Artomat Simulation

Nils Egger

13.08.2019

1 Grundidee

Als wir auf die Idee kamen einen Roboter zu bauen und dazu eine Steuerungssoftware zu schreiben stellte ich mir die Frage wie wir dies am besten Parallel machen. Meiner Meinung nach kann man Software besser entwickeln wenn man sie auch gleich ausprobieren kann. In unserem Fall wäre dies die Kamera auf den aufgehängten Roboter mit den Erkennungspunkten zu richten. Mit diesen Gedanken im Kopf kam ich auf die Idee eine Simulation zu schreiben während David an unserem Roboter arbeitet.

2 Bibliotheken

Für die Simulation gebrauche ich Tkinter für die Darstellung. Im Nachhinein war dies vermutlich ein fehlentschluss, dies nicht weil die Bibliothek die Simulation schlecht darstellt, sondern weil sie die Darstellung nicht einfach in eine Pixel Matrix kopieren kann. Müsste ich nochmals eine auswahl treffen, so würde diese bei OpenCV liegen, denn mit OpenCV hätte ich nur kleine Performance Verluste beim Transport des Bildes zum Navigationsalgorithmus gehabt. Dies wusste ich im voraus nicht da ich weder mit OpenCV oder Tkinter bekannt war, in Zukunft wird dies aber ein No-Brainer.

```
import io
from tkinter import *
from datetime import datetime
import numpy
from PIL import Image
```

```
from sim_objects.object import Object
from sim_objects.simulation import Simulation


class Window(Frame):

    def __init__(self, master, simulation: Simulation):
        Frame.__init__(self, master)
        self.destroyed = False
        self.master = master
        self.master.bind_all('<KeyPress>', self.key_press)
        self.master.bind_all('<KeyRelease>', self.key_release)
        self.keys = {}
        self.canvas = None
        self.simulation = simulation
        self.delta_time = 0
        self.last_frame_datetime = datetime.now()
        self.pause = False
        self.ps_frame = None
        self.on_destroy = None
        self.init_window()

    def init_window(self):
        self.master.title("Spray Testbed")
        self.pack(fill=BOTH, expand=1)
        self.canvas = Canvas(self)
        self.canvas.pack(fill=BOTH, expand=1)

    def key_press(self, event):
        if event.keysym == 'space':
            pass
        else:
            self.keys[event.keysym] = True

    def key_release(self, event):
        if event.keysym == 'space':
            self.pause = not self.pause
        else:
            self.keys[event.keysym] = False
```

```
def frame(self):
    self.delta_time = datetime.now().timestamp() -
        self.last_frame_datetime.timestamp()
    self.last_frame_datetime = datetime.now()

    if not self.pause:

        speed = 50

        if 'w' in self.keys and self.keys['w']:
            self.simulation.spin_left_motor(-speed *
                self.delta_time)
        if 's' in self.keys and self.keys['s']:
            self.simulation.spin_left_motor(speed *
                self.delta_time)

        if 'Up' in self.keys and self.keys['Up']:
            self.simulation.spin_right_motor(-speed *
                self.delta_time)
        if 'Down' in self.keys and self.keys['Down']:
            self.simulation.spin_right_motor(speed *
                self.delta_time)

        if 'q' in self.keys and self.keys['q']:
            self.simulation.spray()

        self.canvas.delete("all")
        self.simulation.draw(self.canvas,
            delta_time=self.delta_time, window=self)

    self.ps_frame = self.canvas.postscript(colormode='color',
        pagewidth=Object.CANVAS_WIDTH-1,
        pageheight=Object.CANVAS_HEIGHT-1)
    self.master.after(16, self.frame)

def get_frame(self):
    return
        numpy.array(Image.open(io.BytesIO(self.ps_frame.encode('utf-8'))))
        if self.ps_frame is not None else None
```

```
    def set_on_destroy_callback(self, callback):
        self.on_destroy = callback

    def destroy(self):
        self.destroyed = True

        if self.on_destroy is not None:
            self.on_destroy()

        super().destroy()
```