Szimulált robot raj vezérlése

Patka Zsolt-András
Számítástechnika III BSc
Sapientia EMTE

2019.09.02

Contents

Dolgozat célja, téma kiírás

A dolgozat célja egy olyan osztott algoritmus kifejlesztése, melynek segítségével vezérelni lehet egy homogén robot rajt. [. . .] Mit kell megvalósítani a dolgozatban:

1. Tanulmányozni kell a robot szimulátorok adta lehetőségeket.
2. El kell készíteni egyetlen robot számára a vezérlő szoftvert. A többi robot ugyan ezt fogja használni.
3. A robot az őt körülvevő világot a szenzorjai (pl. távolság szenzor) alapján érzékeli. Kell készíteni egy szoftveres modult, mely a szenzor adatok alapján információt nyer ki és tárol a világról.
4. Ki kell dolgozni egy akadály kerülő stratégiát.
5. El kell készíteni egy módszert, mely a robot rajt összetartja(lásd *H. Rezaee and F. Abdollahi: A Decentralized Cooperative Control Scheme With Obstacle Avoidance for a Team of Mobile Robots*).

6. Ugyanakkor ki kell dolgozni egy kommunikációs szoftvert, mely lehetővé teszi a robot-robot, illetve robot-központ információ cserét. A központ jelen esetben kizárólagosan az információ begyűjtésére szolgál. A robotok pedig a kapott információt csupán továbbítják a központhoz közelebb álló robot fele.
7. A robot raj működését tesztelni kell az ARGOS és/vagy V-REP szimulációs környezetben.

Diploma dolgozat téma kiírás 2019, Villamosmérnöki Tanszék, Szántó Zoltán

Az imént felsorolt pontokból a nyár során sikerült a negyedik pontig eljutni.

Robot szimulátorok adta lehetőségek tanulmányozása

Az első lépés az a szimulátor szoftver kiválasztása, amelyben az osztott algoritmus fejlesztve és tesztelve lesz. Két szimulátor között van lehetőség választani:

1. V-REP
   - Támogatott nyelvek robot kontrollerlogikájának megírására: C/C++, Python, Java, Lua, Matlab, Octave
   - Rendkívül valósághű
   - Ingyenes, van fizetős verzió is
   - Robotrajokra nincs optimalizálva
   - Aktívan fejlesztik, jelennek meg új verziók
   - Nagy a felhasználók száma
2. ARGoS
   - Támogatott nyelvek robot kontrollerlogikájának megírására: C/C++ és Lua
   - Nagyobb az absztrakciós szint, nem annyira valósághű
   - Ingyenes
   - Robotrajokra teljesen optimalizálva van
   - Fejlesztés alatt áll, bár az új verziók nem konszisztensen érkeznek (2019 júliusán jött ki új verzió, az ezelőtti verzió 2016-os).
   - Viszonylag kevesen használják, nehezebb a hibákra megoldást találni

Döntés: ARGoS

Indoklás: Mivel az ARGoS egy nagyobb absztrakciós szintet ajánl, ezért könnyebb magára az algoritmusra, a logikára fektetni a hangsúlyt.

Ez a nagyobb absztrakciós szint ahhoz is vezet, hogy egy sok robotot tartalmazó robotraj esetén a számítási kapacitás közel se olyan sok, mint a V-REP-nél.

ARGoS-al való barátkozás

Az ARGoS szimulációs szoftverhez írt programok ún. kísérletek struktúrája a következő:

- A robot kontrollerlogikáját tartalmazó C++ kód (*.cpp állományok)
- A kísérletet leíró XML szerű .argos állomány

Az ARGoS képességeinek megismeréséhez az alábbi példaprogramokat néztem át.

Fejlesztői környezet konfigurálása

A szimulációs szoftver választása után a következő nagy feladat a fejlesztői környezet konfigurálása.

Legelső lépés az ARGoS szimulációs szoftver és ehhez szükséges függősék telepítése.

Ez viszonylag fájdalommentesen zajlott. Az ARGoS szimlátor mellé le lehet tölteni példaprogramokat. Ezeket miközben próbáltam futtatni, még merültek fel problémák a függőségekkel, amelyeket meg kellett oldanom.

Ezek után egy bash szkriptet írtam, annak érdekében, hogy a robot kontrollerjeit könnyebben lehessen build-elni és futtatni.

Később hozzáadtam egy új opciót: **capture**. Az ARGoS lehetővé teszi a kísérletek rögzítését, viszont ezt úgy teszi, hogy kimenti képként a frame-eket. A **capture** feladata ezekből egy videót létrehozni, az **ffmpeg** segítségével. A szkript rövid útmutatója:

```
#!/bin/bash
# ================================================================================
# Created by: Patka Zsolt-András
# On: 13.07.2019
# This script makes it easier to clean, build, debug and run the ARGoS experiment
#
# Arguments:$1  The job to execute, can be:
#                   capture     takes the frames created from an argos capture,
#                               creates a video from them and then deletes them.
#                               filename is equal to $2
#                   build       builds the project
#                   run         runs the experiment
#                   build-run   builds the project and runs the experiment
#                   build-debug builds the project and stars the experiment in
#                               debug mode
#           $2  The name of the scene
#
# build - requires no $2
# ================================================================================
```

Fejlesztéshez a **Jetbrains CLion** fejlesztői környezetet használtam, amelyhez a robot kontrollerlogikájának debuggolását is bekonfiguráltam.

Egyetlen robot számára elkészíteni a vezérlő szoftvert

A vezérlő szoftver kifejlesztése több lépésből állt. Az első lépés egyszerűen az volt, hogy a robot képes legyen előre haladni.

Következő lépésben meg a robot képes kellett legyen egy akadály kikerülésére, ennél a lépésnél a robot egyszerűen az akadály észlelésekor visszafordult. Utolsó lépésben egy cél is hozzá volt adva a kísérlethez (egy LED-ként modellezve) és a robot ez a cél felé kellett tudjon haladni és az akadályokat kikerülni.

Ezen alcím alatt az első lépésre fókuszálok.

Az első lépés megvalósításához létre kellett hozni egy c++ header és class állományt, amely a robot kontrollerlogikáját fogja tartalmazza. Ugyanakkor szükség volt egy .argos állományra, amely a kísérlet részleteit írta le: mekkora legyen az aréna, hova legyen elhelyezve a robot, hova legyen elhelyezve az akadály, stb.

Szenzor adatok kinyerése

Az akadályok érzékeléséhez proximity szenzort használtam, a cél érzékeléséhez meg fényszenzort. Ezen szenzorokból 24 darab van és a robot körül vannak elhelyezve.

A szenzoradatokat egyszerűen el lehet érni a robot kontrollerjében. Az információ amit biztosítanak az minden szenzornak pillanatnyi érzékelésének a nagysága és egy szög.

Ebből a két komponensből létre lehet hozni egy vektort minden szenzorpontra, ez egyszerűbbé teszi az akadálykerülő stratégia kidolgozását. Szerencsére az ARGoS keretrendszer rendelkezik beépített osztállyal a vektorokkal való dolgozásra, ez nagyban megkönnyíti a fejlesztést.

Akadálykerülő stratégia kidologzása

Az akadálykerülésre elsőként a taszító-vonzó erők (push-pull forces) megoldást használtam, amely szerint a robotra az akadályok egy fajta taszítóerőként hatnak, az elérendő cél meg egy fajta vonzóerőként.

Ezzel az algoritmussal le volt fejlesztve a második és utolsó lépés is a vezérlő szoftverhez.

A taszító-vonzó erők megoldás egyszerű és viszonylag elegáns, viszont vannak olyan akadályok amiket a robot nem képes kikerülni ilyen algoritmussal. Egy más megoldás az akadálykerülésre a QLearning használata lenne a robot tanításához.

További lépések

A következőkben szükséges lesz kifejleszteni egy olyan algoritmust, amely a robot rajt képes összetartani. Ugyanakkor szükséges a QLearning tanulmányozása az akadálykerülés továbbfejlesztéséhez.

Ha ez a két részfeladat sikeresen megvan, akkor következik egy feladat választása és algoritmus implementálása ennek megoldására a robot raj segítségével.