Hochschule Darmstadt

Lehrveranstaltung: Simulation von Robotersystemen
Dozent: Prof. Dr. Thomas Horsch
Laboringenieur: Rudi Scheitler
Versuch: Konfigurationsraum
Datum: 10.6.2016

Name	Matrikelnummer
Fabian Alexander Wilms	735162

Studiengang: Mechatronik
Abgabedatum: 17.6.2016

Inhaltsverzeichnis

Ein Konfigurationsraum stellt grafisch dar, für welche Gelenkstellungen ein Roboter mit der Umgebung kollidiert. Für zwei Gelenkvariablen lässt sich dies in einem x-y-Diagramm darstellen.

Das Ziel dieses Praktikums ist es, für zwei gegebene Roboter samt Umgebungen jeweils einen Konfigurationsraum zu bestimmen. Dabei beschränken wir uns auf Modelle in der Ebene.

1 Translatorischer Roboter

In der ersten Aufgabe ist ein quadratischer TT-Roboter (2 translatorische Freiheitsgrade in x- und y-Richtung) gegeben. Um die Nulllage herum befinden sich 5 Rechtecke, welche die Hindernisse darstellen. Um sich dies besser vorstellen zu können und den resultierenden Konfigurationsraum einfacher auf Korrektheit überprüfen zu können, ist zudem eine EasyRob .cel-Datei gegeben, die Roboter und Umgebung enthält. In EasyRob können allerdings nur dreidimensionale Modelle hinzugefügt werden, weshalb alle Rechtecke zu Quadern extrudiert sind.

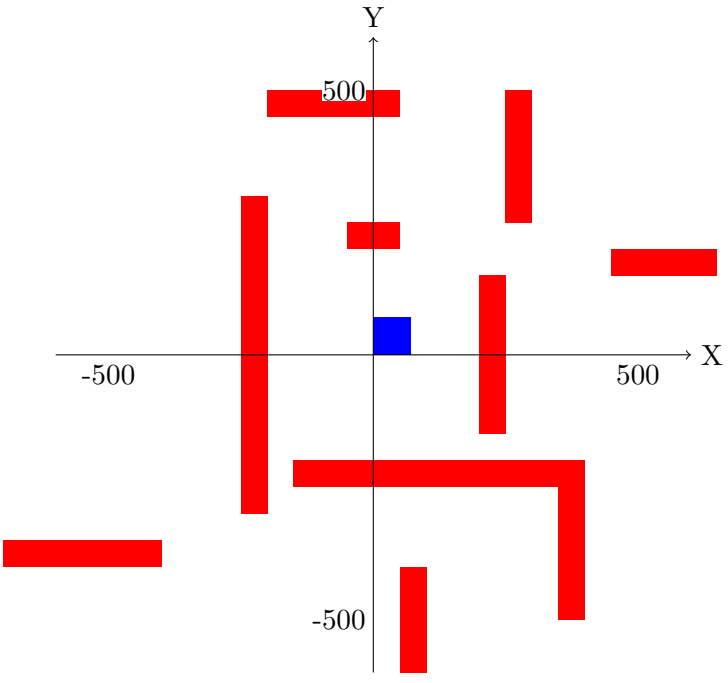

Abbildung 1: Roboter in Nullstellung (blau) und Hindernisse (rot)

Roboter und Umgebung werden in C++ mithilfe der Klasse `Box` aus der OPCODE-Bibliothek modelliert. Diese Bibliothek ermöglicht eine ressourcen-

sparende Kollisionserkennung. Ein Objekt dieser Klasse hat 9 Freiheitsgrade: Skalierung, Translation und Rotation im Raum. Der Konfigurationsraum wird im zweidimensionalen Array `cspace` gespeichert. `cspace` ist implementiert als Array der Größe `height`, welches in jedem Feld ein Array der Größe `width` vom Typ `char` enthält.

Die skalierten und verschobenen Objekte vom Typ `Box` werden in zwei weiteren Array gespeichert: `aRobot` und `aHindernis`.

Die eigentliche Bestimmung des Konfigurationsraums findet mithilfe von vier verschachtelten `for`-Schleifen statt. Die äußere durchläuft alle möglichen Werte für Gelenkvariable 1 von 0 bis `width`. Die nächste Schleife durchläuft alle möglichen Werte für Gelenkvariable 2 von 0 bis `height`. Damit steht innerhalb dieser zweiten Schleife die Position des Roboters fest. Mittels der Methode `Box::Translate()` wird der Roboter an diese Stelle unter Beachtung des Offsets von -500 in beide Richtungen verschoben. Die nächsten beiden Schleifen durchlaufen alle Hindernis- und Armteilindizes. Innerhalb dieser wird mithilfe der Funktion `collide()` abgefragt, ob für die gegebenen Gelenkwinkel das Hindernis `k` mit dem Armteil `l` kollidiert. Ist das der Fall, so wird das Feld `cspace[height - 1 - j][i]` um den um 1 inkrementierten Hindernisindex inkrementiert. Auf diese Art auf die Felder von `cspace` zuzugreifen ist nötig, da zwar `i` im Array und X im Modell in dieselbe Richtung zeigen, `j` und Y jedoch in entgegengesetzte.

```
84      int nCollisions;
85
86      for (int i = 0; i < width; i += 1)
87      {
88          for (int j = 0; j < height; j += 1)
89          {
90              aRoboter[0].Translate((float)(i - height / 2) / 1000,
                    (float)(j - width / 2) / 1000, 0.0f);
91
92              for (int k = 0; k < nHind; k++)
93              {
94                  for (int l = 0; l < nRob; l++)
95                  {
96                      cspace[height - 1 - j][i] += (k + 1) * (int)
                            collide(&aRoboter[l], &aHindernis[k]);
97                      if (cspace[height -1 - j][i] != 0) nCollisions
                            ++;
98                      nTests++;
99                  }
100             }
101         }
102     }
103
104     // Zeit für das Aufstellen des Konfigurationsraumes ausgeben (
            in ms)
```

```
105     DWORD dwElapsed = GetTickCount() - dwStart;
106     float percentage = (float) nCollisions / nTests;
107     printf("\nBerechnung dauerte %d ms\n", dwElapsed);
108     printf("Anzahl Kollisionstests: %d\n", nTests);
109     printf("Anzahl Kollisionen: %d\n", nCollisions);
110     printf("Anteil Kollisionen: %f \n", percentage);
111
112     // Konfigurationsraum als PNG speichern
113     if (!SaveAsPNG("cspace_prismatic.png", cspace, width, height))
114         printf("Fehler beim Erzeugen der PNG Datei\n");
115     else
116         printf("PNG erfolgreich erzeugt\n");
```

Listing 1: Bestimmung des Konfigurationsraums

Die Ausgabe des Programms lautet wie folgt:

```
1 Berechnung dauerte 14586 ms
2 Anzahl Kollisionstests: 10000000
3 Anzahl Kollisionen: 2146190
4 Anteil Kollisionen: 0.214619
5 PNG erfolgreich erzeugt
```

Basierend auf den Werten, die nun in der Variablen `cspace` stehen wird diese Visualisierung des Konfigurationsraums erstellt:

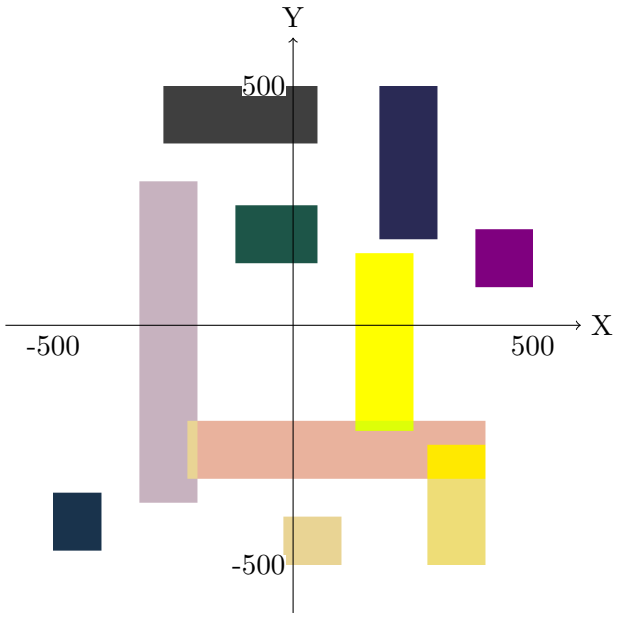

Abbildung 2: Konfigurationsraum des TT-Roboters

Es ist erkennbar, dass man die Minkowski-Summen der des Roboters und

der jeweiligen Hindernisse gebildet hat, wodurch die Hindernisse im Konfigurationsraum größer erscheinen. Das heißt, dass man mit der Koordinate, die die Position des Roboters auszeichnet nicht beliebig nah an die Hindernisse heranfahren kann, da der Roboter auch eine Ausdehnung hat.

2 Rotatorischer Roboter

Dieselbe Methode zur Bestimmung des Konfigurationsraumes soll nun auf einen RR-Roboter (2 rotatorische Freiheitsgrade, Rotationsachse z) angewandt werden.

Es werden wieder vier verschachtelte Schleifen verwendet, nur die Art, auf die die Armteile transformiert werden, unterscheidet sich zum TT-Roboter. Die zwei äußeren Schleifen durchlaufen mit den beiden Gelenkvariablen jeweils Werte von 0° bis 360°. Diese Werte werden, um den Offset von -180° bereinigt in dem zweidimensionalen Vektor `axes` gespeichert. Mit der Funktion `ForwardKinematics()` werden dann die Transformationsmatrizen der beiden Armteile bestimmt. Mit diesen sollen die als `Box` modellierten Armteile transformiert werden. Da diese Funktion auf der Vectmath-Bibliothek aufbaut, `Box` aber auf der OPCODE-Bibliothek, sind die Transformationsmatrizen nicht gleich aufgebaut. Daher müssen die auf die `Box`-Objekte anzuwendenden Transformationsmatrizen manuell aus den mittels der Vorwärtskinematik berechneten Matrizen zusammengestellt werden.

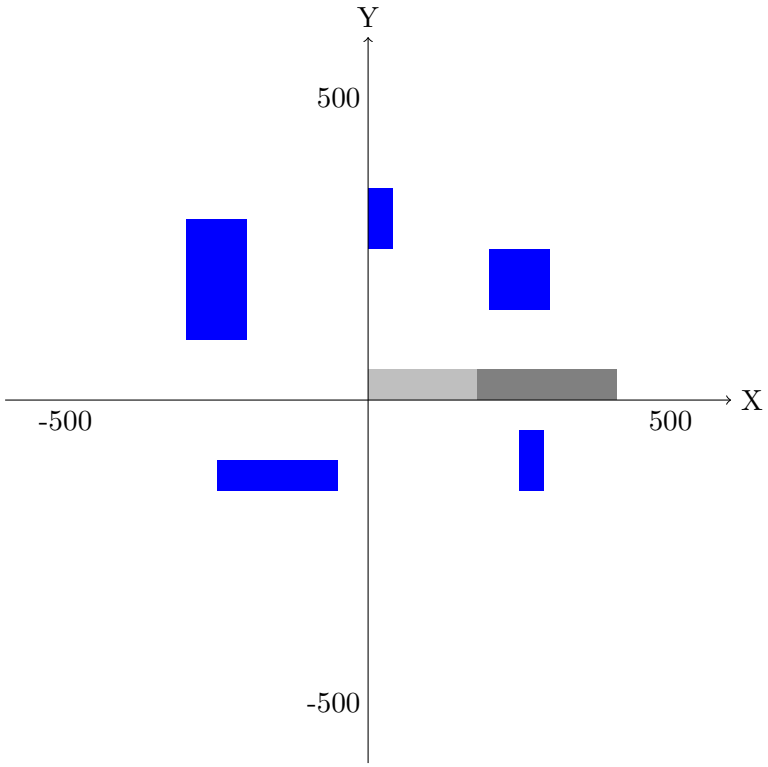

Abbildung 3: Roboter in Nullstellung (grau) und Hindernisse (blau)

```
97      aRoboter[0].Scale(0.18f, 0.05f, 0.20f);        // QUADER
            0.18000     0.05000     0.20000
98      aRoboter[1].Scale(0.23f, 0.05f, 0.20f);        // QUADER
            0.23000     0.05000     0.20000
99
100     // Startzeit
101     DWORD dwStart = GetTickCount();
102
103     // Konfigurationsraum aufbauen
104     // Ihr Code ...
105     int nCollisions = 0;
106
107     for (int j = 0; j < width; j += 1)
108     {
109         for (int k = 0; k < height; k += 1)
110         {
111             axes[0] = j - width / 2;
112             axes[1] = k - height / 2;
113
114             // Berechnen der Vorwärtstransformation
115             T1 = ForwardKinematics(1, WK2Basis, DOF2, axes);
116             // Setzen der homogenen Transformationsmatrix in
                    Opcode
117             aRoboter[0].m_Matrix.Set((float)T1[0][0], (float)T1
```

```
                [1][0], (float)T1[2][0], (float)T1[3][0],
118             (float)T1[0][1], (float)T1[1][1], (float)T1[2][1],
                    (float)T1[3][1],
119             (float)T1[0][2], (float)T1[1][2], (float)T1[2][2],
                    (float)T1[3][2],
120             (float)T1[0][3], (float)T1[1][3], (float)T1[2][3],
                    (float)T1[3][3]);
121
122         // Berechnen der Vorwärtstransformation
123         T2 = ForwardKinematics(2, WK2Basis, DOF2, axes);
124         // Setzen der homogenen Transformationsmatrix in
                Opcode
125         aRoboter[1].m_Matrix.Set((float)T2[0][0], (float)T2
                [1][0], (float)T2[2][0], (float)T2[3][0],
126             (float)T2[0][1], (float)T2[1][1], (float)T2[2][1],
                    (float)T2[3][1],
127             (float)T2[0][2], (float)T2[1][2], (float)T2[2][2],
                    (float)T2[3][2],
128             (float)T2[0][3], (float)T2[1][3], (float)T2[2][3],
                    (float)T2[3][3]);
129
130         for (int i = 0; i < nHind; i++)
131         {
132             for (int l = 0; l < nRob; l++)
133             {
134                 cspace[height - 1 - k][j] += (i + 1)  * (int)
                        collide(&aRoboter[l], &aHindernis[i]);
135                 if (cspace[height - 1 -  k][j] != 0)
                        nCollisions++;
136                 nTests++;
137             }
138         }
139     }
140 }
141
142 // Zeit für das Aufstellen des Konfigurationsraumes ausgeben (
        in ms)
143 DWORD dwElapsed = GetTickCount() - dwStart;
144 float percentage = (float) nCollisions / nTests;
145 printf("\nBerechnung dauerte %d ms\n", dwElapsed);
146 printf("Anzahl Kollisionstests: %d\n", nTests);
147 printf("Anzahl Kollisionen: %d\n", nCollisions);
148 printf("Anteil Kollisionen: %f \n", percentage);
149
150 // Konfigurationsraum als PNG speichern
151 if (!SaveAsPNG("cspace_revolute.png", cspace, width, height))
152     printf("Fehler beim Erzeugen der PNG Datei\n");
153 else
154     printf("PNG erfolgreich erzeugt\n");
```

Listing 2: Bestimmung des Konfigurationsraumes

Die Ausgabe des Programms lautet wie folgt:

```
Berechnung dauerte 3261 ms
Anzahl Kollisionstests: 1296000
Anzahl Kollisionen: 264661
Anteil Kollisionen: 0.204214
PNG erfolgreich erzeugt
```

Es ergibt sich folgender Konfigurationsraum:

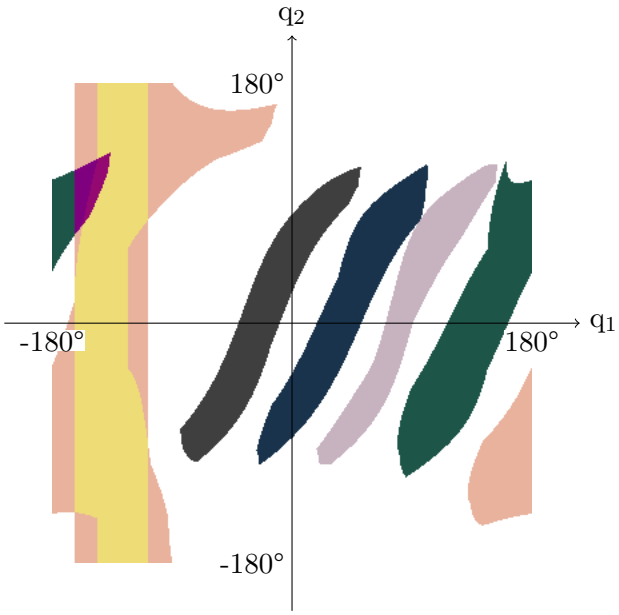

Abbildung 4: Konfigurationsraum des RR-Roboters

Es ist erkennbar, dass die Werte bei +180° und -180° in beiden Richtungen übereinstimmen, da die Armteile in diesen Fällen jeweils dieselbe Position und Rotation haben. Damit sind auch die Kollisionen dieselben.

Zwischen Werten von -164° bis -108° für q_1 ist ein für alle Werte von q_2 durchgehendes Band zu erkennen. Die bedeutet, dass für diese Gelenkstellungen eine Kollision von Armteil 1 unvermeidbar ist. Ähnliches gilt für Armteil 2. Dies lässt sich mit EasyRob leicht verifizieren:

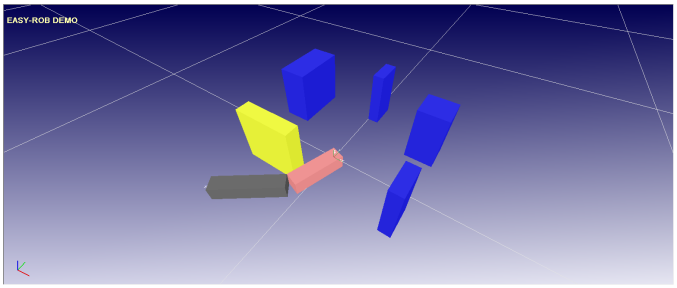

Abbildung 5: Kollision von Armteil 1