Robotic Games WS19/20

Zwischenbericht

Enrico Kaack, Robin Fleige, Laura Nell

November 26, 2019

Chapter 1

Implementierung

1.1 Collision avoidance

Um die Kollisionsvermeidung zu verwirklichen, wird die Momentangeschwindigkeit des Roboters durch einen Subscriber ausgelesen. Zusätzlich sind auf die gleiche Weise die Messwerte der Soanrsensoren in Form von Abständen zum nächsten Hindernis bekannt. Für die Kollisionsvermeidung wird ein Kraftansatz gewählt, bei welchem die Kraft, welche auf den Roboter durch ein sich näherndes Hindernis ausgeübt wird, die Änderung der Linear- und Winkelgeschwindigkeit bestimmt. Dies ist in der folgenden Funktion *calculate_force(self, sonar_angles, sonar_ranges)* verwirklicht. Dabie handelt es sich bei sonar_angles um die Winkel, in welchen die Sonarsensoren vom Roboter abstrahlen und bei sonar_ranges um den jeweils zurückgegebenen Abstand.

```
1 def calculate_force(self, sonar_angles, sonar_ranges):
2   sum = np.zeros(2)
3
4   for i, sonar_range in enumerate(sonar_ranges):
5       if sonar_range == 0.0:
6           rospy.logerr('Catched␣Zero')
7           sonar_range = 1e-12
8       vec = np.array([1/sonar_range * np.cos(sonar_angles[i]), 1/
    sonar_range * np.sin(sonar_angles[i])])
9       sum += vec
```

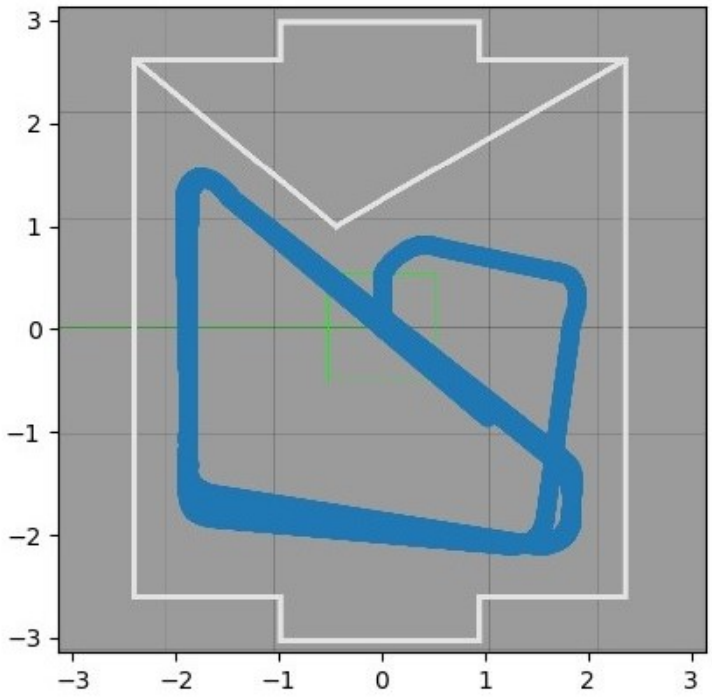
```
10
11  sum *= (-1)
12  sum_dist = np.linalg.norm(sum)
13
14  if sum_dist < 10.0:
15      return np.zeros(2)
16
17  phi = -np.arctan2(sum[1], sum[0])
18  force = np.zeros(2)
19  force[0] = np.clip(-sum[0] / 20, 0, 1)
20  force[1] = np.clip(phi, -np.pi/2, np.pi/2)
21
22  if -force[1] == self.last_angle:
23      force[1] = self.last_angle
24
25  self.last_angle = force[1]
26  return force
```

Um die auf den gesamten Roboter wirkende ´Kraft´ zu bestimmen, wird die Resultierende aus allen von den Sonarsensoren aufgezeichneten Abständen bestimmt. Dazu werden diese in der for-Schleife ab Zeile 4 zunächst in ihre x- und y-Komponenten unterteilt und diese im Array *sum* aufsummiert. Da die ´Kraft´ entgegen des Roboters wirken soll, wird dieses negiert und anschließend der auf den Roboter wirkende Vektor mithilfe der Funktion *numpy.linalg.norm()* berechnet.
Sollte der resultierende Abstand zu einem Hindernis noch größer als 10m sein, wird der Fahrtweg des Roboters nicht verändert, es wird ein Nullarray zurückgegeben (Zeile 14f.).
In jedem anderen Fall wird das Array *force[ ]* erstellt und zurückgegeben, wobei *force[0]* die Änderung der Linearbeschleunigung beinhaltet und *force[1]* die Änderung der Winkelbeschleunigung. Die Berechnung der Änderung der Linear- und Winkelbeschleunigung erfolgt mithilfe der Funktion *numpy.clip(a, a_min, a_max)*, welche den Wert von a auf das Intervall [a_min, a_max] beschränkt. Liegt der Wert a unterhalb a_min, wird a_min zurückgegeben, liegt a überhalb a_max, wird a_max zurückgegeben.
Da die Änderung der Lineargeschwindigkeit an den Roboter in der Form 1 - Δx übermittelt wird, wird der x-Anteil des resultierenden Vektors / 20 hierfür auf

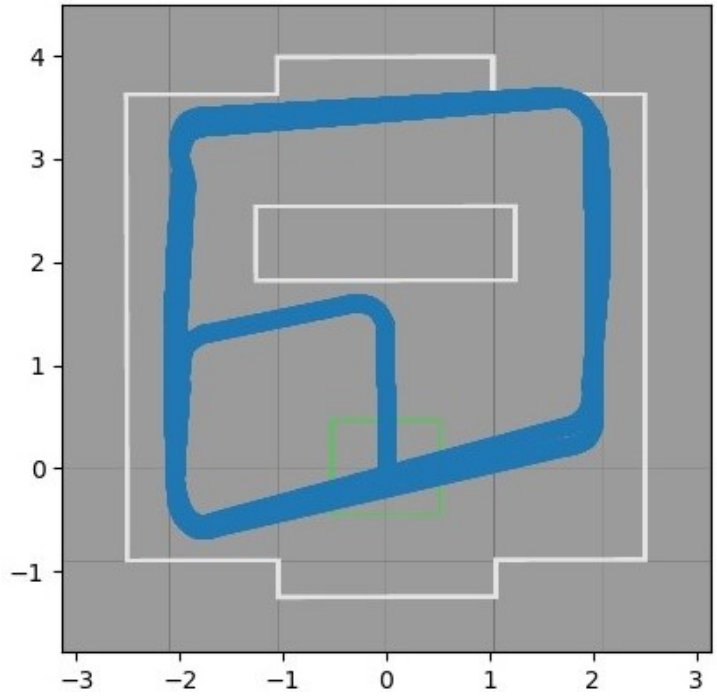
das Intervall [0,1] begrenzt, da der Roboter nicht rückwärts fahren soll (negative Linearbeschleunigung) (Zeile 19). Damit wird die Lineargeschwindigkeit nicht geändert, wenn der Wert x/20 oberhalb von 1 liegt, also der Abstand des Roboters zum Hindernis in x-Richtung noch groß genug ist, und maximal geändert, sollte der Wert negativ sein. Dieser Fall sollte allerdings nie eintreten, da dies bedeuten würde, dass der Roboter bereits an/hinter der Wand steht.
Die Änderung der Winkelbeschleunigung wird über den negativen arctan2 des resultierenden Vektors, begrenzt auf [-$\pi/2$, $\pi/2$], bestimmt (Zeile 20). Dabei sind die Maximalwerte so gewählt, dass sich der Roboter in einem Schritt nicht zu viel dreht, sondern dies auf mehrere Schritte aufgeteilt wird.
Sollte der Roboter genau senkrecht auf eine Ecke zufahren, kann der Fall auftreten, dass die resultierende ´Kraft´ genau senkrecht auf den Roboter wirkt bzw. dadurch die Änderung der Winkelbeschleunigung zwischen beiden Seiten alterniert, sodass der Roboter sich nicht befreien kann. Dies wird abgefangen, indem die Änderung der Winkelbeschleunigung in diesem Fall auf den positiven Fall festgesetzt wird (Zeile 22f.). Die zurückgegebenen Werte werden dem Roboter mithilfe eines Publishers übermittelt.

Die Funktionalität des implementierten Programms wird anhand der zur Verfügung gestellten Funktion *create_visualisation* geprüft und visualisiert. In den folgenden Abbildung sind dessen Ergebnisse gegenübergestellt.

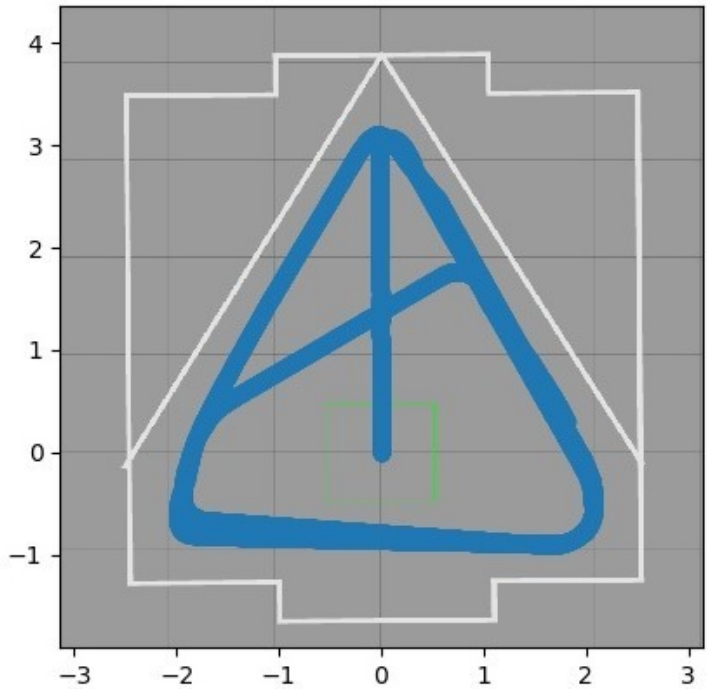
Visualisieriung Welt 1



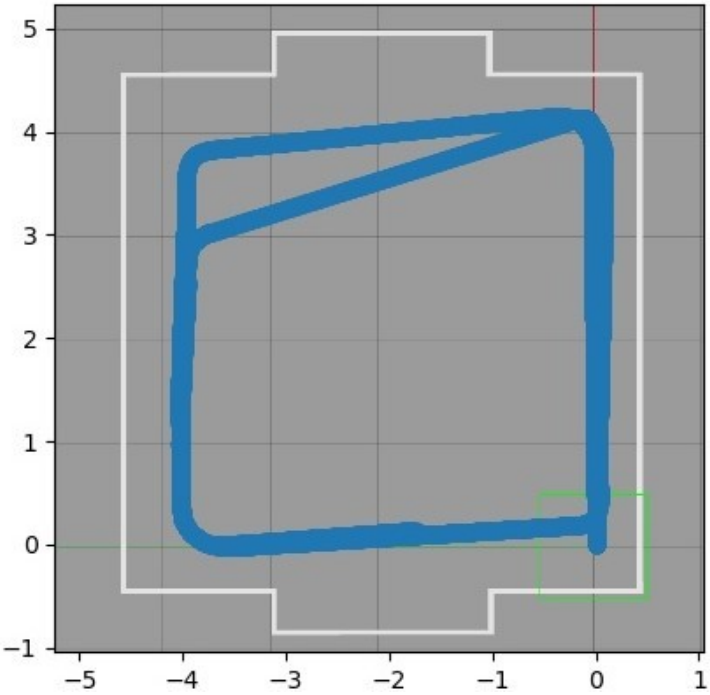
Visualisieriung Welt 2



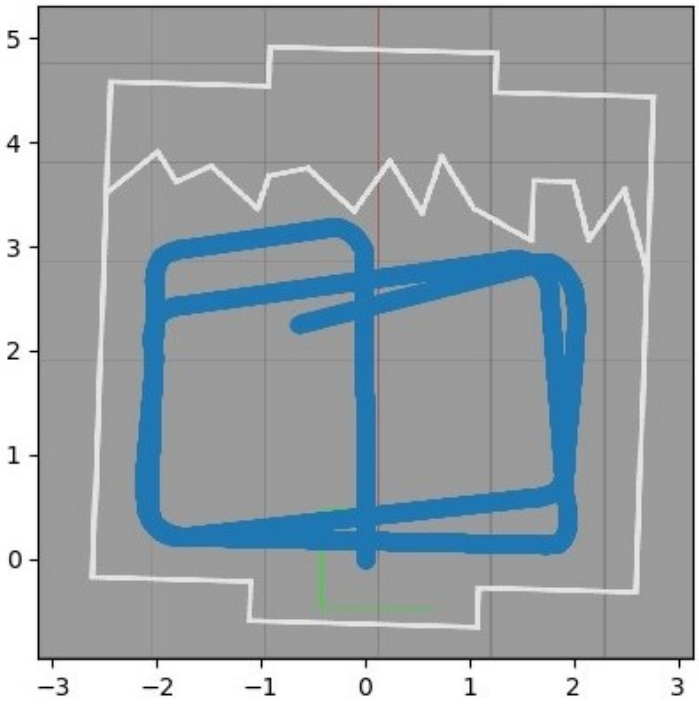
Visualisieriung Welt 3



Visualisieriung Welt 4



Visualisieriung Welt 5



Diskussion → Verbesserungen?

1.2 Homing

```
rospy.Subscriber("dead_reckoning", Pose, self.callback)
```

```
def callback(self, pos):
  self.orientation[0] = pos.orientation.z
  self.position[0] = pos.position.x
  self.position[1] = pos.position.y
```

```
def closed_loop(position,orientation,target):
  output=Twist()

```

```
4   target_rel_x = target[0] - position[0]
5   target_rel_y = target[1] - position[1]
6
7   dir_x = np.cos(orientation[0]-np.pi/2)*target_rel_x + np.sin(
     orientation[0]-np.pi/2)*target_rel_y
8   dir_y = np.cos(orientation[0]-np.pi/2)*target_rel_y - np.sin(
     orientation[0]-np.pi/2)*target_rel_x
9
10  if np.abs(np.arctan2(dir_x, dir_y)) < np.pi/4 and dir_x*dir_x+
     dir_y*dir_y > 0.01:
11      output.linear.x = 0.5
12  else:
13      output.linear.x = 0
14  if np.abs(np.arctan2(dir_x,dir_y)) > np.pi/30:
15      output.angular.z = np.arctan2(dir_x, dir_y)/3
16
17  return output
```

1.3 Free Space

Das Free Space Verhalten soll dafür sorgen, dass der Roboter in Richtung der freien Fläche im Raum fährt. Die zur Verfügung stehenden Informationen beschränken sich dabei auf die vorhandenen acht Ultraschall Sensoren und deren Entfernungsmessung.
Der Algorithmus prüft, ob die größte Entfernung links oder rechts der Hauptachse gemessen wurde und dreht den Roboter entsprechend nach rechts oder links mit einer konstanten Drehrate.
Bei diesem Ansatz werden an zwei Stellen Probleme erwartet. Zum einen wird der Sensor mit der maximalen Entfernung zur Bestimmung der Drehrichtung benutzt. Wenn dieser Sensor allerdings einen Gang misst, der vom Roboter weg verläuft, würde sich der Roboter in diese Richtung orientieren. Dieses Verhalten wäre dementsprechend falsch, da der Gang nicht der freien Fläche entspricht. Um dem entgegen zu wirken, könnte man die Sensorwerte von jeweils der rechten und linken Seite mitteln und darauf basierend die Richtungsentscheidung treffen. Im späteren Schritt der Fusion wird der Free Space Teil allerdings nur eine unterge-

ordnete Rolle spielen, sodass dieses Problem nicht in dieser Form auftritt.
Eine zweite Schwäche liegt darin, dass eine konstante Drehrate in die jeweilige Richtung gesetzt wird. Eine Auswirkung hiervon wäre, dass bei zu hoher Geschwindigkeit der Roboter gegen ein Hindernis fahren könnte, da die konstante Drehrate nicht auf die Lineargeschwindigkeit angepasst ist und der Kurvenradius demnach größer wird. Eine Lineargeschwindigkeit abhängig von der Entfernung des Sensors mit der niedrigsten Entfernung würde dies beheben. Sollte in der Fusion dies zu einem Problem führen, wird diese Idee entsprechend aufgegriffen. Da allerdings die Kollisionsvermeidung eine höhere Gewichtung in der Nähe einer Wand erhalten wird, sollte dies nicht zu einem Problem führen.

Verhaltensfusion