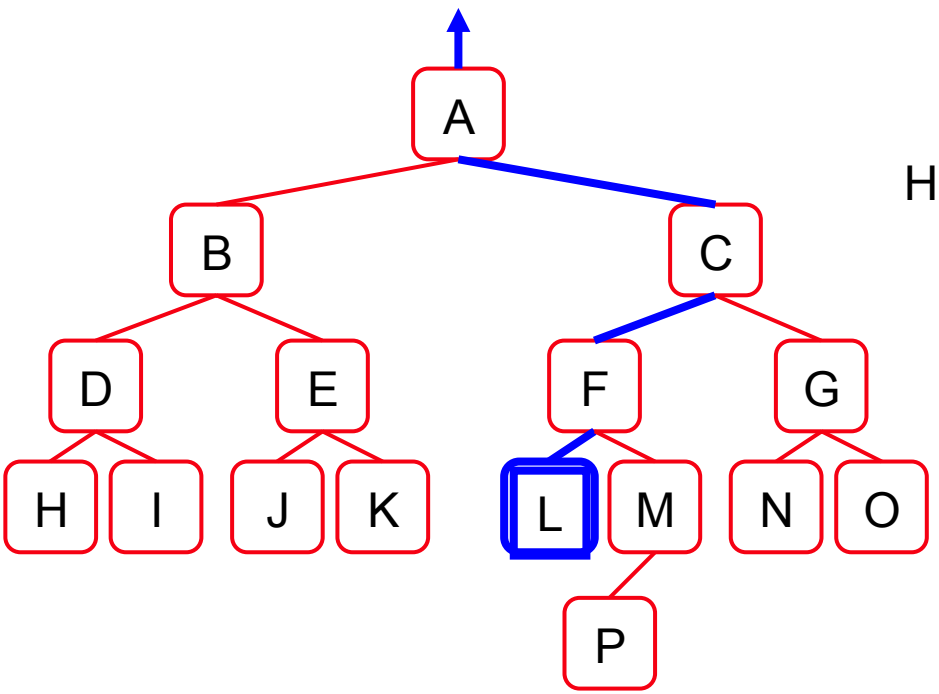
× betrachteter Vektor / im k-D Baum

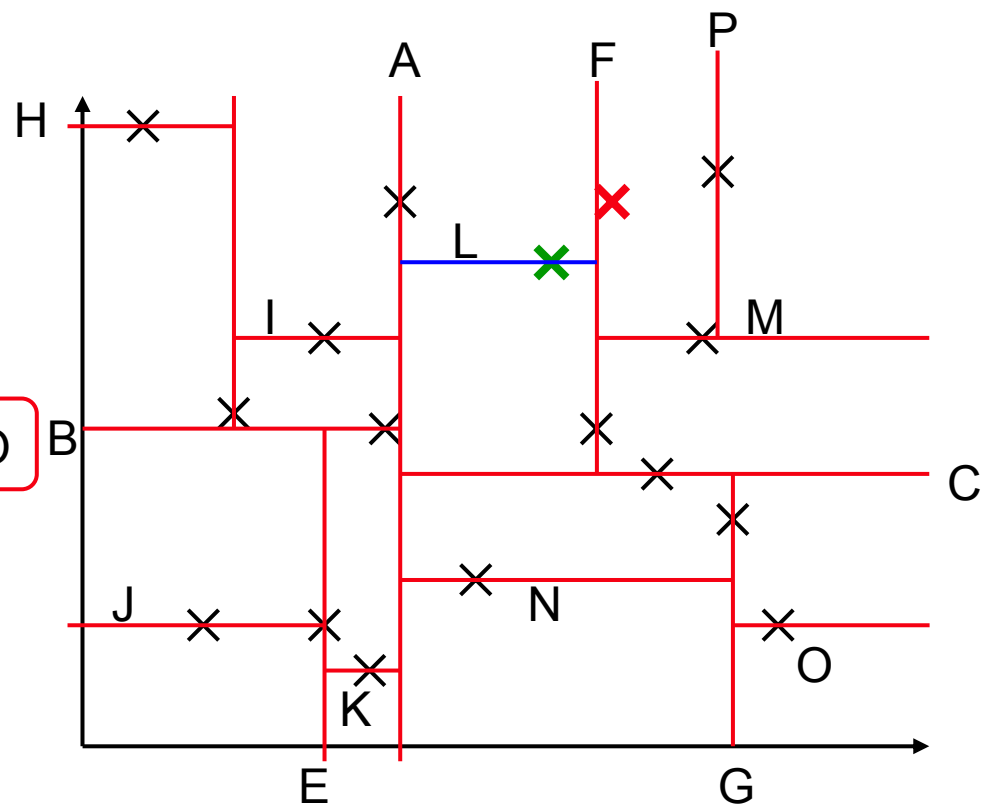
| Trennlinie bzw. Knoten

× Anfragevektor

× bisher nächster Nachbar

k-D Baum für Farbsegmentierung





× betrachteter Vektor / im k-D Baum

| Trennlinie bzw. Knoten

× Anfragevektor

× bisher nächster Nachbar

k-D Baum für Farbsegmentierung

k-D Baum: Implementierung nächster Nachbar

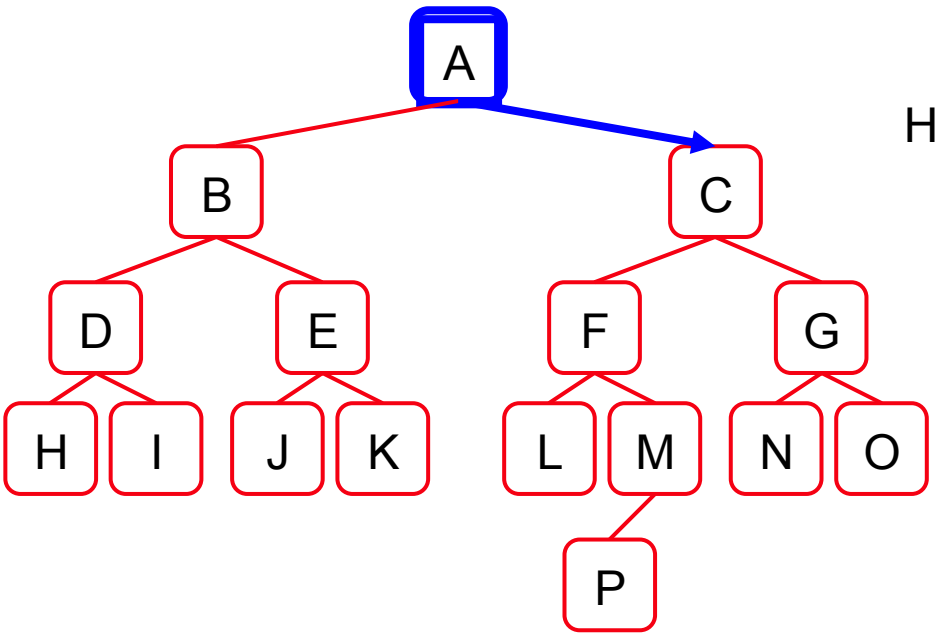
```
nearestNeighbour (TreeRoot, x, level, minD, nN) {
   d = TreeRoot->distance(x);
   if (d<minD) {minD=d; nN=TreeRoot;} // aktuellen Knoten betrachten
   d = x.color[level%k] - TreeRoot->color[level%k];
   if (d<0) { // x negativ vom aktuellen Knoten
      if (TreeRoot->below!=NULL) // erst negative Hälfte
         nextNeighbour (TreeRoot->below, x, level+1, minD, nN);
      if (TreeRoot->above!=NULL && -d<minD) // falls nötig positive
         nextNeighbour (TreeRoot->above, x, level+1, minD, nN);
   }
   else { // x positiv vom aktuellen Knoten
      if (TreeRoot->above!=NULL) // erst positive Hälfte
         nextNeighbour (TreeRoot->above, x, level+1, minD, nN);
      if (TreeRoot->below!=NULL && d<minD) // falls nötig negative
         nextNeighbour (TreeRoot->below, x, level+1, minD, nN);
   }
}
```

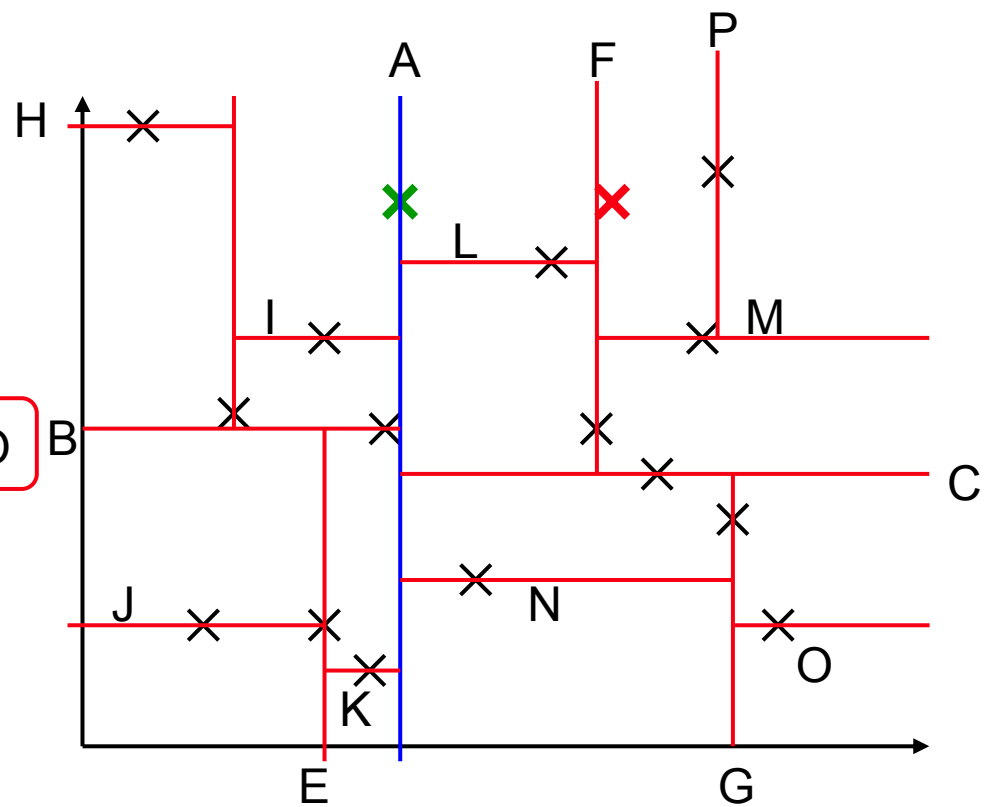
k-D Baum für Farbsegmentierung

k-D Baum: m nächste Nachbarn

- **Generalisierung: Sortierte Liste der m *bisher* nächsten Nachbarn**
- **Suche *m*-nächste Nachbarn**
  - Ist betrachteter Vektor näher als Listenende?
    Sortiere passend ein und entferne Ende. (`.insert`)
  - Rekursion nähere Hälfte
  - Rekursion fernere Hälfte, es sei denn…
  - ...Entfernung zur ferneren Hälfte ist größer als das Listenende
- **Initialisiere Liste mit m mal maximaler akzeptierter Distanz**
  - so einfacher und schneller

k-D Baum für Farbsegmentierung





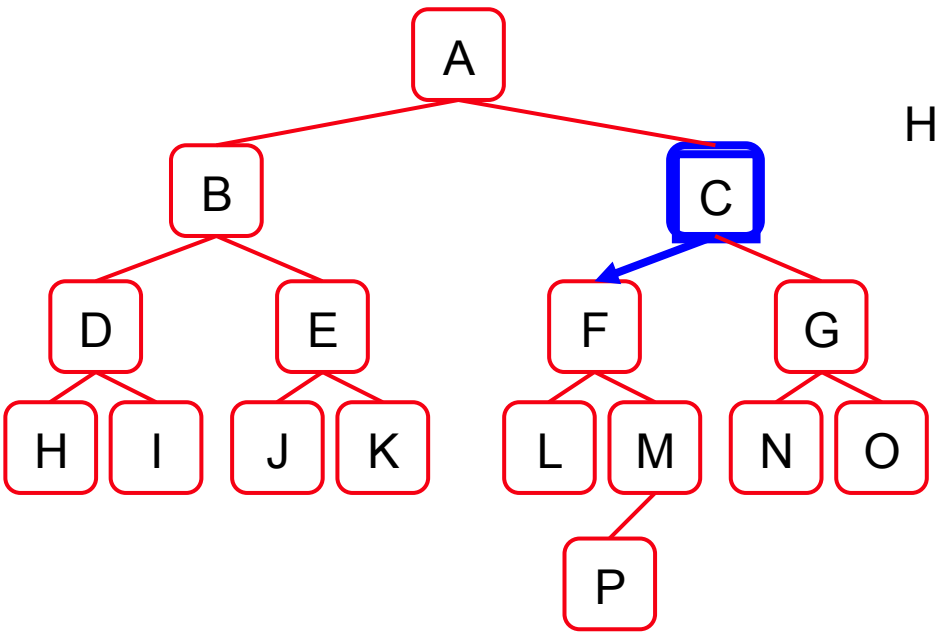
× betrachteter Vektor / im k-D Baum

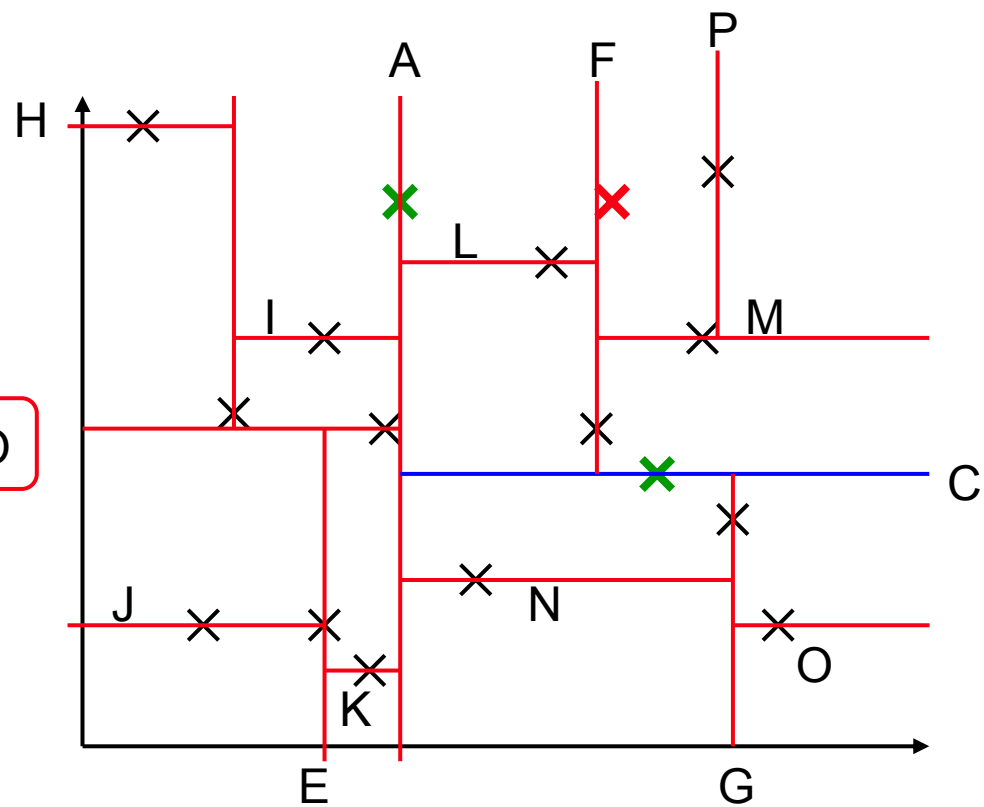
| Trennlinie bzw. Knoten

× Anfragevektor

× bisher nächster Nachbar

k-D Baum für Farbsegmentierung





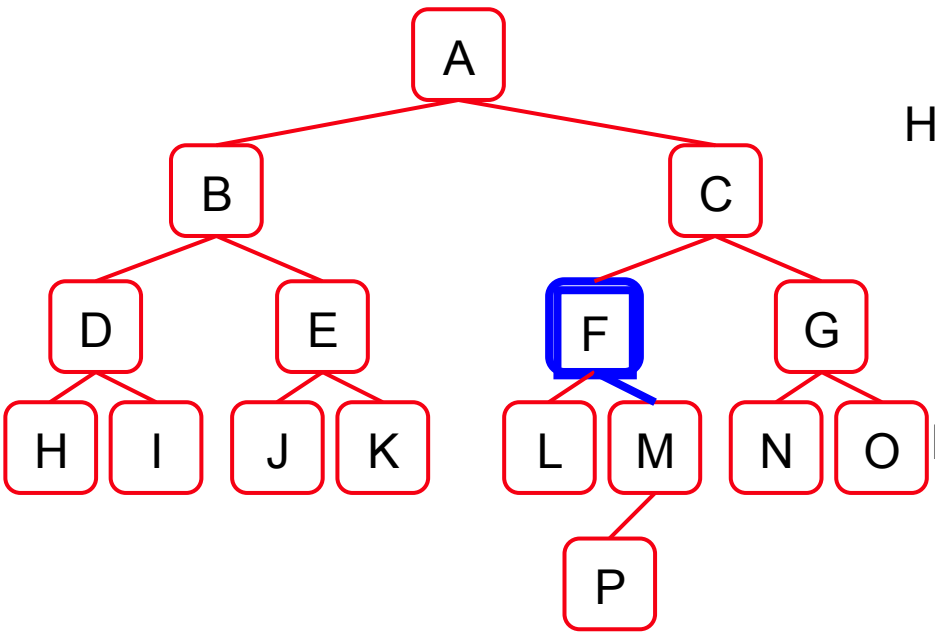
× betrachteter Vektor / im k-D Baum

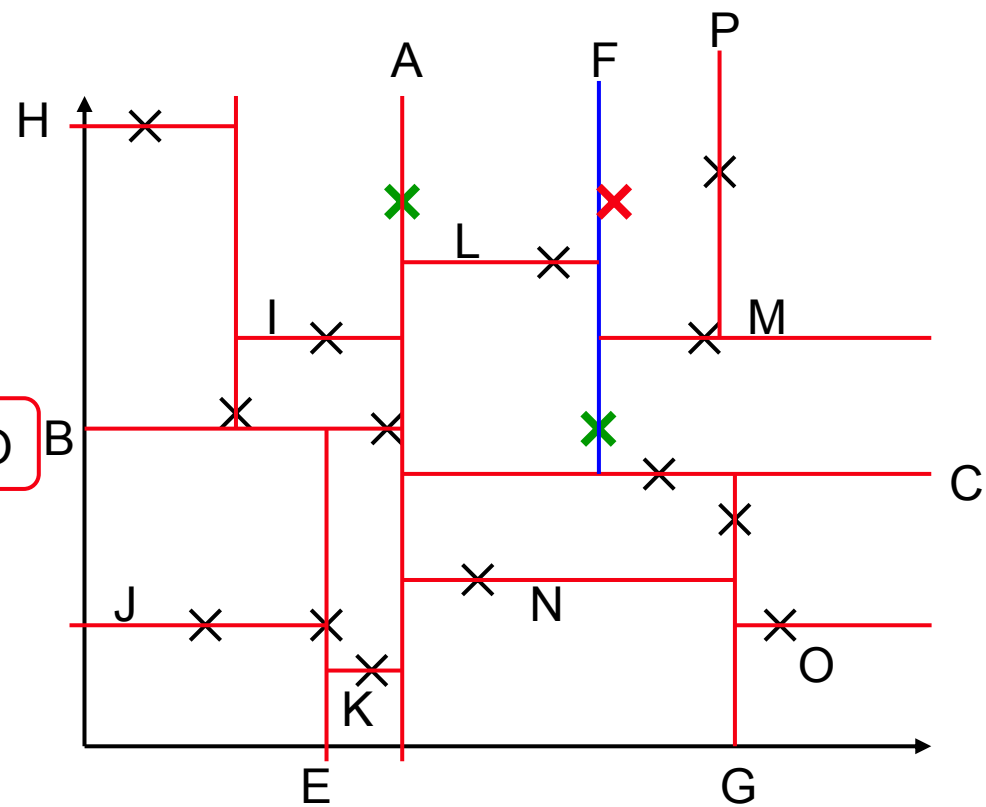
| Trennlinie bzw. Knoten

× Anfragevektor

× bisher nächster Nachbar

k-D Baum für Farbsegmentierung





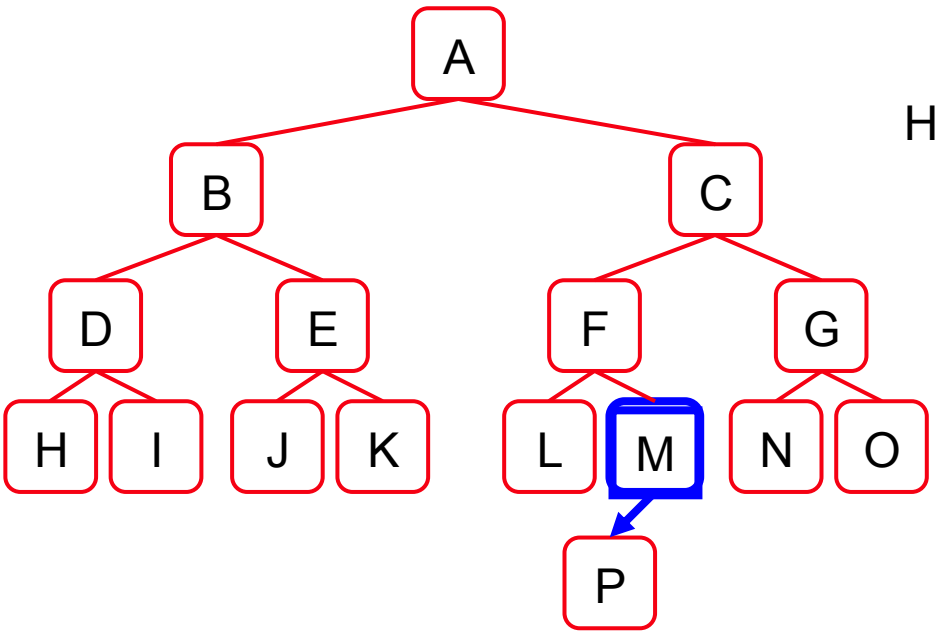
× betrachteter Vektor / im k-D Baum

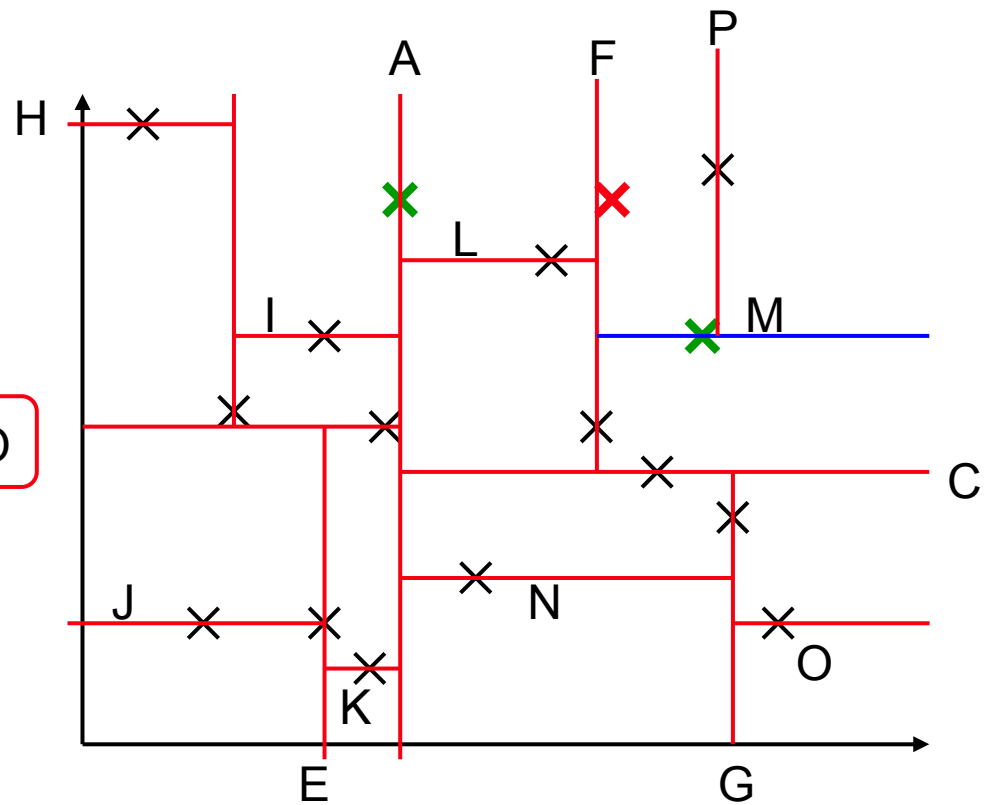
| Trennlinie bzw. Knoten

× Anfragevektor

× bisher nächster Nachbar

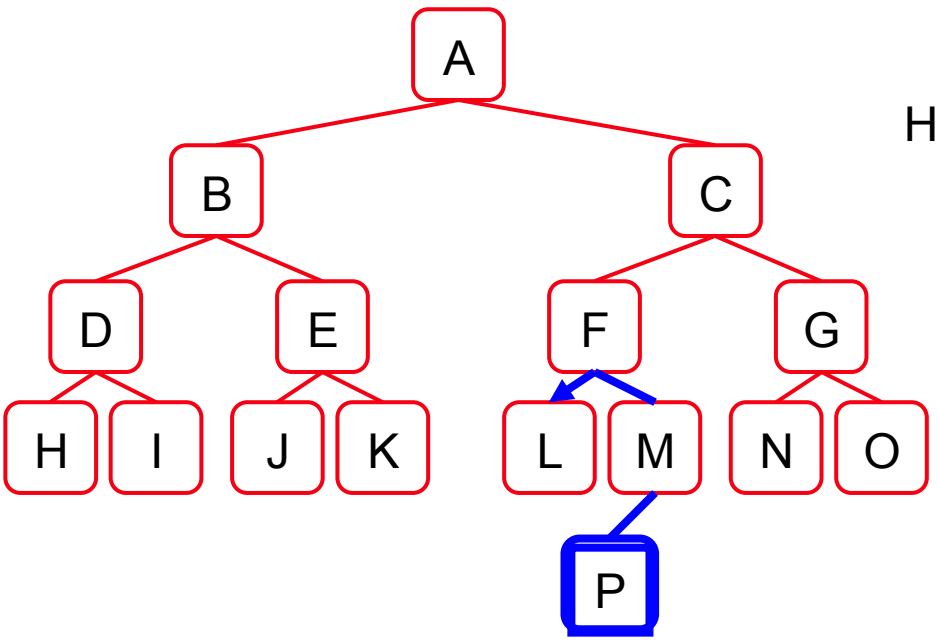
k-D Baum für Farbsegmentierung

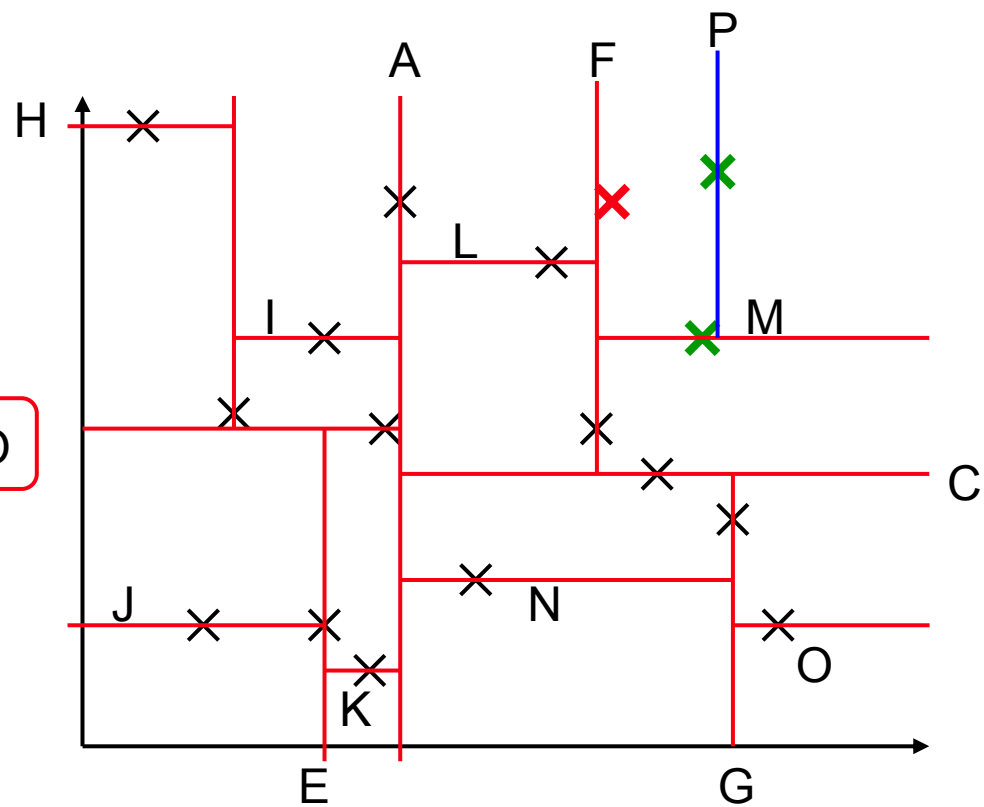




- betrachteter Vektor / im k-D Baum
- Trennlinie bzw. Knoten
- Anfragevektor
- bisher nächster Nachbar

k-D Baum für Farbsegmentierung





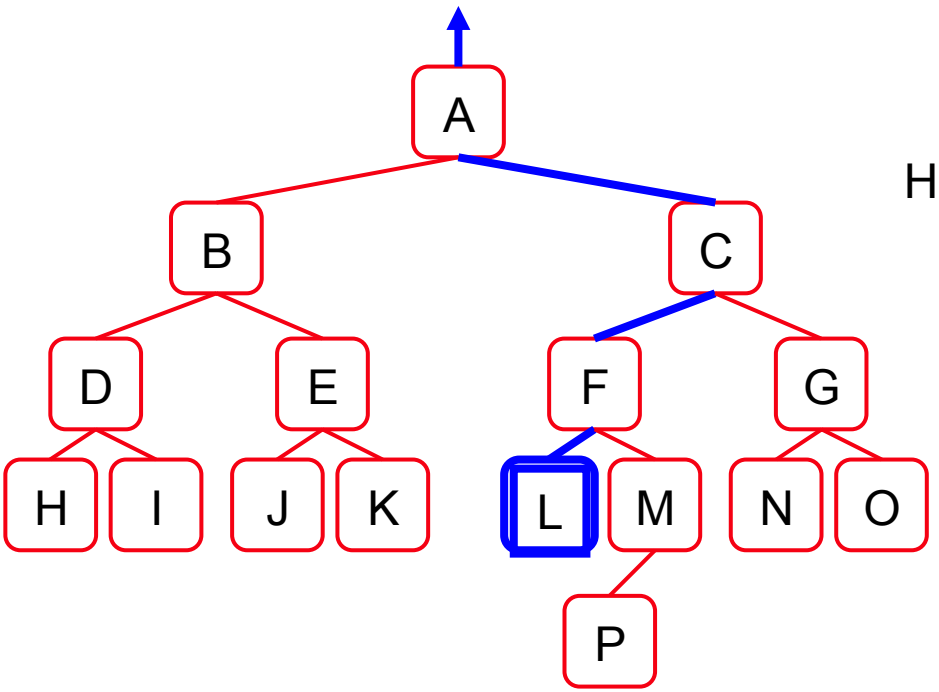
× betrachteter Vektor / im k-D Baum

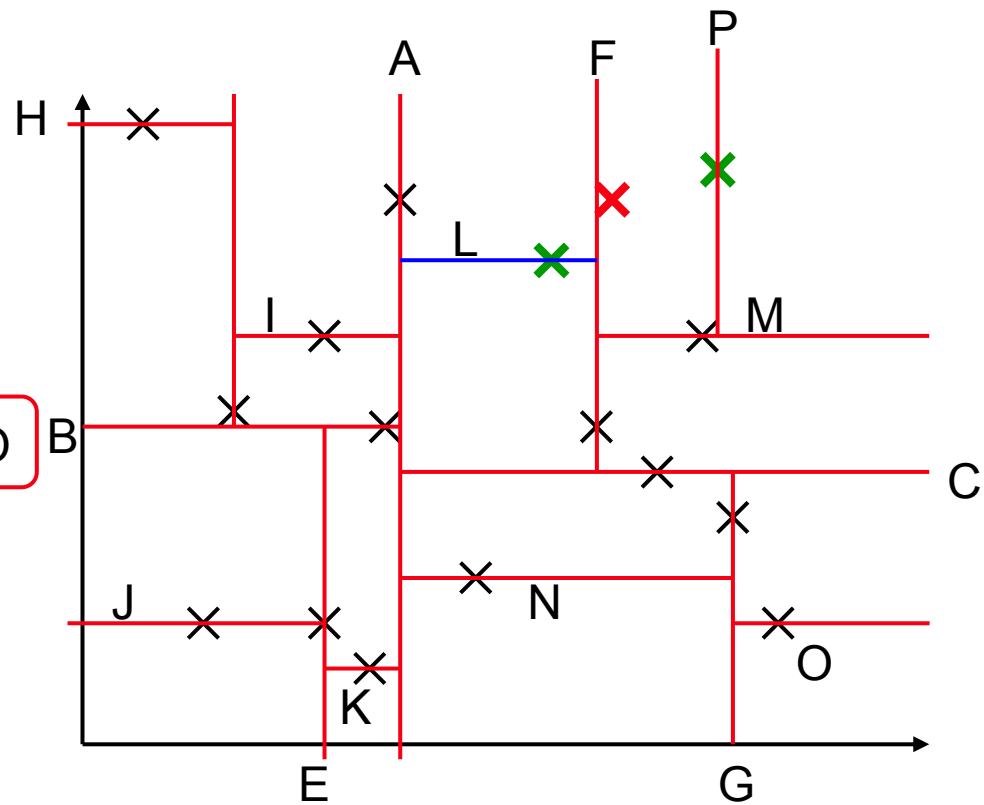
| Trennlinie bzw. Knoten

× Anfragevektor

× bisher nächster Nachbar

k-D Baum für Farbsegmentierung





× betrachteter Vektor / im k-D Baum

| Trennlinie bzw. Knoten

× Anfragevektor

× bisher nächster Nachbar

k-D Baum für Farbsegmentierung

k-D Baum: Implementierung nächste Nachbarn

```
mNearestNeighbours (TreeRoot, x, level, nNs) {
   nNs.insert (TreeRoot); // aktuellen Knoten in nNs einsortieren auf m abschneiden
   d = x.color[level%k] - TreeRoot->color[level%k];
   if (d<0) { // x negativ vom aktuellen Knoten
      if (TreeRoot->below!=NULL) // erst negative Hälfte
          mNearestNeighbours (TreeRoot->below, x, level+1, nNs);
      if (TreeRoot->above!=NULL && -d<nNs.lastDistance) // falls nötig positive
          mNearestNeighbours (TreeRoot->above, x, level+1, nNs);
   }
   else { // x positiv vom aktuellen Knoten
      if (TreeRoot->above!=NULL) // erst positive Hälfte
          mNearestNeighbours (TreeRoot->above, x, level+1, nNs);
      if (TreeRoot->below!=NULL && d<nNs.lastDistance) // falls nötig negative
          mNearestNeighbours (TreeRoot->below, x, level+1, nNs);
   }
}
```

k-D Baum für Farbsegmentierung

m-Nearest Neighbour: Look-up-tables

- **Trotz Optimierung nicht echtzeitfähig (ca. 274ms für 176×144 Bild auf P-M 1.5GHz)**
- **Optimierung: Tabelliere für alle (16M) möglichen Farben die Klassifizierung vor**
- **Anwenden der Tabelle viel schneller (ca. 0.774ms)**

Farbs. durch Look-up-tables

- **Frage an das Auditorium: Ist das (176×144 Bild, 0.77ms) eigentlich schnell?**
- **Ganz okay, aber immerhin 46 Zyklen / Pixel auf Pentium M, 1.5 GHz**
- **Frage an das Auditorium: Wo geht die Rechenzeit verloren?**
- **Willkürlicher Speicherzugriff.**
- **Wenn die Daten nicht im Cache stehen, haben modernen Rechnern ca. 100 Taktzyklen Latenzzeit**
- **Frage an das Auditorium: Wie könnte man noch schneller werden?**
- **Kleinere Tabelle, z.B. für jeden RG Wert Intervalle von B Werten.**
- **Ergebnis (siehe Musterlösung): 0.44ms**

Applikation: RoboCup

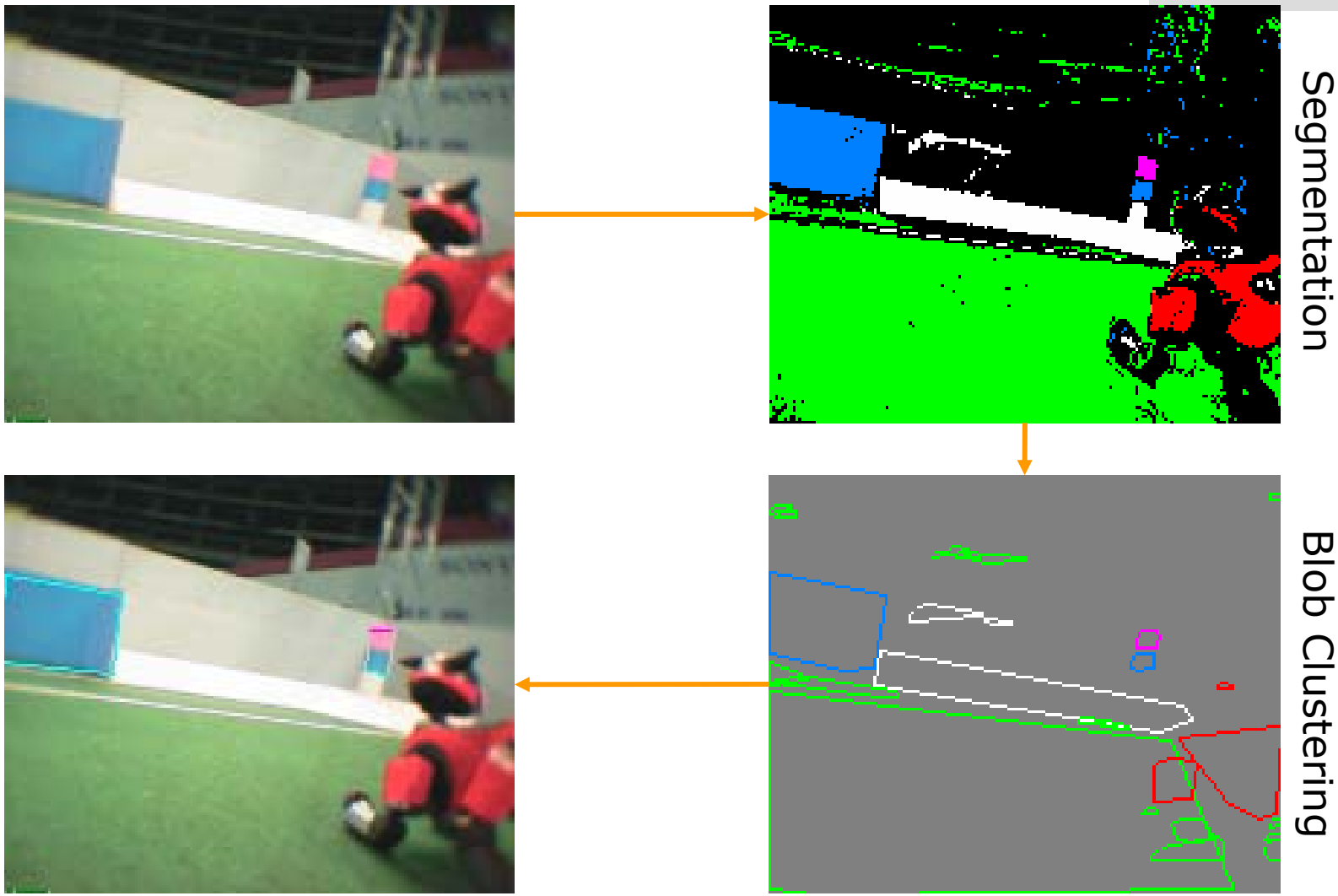
- By the year 2050,
develop a team of fully autonomous humanoid robots
that can win against the human world soccer champion team. -

Quelle des Abschnittes: Thomas Röfer, SFB/TR 8 Kolloqiumsvortrag 2004
Teile der Grafiken: Hans-Dieter Burkhardt
www.**germanteam**.org


Sony Legged Robot League 2004 in Lisbon
10 : 00
GERMAN TEAM
0
0

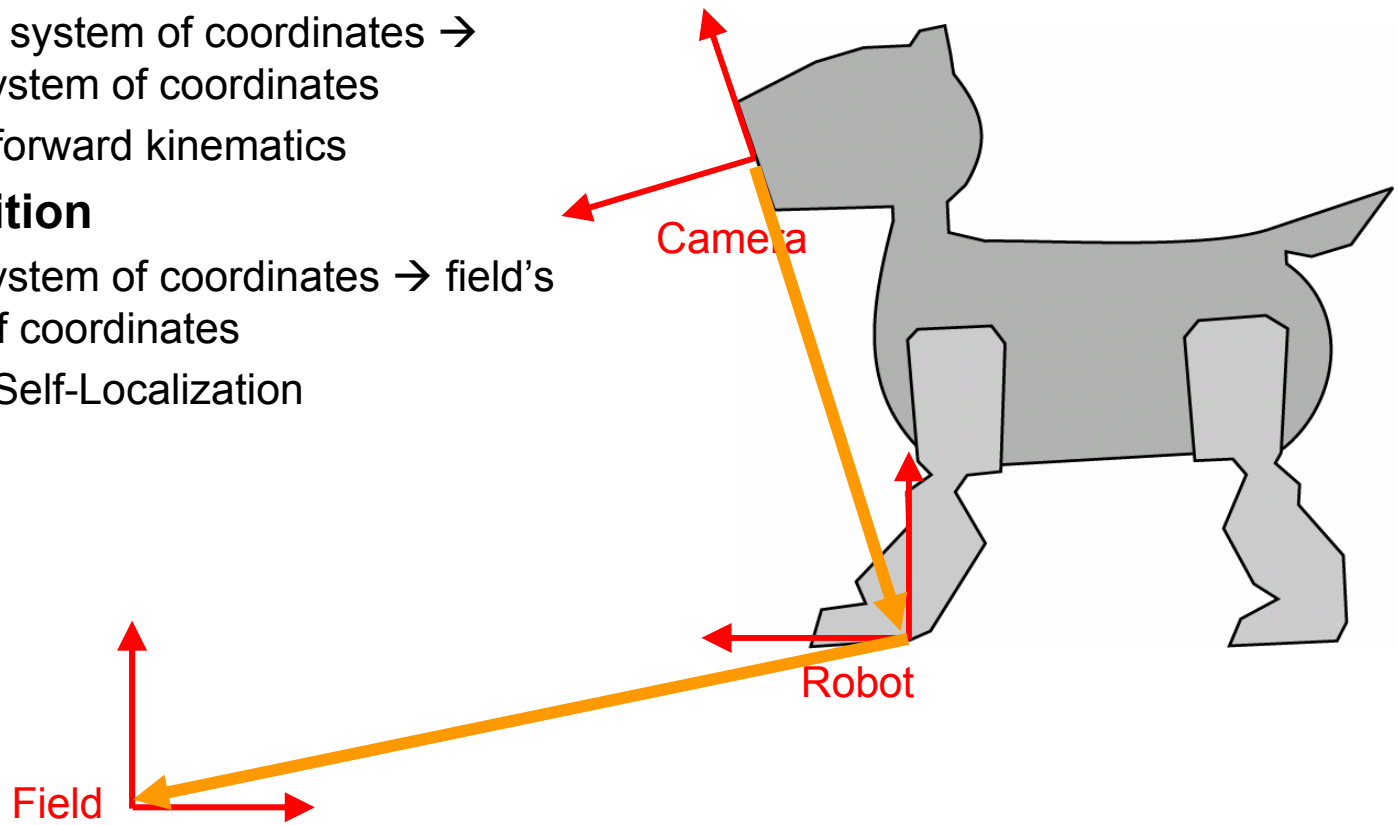
Applikation: RoboCup

Applikation: RoboCup

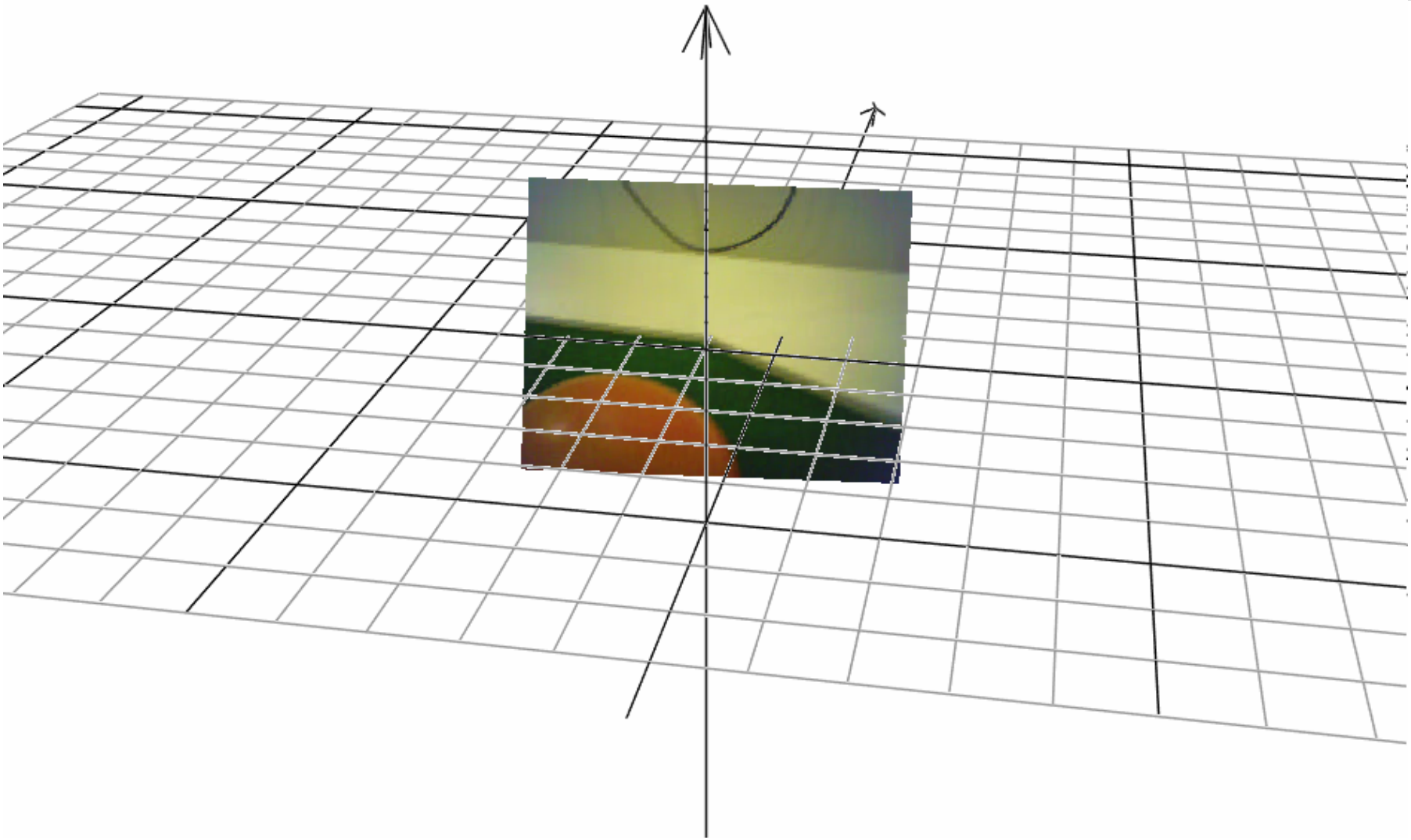


Applikation: RoboCup

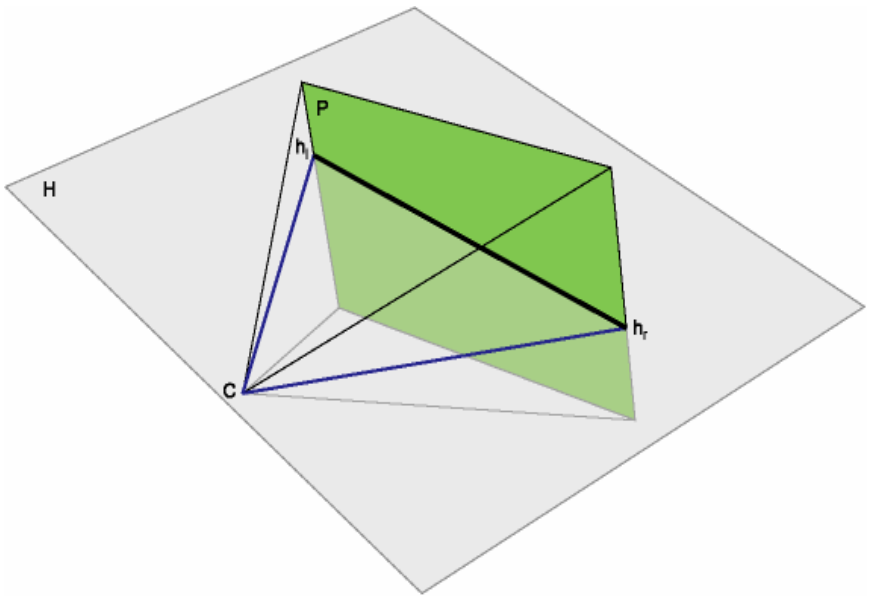
- **Camera Position**
  - camera's system of coordinates → robot's system of coordinates
  - Method: forward kinematics
- **Robot Position**
  - robot's system of coordinates → field's system of coordinates
  - Method: Self-Localization



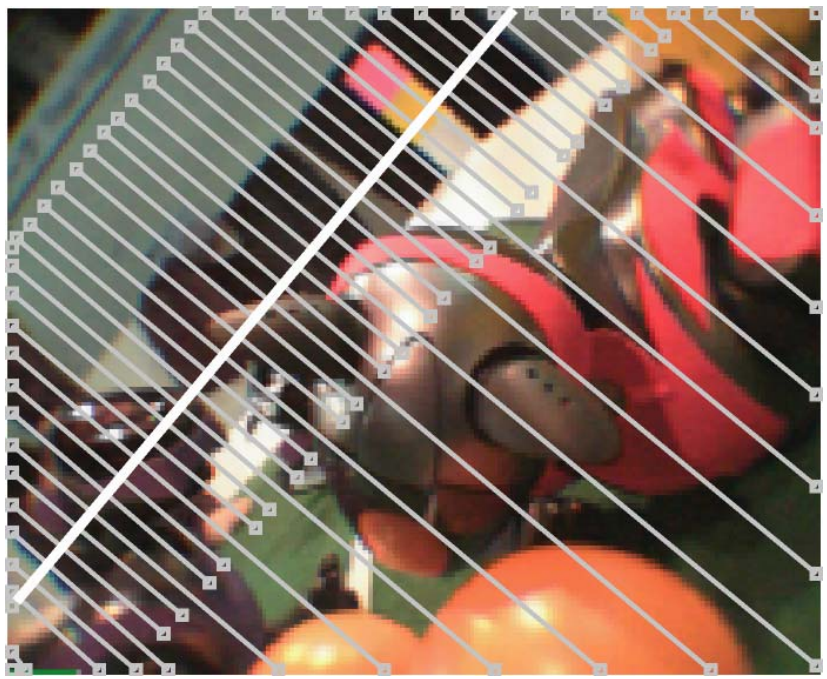
Applikation: RoboCup Coordinates



Applikation: RoboCup

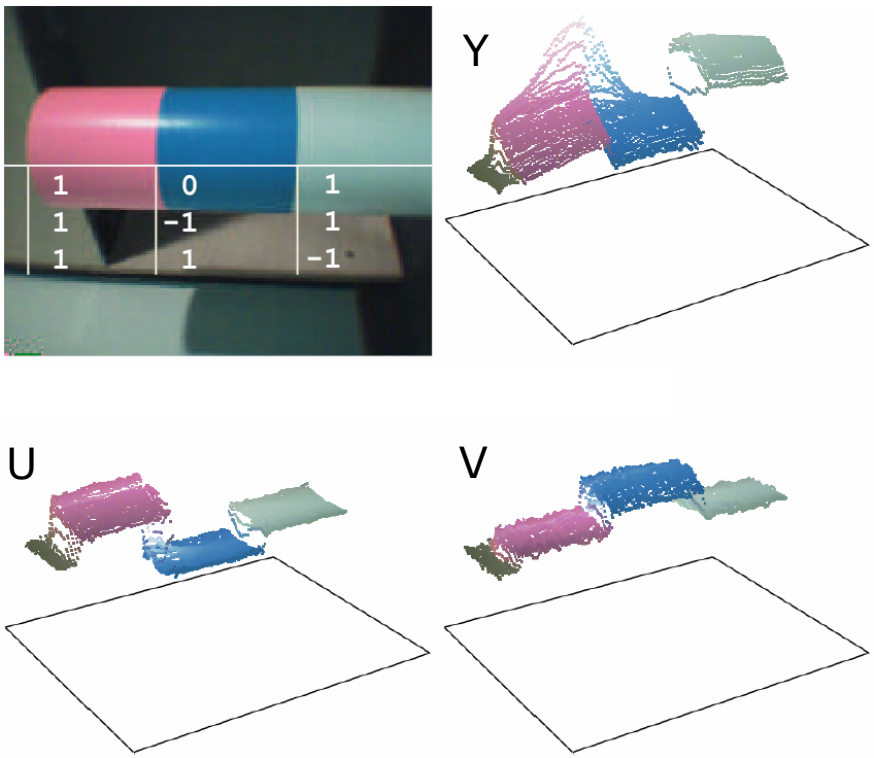


Horizon

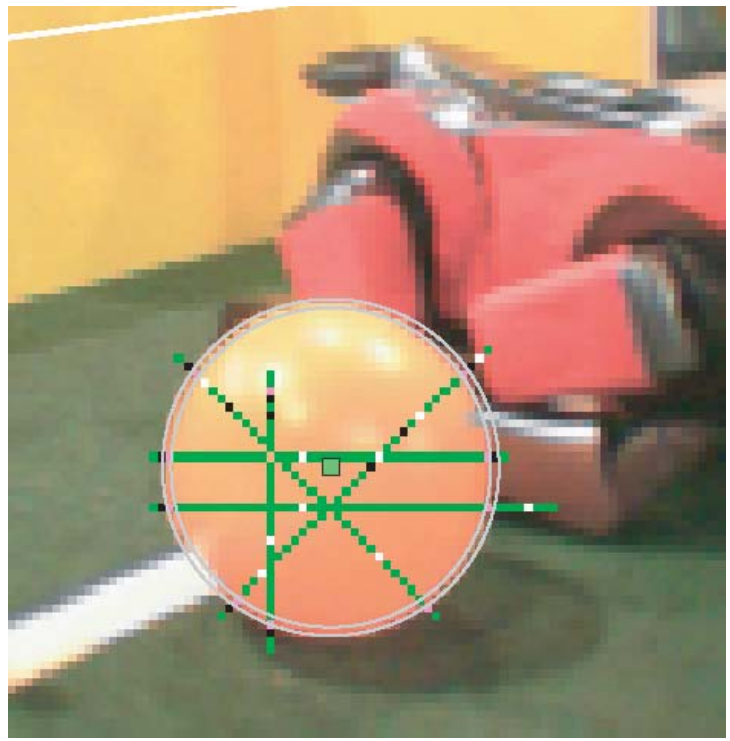


Horizon-aligned grid

Applikation: RoboCup

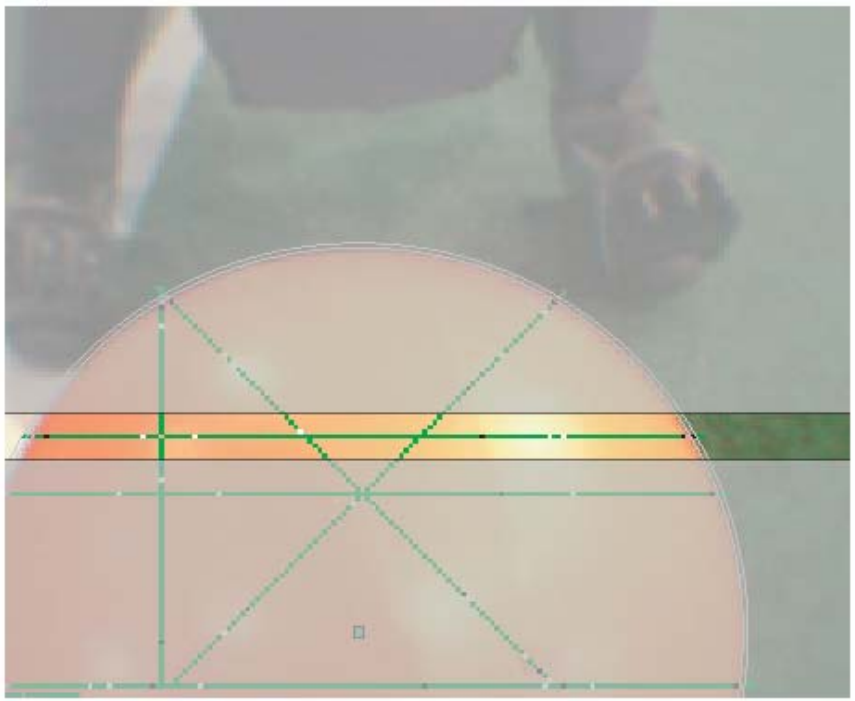


Pattern of a beacon

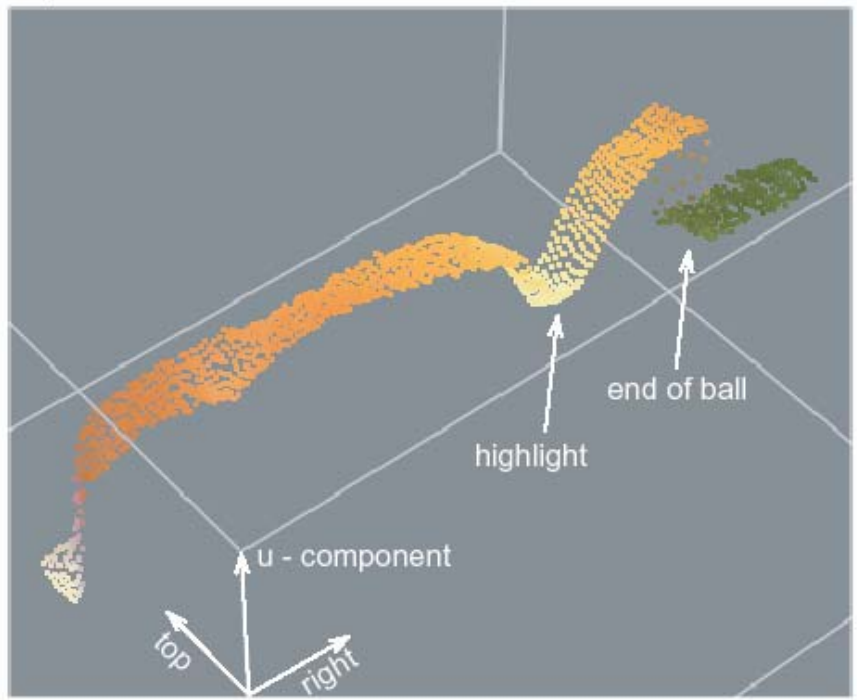


Ball specialist

Applikation: RoboCup



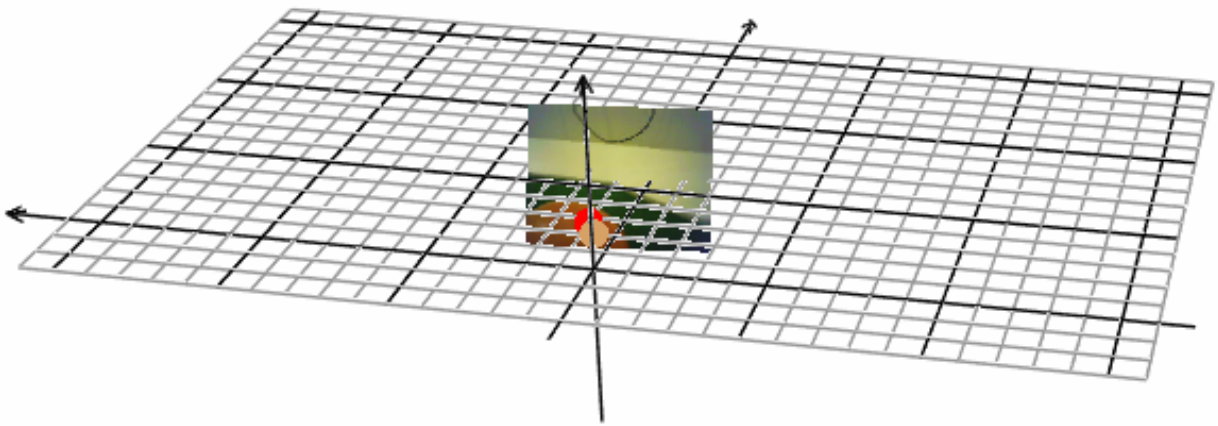
One scanline of the specialist



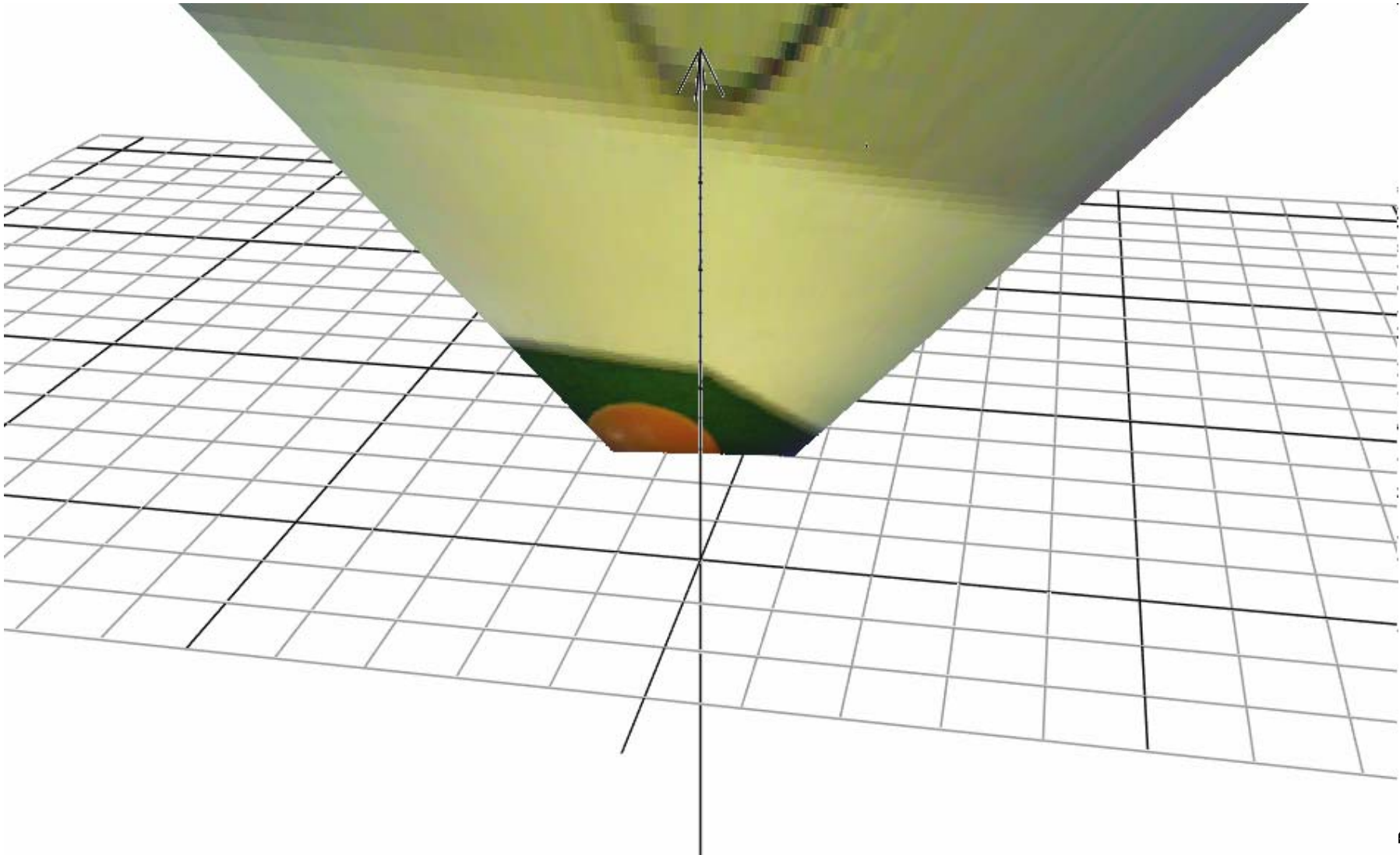
U-channel on scanline

Applikation: RoboCup

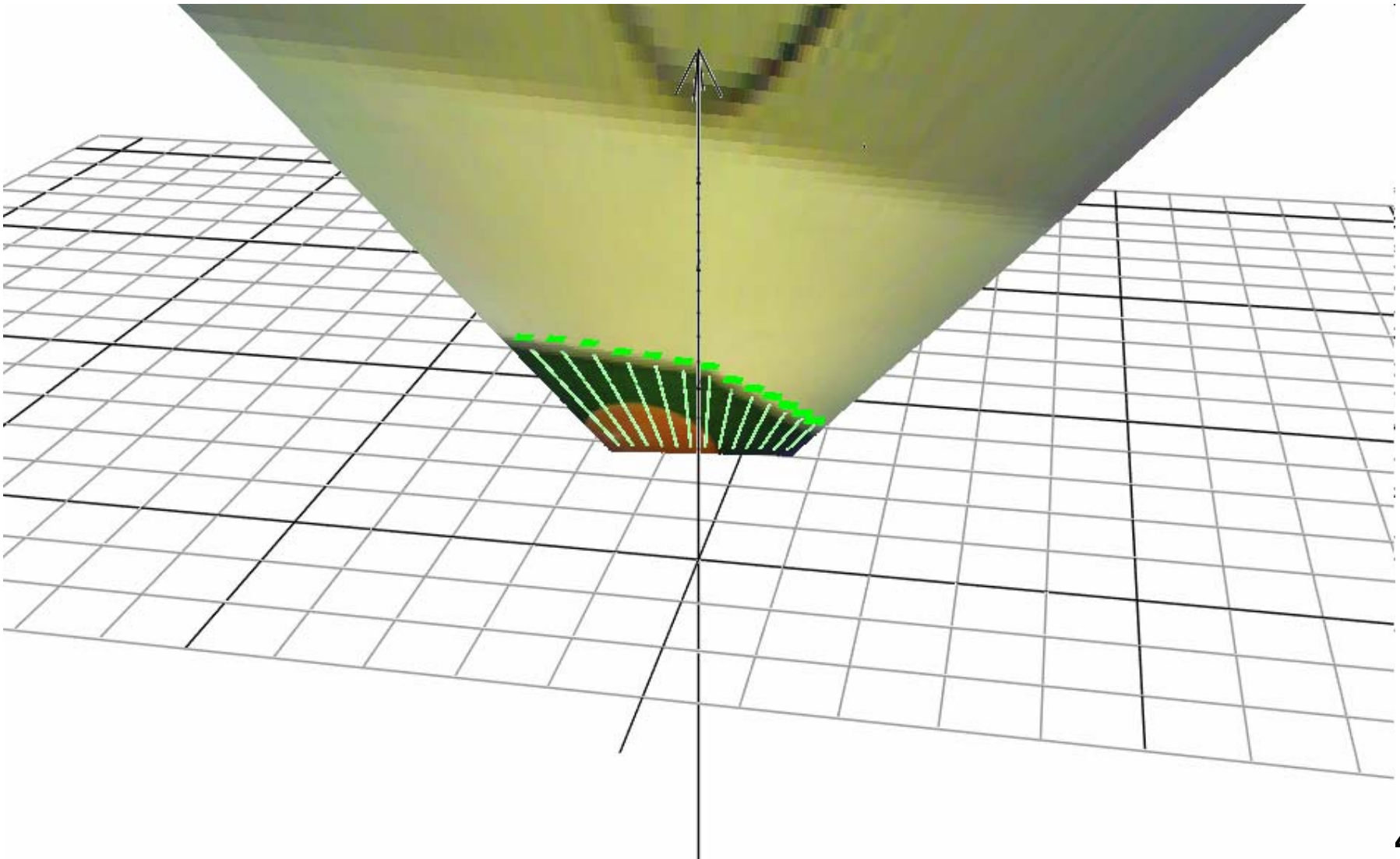
- **Egocentric Positions of Percepts**
  - direction (from images and camera position)
  - distance (from object size)



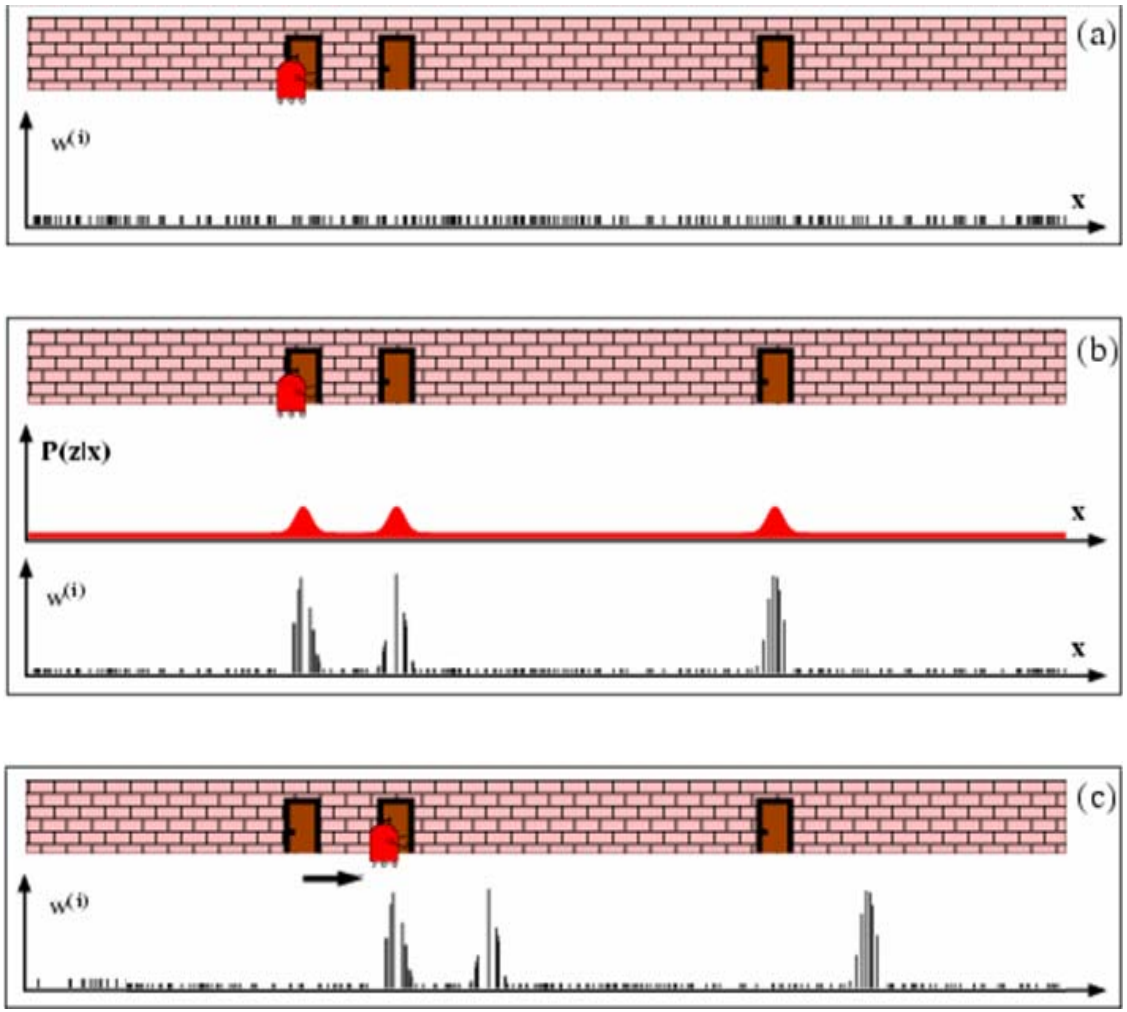
Applikation: RoboCup

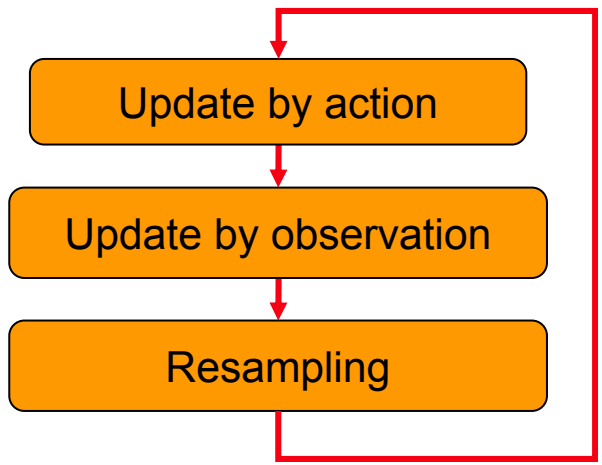


Applikation: RoboCup

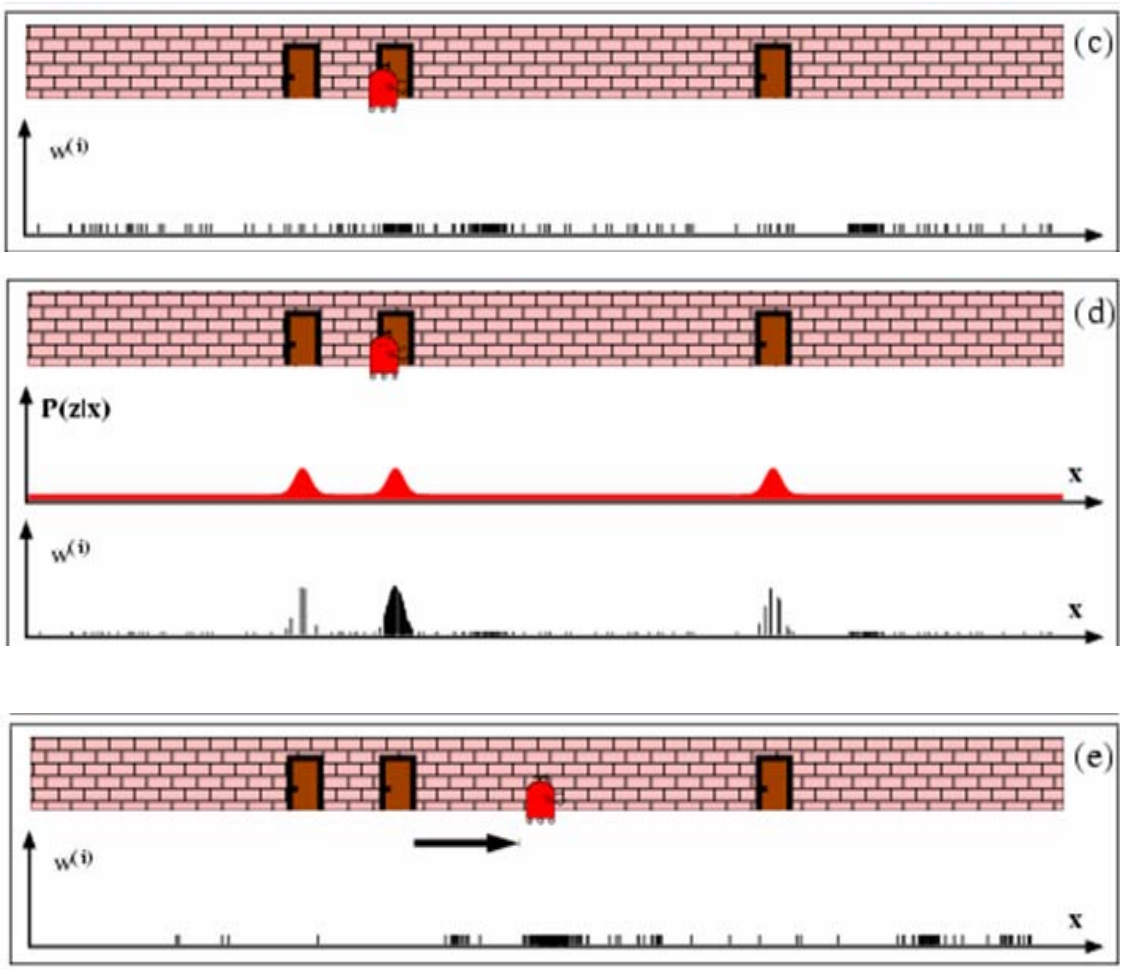


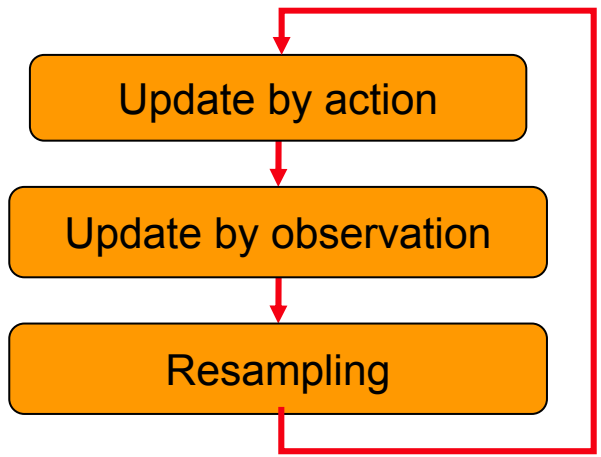
Self-Localization – Particle Filter



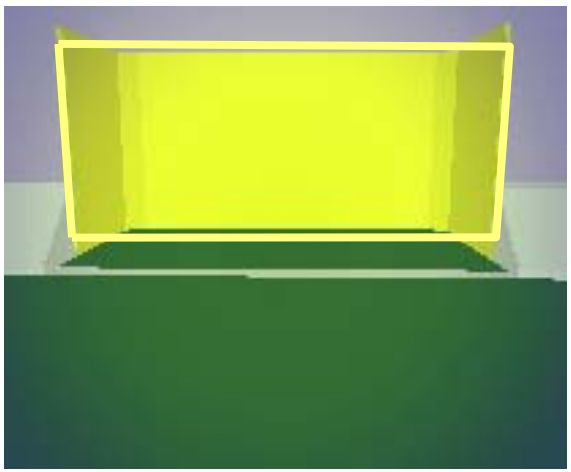


Self-Localization – Particle Filter

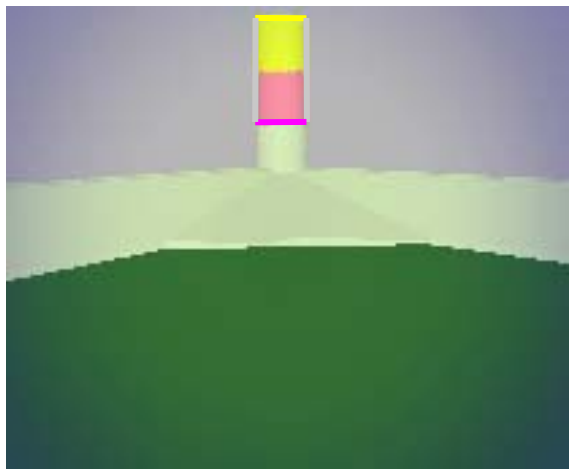




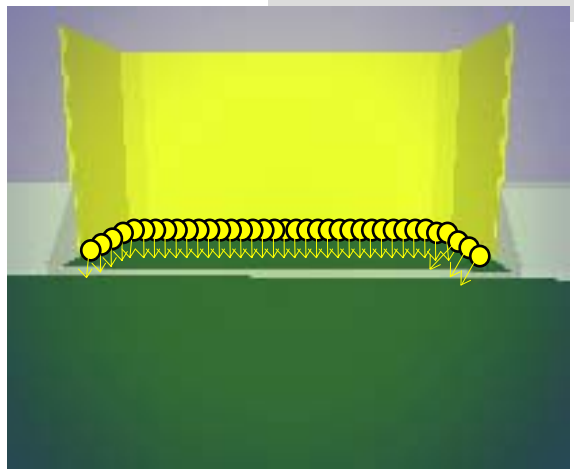
Applikation: RoboCup



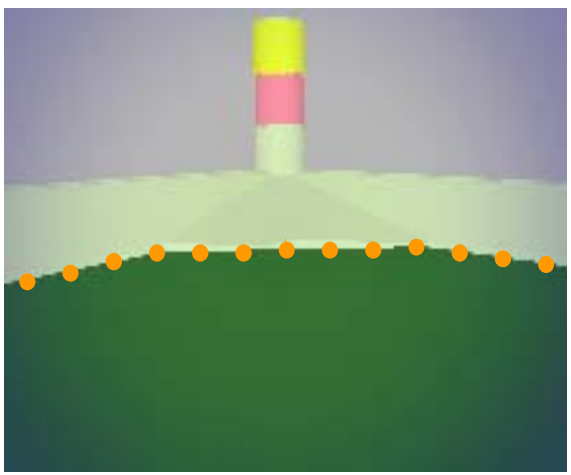
Goals



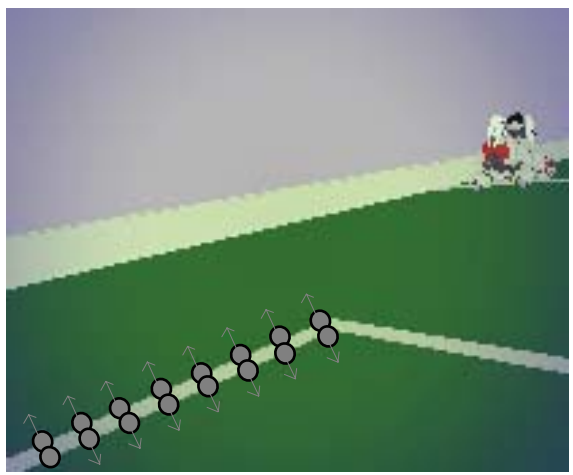
Beacons



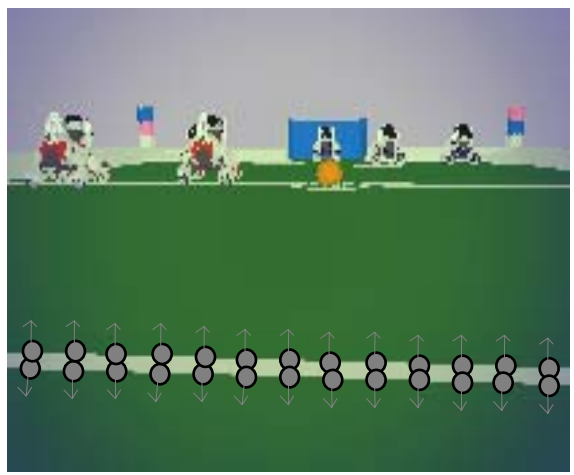
Goal points



Edges field/wall

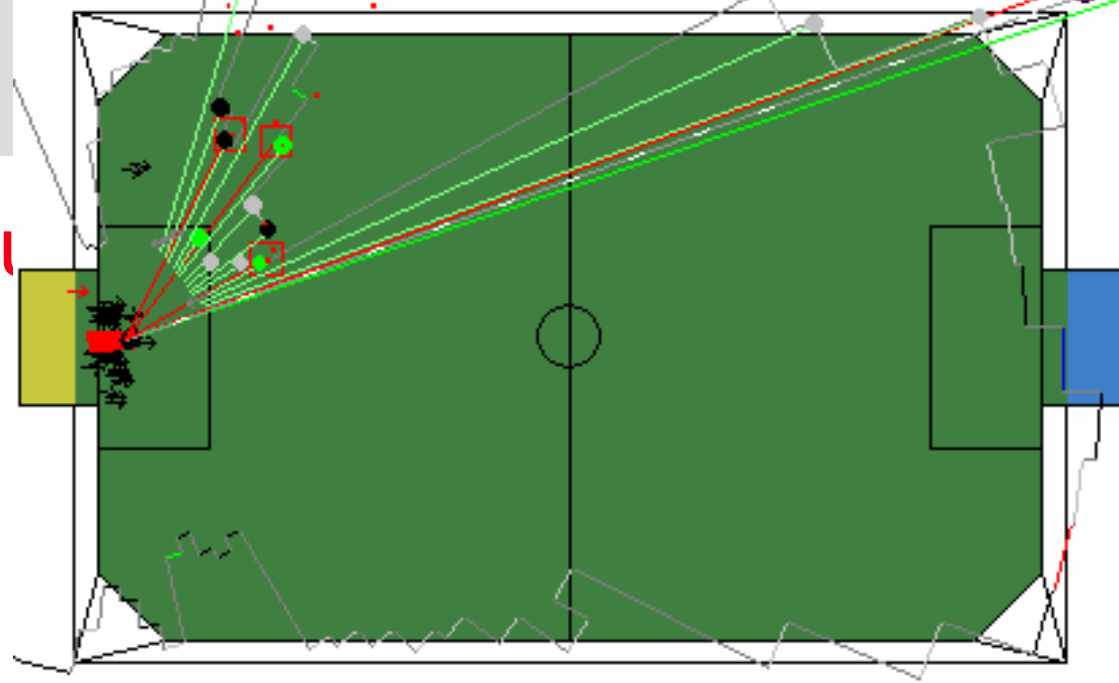


Field lines (vert.)



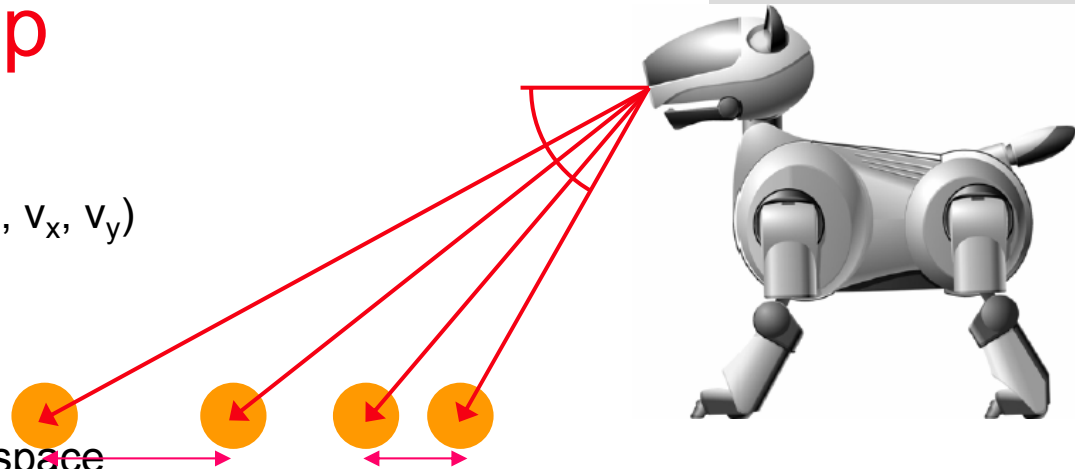
Field lines (horiz.)

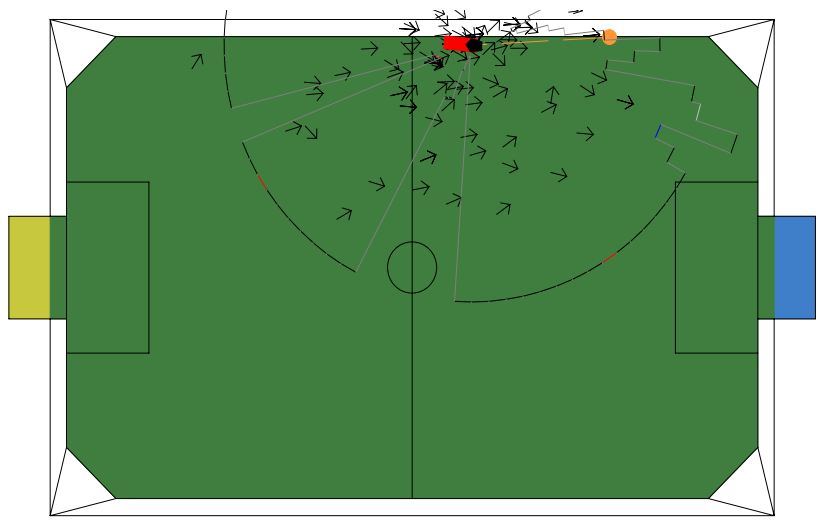
Applikation: RoboCu



Applikation: RoboCup

- **Ball Model**
  - Kalman Filter estimates (x, y, v_x, v_y)
  - Communicated positions are only used after a while
- **Obstacle Model**
  - Polar representation of free space
  - Border of free space is labeled by obstacle types





Zusammenfassung

- **k-D Baum für m-Nearest Neighbour Farbsegmentierung**
  - Klassifiziere eine Farbe gemäß der absoluten Mehrheit der m nächsten Trainingsvektoren (oder weise sie zurück)
  - Bilde Binärbaum der entlang abwechselnder Dimensionen jeweils halbiert (Aufbau mit Sortieren, Median und Rekursion)
  - Suche rekursiv, steige zuerst in die nähere Hälfte ab, dann in die fernere, falls der m-t nächste bisher gefundene weiter entfernt ist als die Grenze
- **Tabelliere das Ergebnis der Klassifikation für alle Farben**
- **Vermeide Tabelle größer als Cache, dann lieber etwas rechnen**
- **Bildverarbeitung GermanTeam im Sony Four-legged RoboCup**
  - Tabellenbasierte Farbsegmentierung als Basis
  - Projektion auf den Boden durch bekannte Kamerapose
  - Lokalisation mit Partikelfiltern
  - Viele spezielle Tricks um Effizienz und Robustheit zu verbessern