Roboterlabyrinth

Ein Projekt von Dominic Braun, Henry Kusch, Robin R., Zehra

Einleitung

Wie kann man bestimmte Daten, Dateien oder auch Wege innerhalb eines Systems oder einer Karte finden? Die komplexen Arbeitsweisen dieser Algorithmen wurden über Jahre ausgiebig von Mathematikern, Informatikern und Theoretikern auf Basis von mathematischen Grundlagen ausgearbeitet, verfeinert oder neu konzipiert.

Für die meisten User sind heutzutage dennoch die Such- und Findungskonzepte der Informatik heutzutage nicht verständlich genug. Zwar nutzen Sie nahezu täglich diese Konzepte, entweder direkt in der Form von Navigationsgeräten oder indirekt durch nutzen von Paketdienstleistern.

Im Rahmen des Moduls “lab:prepare” soll das Projekt Robo-Lab die Funktionsweise von verschiedenen und grundlegenden Suchalgorithmen wie A* oder Dijkstra dargestellt und für die nicht-technische Zielgruppe verständlich visualisiert werden.

Dafür soll der von Anki hergestellte Cozmo-Roboter die unterschiedlichen Algorithmen implementiert bekommen, um so eigenständig auf einer simplifizierten Vektorkarte der Technischen Universität Berlin den kürzesten Weg zwischen zwei Punkten A und B zu finden. Hierbei sollen die Sensoren und die Kamera des handflächen großen Roboters helfen, sich auf der erstellten Karte zu orientieren und zu navigieren.

Für die Zielgruppe sollen mehrere Optionen zum aktiven Arbeiten möglich sein: Angeben, was die Start- und die Zielpunkte sind Evtl. Auswahl des zu benutzenden Algorithmus oder Erstellen eigener Wettkampf-Modus: zwei Cozmo, die auf Basis von verschiedenen Algorithmen arbeiten, laufen gegeneinander um die Wette

Somit wollen wir vielerlei Möglichkeiten schaffen, diese Konzepte mit Spaß und Leichtigkeit für jedes Alter anschaulich und interaktiv darzustellen.

Theorie

Viele Probleme der Graphentheorie können mithilfe von Suchalgorithmen effizient gelöst werden. Beispiele für diese Probleme sind das Problem des Handlungsreisenden, die Berechnung kürzester Pfade und die Konstruktion eines minimalen Spannbaums. Für unser Projekt werden solche Suchalgorithmen mittels eines fahrenden Roboters veranschaulicht. Auf einer Karte kann die Effizienz verschiedener Algorithmen auf diese Weise visuell verglichen werden. Die entsprechenden Algorithmen sind zum Beispiel Bellman-Ford Algorithmus, Dijkstras Algorithmus oder A* Algorithmus, die als Erweiterungen der Algorithmen für die Suche in Bäumen gesehen werden können. Einfacher zu implementierende Algorithmen umfassen dabei beispielsweise die bekannte Tiefen- und Breitensuche.

Zusätzlich muss das Kamerabild so verarbeitet werden, sodass die schwarze linie eindeutig identifiziert werden kann.

Bauplan

Da es sich hierbei um einen fertig gebauten Roboter handelt, der im Handel käuflich zu erwerben ist, können wir keinen eigenen Bauplan vorzeigen. Weitere Infos zum Roboter sind auf der eigenen Website unter https://www.digitaldreamlabs.com/ zu finden.

Zu der SW Entwicklung haben wir uns die Dokumentation und das reichlich vorhandene Onlinematerial (Tutorials, Git-Repos etc.) angeschaut.

Material

- Cozmo

- Handy mit Cozmo App

- Karte

Aufbau

In dem Bild ist der Cozmo zu sehen, wie er exemplarisch einer Linie folgen soll.

![cozmo_linie_karte](uploads/edc432a0c545c43cb22b1be1c9ae507b/cozmo_linie_karte.jpg)

Im späteren Verlauf des Projektes soll es sich dann nicht nur um eine Linie handeln, sondern um ein verzweigtes Netz von Wegen gedruckt auf einer Karte. Hierbei stellen wir uns eine Art von Vektorkarte vor, die dieses Netz von Wegen aufzeigen soll. Diese kann ausgedruckt werden oder mittels durchsichtiger Folie variabel gestaltet werden. Es ist ein Beispiel für eine Karte der Umgebung der TU gezeigt. Abhängig von den Fähigkeiten des Cozmo werden Farbe, Glättung, Winkel und Breite der Linien angepasst.

![example-card-vektor](uploads/66b47d3c3b4c79f41159dccf224a077d/example-card-vektor.png)

Start Anleitung:

- Handy mit dem PC verbinden - Cozmo anschalten und mit dem WLAN des Cozmos verbinden - App starten und auf verbinden drücken → warten bis Cozmo verbunden - In den einstellungen der App (oben rechts) muss die SDK eingeschaltet werden - Python Programm starten mittels

```
./linefollow.py
```

Zwischenstand

Das aktuelle Problem besteht darin, Cozmo eine Linie folgen zu lassen. Der Ansatz ist das Nutzen eines bereits erhältlichen Scriptes (line_follower.py) und dieses mit eigenen Erweiterungen auf dem Cozmo zu implementieren. Der nächste Schritt ist dann, die Suchalgortihmen und den line_follower so zu verbinden, dass der Cozmo die Linien abfährt, dabei die Wege mappt und anschließend den schnellsten Weg berechnet. Hierbei sollen die Algorithmen eigenständig implementiert werden, aber über eine Schnittstelle mit line_follower verbunden werden können. Probleme bestehen darin, die Skripte auf den Cozmo zu übertragen und korrekt auszuführen.

Verbesserungen für Version

(nach 09.Juli.2021)

Es besteht die Möglichkeit eine direkte verbindung über den PC, ohne die mobile App, mit dem Cozmo aufzubauen. Dies ist ein open Source Projekt (PyCozmo) von privaten Entwicklern und bietet fast die gleichen funktionen, wie die Cozmo SDK.

Digitalisierung

Da eine Fertigstellung nicht mehr zu realisieren war, wurde der Ansatz verfolgt, die Algorithmen virtuell darzustellen:

main.py:

```
import pygame,sys,node
from pygame.locals import *
import pygamemenu

pygame.init()
disp=pygame.display.set_mode\(\(640,640))
items = [„Algorithm 1“,„Algorithm 2“,„Algorithm 3“,„exit“]
nodelist = node.grow(64,64)
pixel = pygame.Surface\[1)
pixel.fill\[2)
for i in nodelist:
disp.blit(pixel,(i.Position[0]*10,i.Position[1]*10))
choice = 0
while True:
pygame.time.delay(100)
pygame.display.update()
for event in pygame.event.get():
```

```
if event.type==QUIT:
pygame.quit()
sys.exit()
```

node.py:

```
from random import randint as rand
class Node:
def \_\_init\_\_(self, position):
self.Position = position
self.Neighbours = [None, None, None, None]
# 0
#3 1
# 2
def \_\_eq\_\_(self,other):
return self.Position == other.Position
def \_\_repr\_\_(self):
return str(self.Position[0])+','+str(self.Position)
def grow(x,y):
nodelist=[]
Start_node = Node\[3)]
nodelist.append(Start_node)
cur = Start_node
is_valid = lambda a,b: (a<x and b<y and a>=0 and b>=0)
i = 0
while i < (x*y2):#creates a random maze
cur = nodelist[rand(0,len(nodelist)-1)]#i know randchoice is a thing
r = rand(0,3)
nposx,nposy=cposx,cposy = cur.Position
if r == 0: nposy += -1
if r == 2: nposy += +1
if r == 1: nposx += +1
if r == 3: nposx += -1
if not is_valid(nposx,nposy):
continue
new = Node\[4)]
if new in nodelist:
continue
'
Es werden 4 neue Knoten erstellt, die später wieder gelöscht werden. Eine
Hashmap wäre hier effizienter.
'
getnode = lambda a,b : nodelist[nodelist.index(Node\[5)])]
otherlist=[]
try:
otherlist.append(getnode(nposx,nposy-1))
except ValueError:pass
try:
otherlist.append(getnode(nposx+1,nposy))
except ValueError:pass
```

```
try:
otherlist.append(getnode(nposx,nposy+1))
except ValueError:pass
try:
otherlist.append(getnode(nposx-1,nposy))
except ValueError:pass
if len(otherlist)!=1:
continue
other=otherlist[0]
if other.Position[0]-1 == nposx:
new.Neighbours[1]=other
other.Neighbours[3]=new
if other.Position[0]+1 == nposx:
new.Neighbours[3]=other
other.Neighbours[1]=new
if other.Position[1]-1 == nposy:
new.Neighbours[2]=other
other.Neighbours[0]=new
if other.Position[1]+1 == nposy:
new.Neighbours[0]=other
other.Neighbours[2]=new
nodelist.append(new)
i+=1
print(i,'of',x*y2)
return nodelist
```

Quellenverzeichnis

- [libcozmo](https://github.com/vinitha910/libcozmo) - [cozmosdk](http://cozmosdk.anki.com/docs/index.html) - [Line-follower](https://github.com/okoeth/cozmo-linefollow) - [pycozmo](https://pypi.org/project/pycozmo/0.8.0/) - [Cozmo Functions](https://pycozmo.readthedocs.io/en/stable/external/functions.html) - [Cozmo Robot, a distant relative of Vector: Teardown](https://www.microcontrollertips.com/teardown-anki-cozmo-vector/) - [Download Python example scripts that use the Cozmo SDK.](http://cozmosdk.anki.com/docs/downloads.html) - [To use the Cozmo SDK](http://cozmosdk.anki.com/docs/initial.html) - [mithilfe eines mikrocontrollergesteuerten RGB-LED-Steifens verschiedene Sortieralgorithmen](https://gitlab.tu-berlin.de/LEDSort/Sort_Box) - [Rosettacode](https://rosettacode.org/wiki/Dijkstra%27s_algorithm)

1)
10,10
2)
200,200,200
3)
0,0
4)
nposx,nposy
5)

a,b

From:
http://www.labprepare.tu-berlin.de/wiki/ - **Project Sci.Com Wiki**

Permanent link:
http://www.labprepare.tu-berlin.de/wiki/doku.php?id=ss21:roboterlabyrinth&rev=1659396684

Last update: **2022/08/02 01:31**