Differenciális hajtású robot navigációja

Bognár Máté
E2I8ZE

Projektfeladat 2020

Tartalomjegyzék

Bevezetés

A feladatom egy a tanszéki robot pozíciószabályozásának megvalósítása volt.

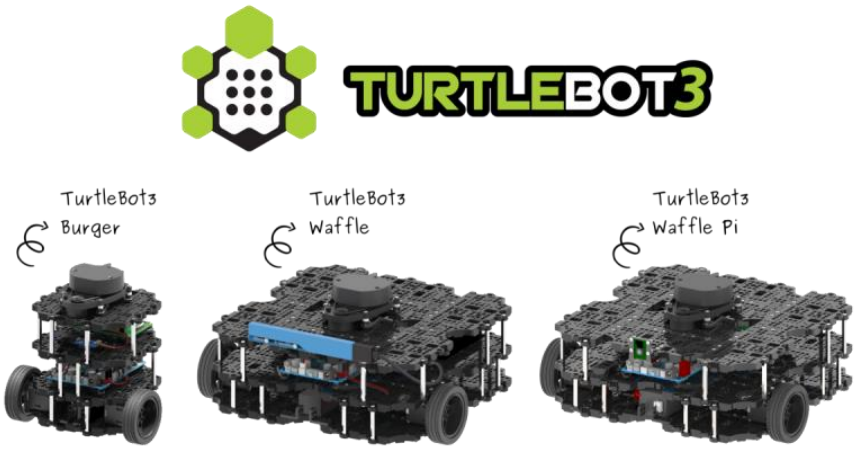
A félév során bevezetett távoktatási rend miatt a valódi robotokhoz nem volt hozzáférésem, így a projektfeladatot a hozzájuk tartotó szimulációs környezet segítségével készítettem.

A navigáció alapja egy lézeres távolságérzékelő szenzor, ami a robot útjába kerülő akadályokat érzékeli. Ennek adatai alapján hoz létre a robot egy virtuális vonalat, amin a szabályozó haladás közben rajta tartja a robotot.

Használt eszközök

Differenciális hajtású robot

A feladat során használt robot a ROBORIS cég által gyártott TurtleBot3 robotcsalád. Ennek három tagja közül a szimulációban a Burger típust használtam.



1. ábra TurtleBot3 robotok

Az általam készített algoritmus a robotok beépített szenzorjai közül csak olyanokat használ, ami mindhárom roboton megtalálható.

Egy IMU, melyen van egy háromtengelyes gyorsulásérzékelő, giroszkóp és magnetométer, a motorok encoderei és egy 360 fokos lézeres távolságérzékelő.

Az első kettő a robot egy globális koordináta rendszerben értelmezett helyzetének mérésére szolgál. Ennek kiszámítása automatikusan történik, a robot operációs rendszerén, röviden a ROS-on keresztül készen elérhető.

Az algoritmus szempontjából leglényegesebb szenzor a lézeres távolságérzékelő, későbbiekben LiDAR. A LiDAR a robot tetején elhelyezett körben forgó szenzor, mely a robot körül 360 fokban ad távolság információt.

Néhány specifikációja:

- Mintavételezési frekvenciája: 1,8 kHz.
- Mérési távolság tartománya: $120 - 3500\ mm$.
- Felbontása szög szerint: 1°.
- Beolvasási frekvenciája: $300 \pm 10\ rpm$.

Ennek megfelelően másodpercenként ötször ad adatot a robot körül az akadályok távolságáról egy fokos felbontással.

A robotokat hajtó szervomotorok nem ugyanazon típusúak a három roboton, azonban a beépített PID szabályozóknak köszönhetően a ROS-on keresztül csupán a robot kívánt sebességét és szögsebességét kell csak megadni, a motorok alacsony szintű vezérlése automatikus.

ROS rendszer

A ROS egy úgynevezett meta-operációs rendszer, mely egyrészt alacsony szinten kezeli a hardvereket, emellett a különböző kiegészítőivel segítséget nyújt robot irányítási programok készítésére. Tehát ellát operációs rendszer szintű feladatokat a roboton, ugyanakkor nem használható más operációs rendszer nélkül. Ez az általam használt verzió, ROS Kinetic esetében az Ubuntu 16.04-es verziója. A ROS tehát kapcsolatot teremt a robotot vezérlő program és a robot hardverei között.

A ROS rengeteg funkcióval rendelkezik, ezek közül azokat említeném meg, melyeket én használtan a projekt során.

A ROS rendszerben a legalacsonyabb egység az úgynevezett node, ez tekinthető egy futtatható programnak. A nodeok egyirányú kommunikációja topicokon keresztül történik. A topic definiálja egy adott küldendő információ méretét és típusát. A topicot egy node, az úgynevezett publisher regisztrálja, ez a node képes adatokat küldeni az adott topic-on keresztül. Más nodeok, hogy megkapják, az információt fel kell iratkozniuk a topicra, ezek a subscriber node-ok. Ilyenből több is lehet egy topic esetén.

A TurtleBot3 robotok esetén a szenzorok adataihoz és motorok meghajtásához előre definiált topicok vannak létrehozva. A szenzorok topicjaira a szenzor nodeja küldi az adatokat, a vezérlő program subscriber szerepét tölti be, a motorok topicjára a motorok nodeja subscriberként van definiálva, erre a vezérlő programnak kell, publisherként adatokat küldenie a motorok felé.

A navigációs algoritmus tesztelésére a Gazebo nevű 3D-s szimulációs környezetet használtam. A ROS-nak van kiegészítője a Gazebo-hoz, mely a TurtleBot3 honlapjáról letölthető. A kiegészítő lehetővé teszi a robot ROS-on keresztüli szimulációját. A Gazebo a fizikai szimuláción kívül modellezi a szenzorok jeleit, opcionálisan zaj hozzáadása mellett is.

Algoritmus

Az algoritmus egy egyszerű akadály elkerülés, bonyolultabb környezetben nem képes biztosan a kívánt pozícióba eljutni.

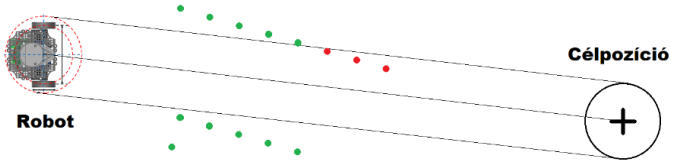
A feladat úgy indul, hogy a robot kap kívánt pozíció koordinátákat, ahova el kell jutnia. Ezt úgy teszi, hogy az akadályoktól függően köztes, célpozíciót definiál, melyet biztosan meg tud közelíteni és ilyen célpozíciókon keresztül jut el a kívánt pozícióba.

A célpozíciók között egyeneseket definiál a robot, a szabályozó menet közben ezen egyeneseken tartja a robotot.

Célpozíció keresés

A LiDAR szenzor másodpercenként ötször fordul körbe és ad távolság adatokat. Ezeket, illetve a robot pozícióadatait felhasználva definiálok célpozíciót, a globális koordináta rendszerben.

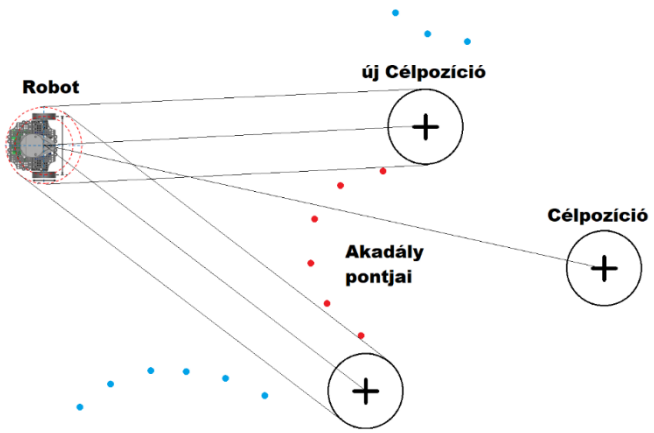
Első lépés az aktuális célpozíció, ami kezdetben a kívánt pozíció, robothoz képesti irányának meghatározása. Ezután a robot megvizsgálja, hogy ez a pozíció megközelíthető-e, azaz aktuális helyezéből egyenes vonalban, robot szélességében van-e a LiDAR által érzékelt tárgy.



2. ábra Célpozíció megközelíthetősége.

Ha nincs semmi a robot útjában, akkor elindul a robot az aktuális pozíciótól a célpozícióig.

Ha van, akkor a LiDAR által érzékelt pontokon definiál egy akadályt. Ezt úgy teszi, hogy minden pont része az akadálynak, mely a robot átmérőjénél közelebb van az akadály többi pontjához, a talált, útban lévő pontból kiindulva, azaz ha két pont között nem tud áthaladni a robot, akkor egy akadályba tartoznak.



3. ábra Új Célpozíció kiválasztása.

Ezután kiválasztja, azt az irányt, ami éppen elkerüli az akadályt, és kisebb a szöge az aktuális célpozícióhoz képest. Az új célpozíció az akadályt megkerülve van definiálva.

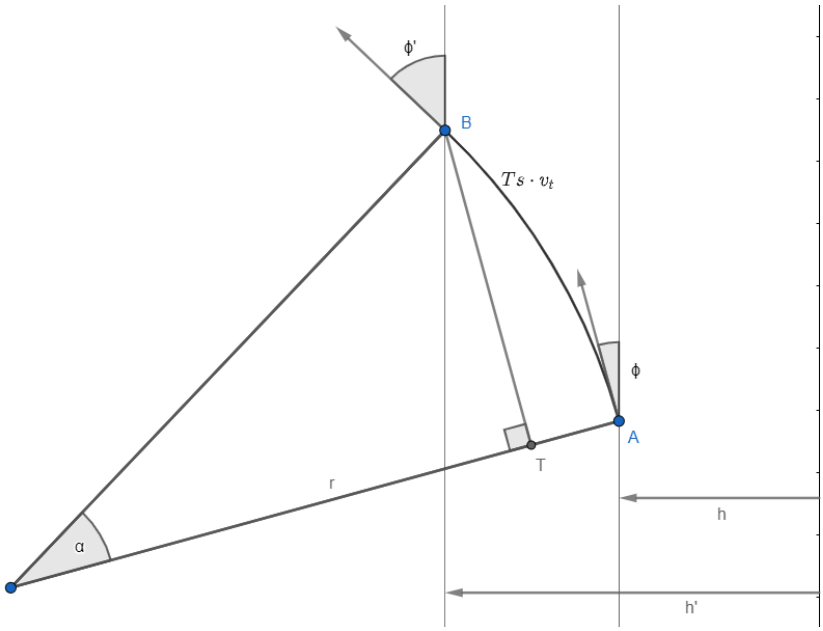
Az egyenest, amin a robot halad, az aktuális pozíció és az új célpozíció definiálja.

Szabályozó

A definiált vonalak egyenes szakaszokból állnak, ennek megfelelően a két számon tartott változó a robot vonaltól való távolsága és a robot vonalhoz képesti szöge.

Minden lépésben a robotnak meg kell adni a kívánt sebességet és szögsebességet, ezért a modell alapja, hogy két lépés között a robot a megadott, állandó sebességgel mozog előre és állandó szögsebességgel fordul, azaz állandó sebességgel körmozgást végez.

Ebből a kapott modell:



4. ábra Robot mozgásának modellje.

A 4. ábra alapján a robot:

- A pontból B pontba mozog.
- Sebessége: v_t.
- Szögsebessége: v_r.
- Az eltelt idő: Ts.
- A pontban a vonaltól mért távolság: h.
- A pontban a vonalhoz képesti szög: φ.
- B pontban a vonaltól mért távolság: h'.
- B pontban a vonalhoz képesti szög: φ'.
- A körmozgás sugara: r.
- A mozgás során átívelt szög: α, ez a robot elfordulási szöge is egyben.

Az állandó sebesség miatt, a megtett út, a kör mentén a megadott sebesség és az eltelt idő szorzata, ez egyenlő az átívelt szög radiánban vett értékének és a sugárnak a szorzatával,

$$Ts \cdot v_t = r \cdot \alpha.$$

Az állandó szögsebesség miatt, az α szög a megadott szögsebesség és az eltelt idő szorzata,

$$\alpha = Ts \cdot v_r.$$

Az előző két egyenletből kifejezve a sugarat:

$$r = \frac{Ts \cdot v_t}{\alpha} = \frac{Ts \cdot v_t}{Ts \cdot v_r} = \frac{v_t}{v_r}.$$

Az AT és BT szakaszok hossza a sugárral és átívelt szöggel kifejezve,

$$AT = r \cdot (1 - \cos(\alpha)),$$
$$BT = r \cdot \sin(\alpha).$$

Ezeket felhasználva a szükséges két paraméter új értéke kifejezhető:
A vonalhoz képesti szög:

$$\boldsymbol{\varphi' = \varphi + \alpha = \varphi + Ts \cdot v_r}.$$

A vonaltól mért távolság:

$$h' = h + \sin(\varphi) \cdot BT + \cos(\varphi) \cdot AT,$$
$$h' = h + \sin(\varphi) \cdot r \cdot \sin(\alpha) + \cos(\varphi) \cdot r \cdot (1 - \cos(\alpha)),$$
$$h' = h + \sin(\varphi) \cdot \frac{v_t}{v_r} \cdot \sin(Ts \cdot v_r) + \cos(\varphi) \cdot \frac{v_t}{v_r} \cdot (1 - \cos(Ts \cdot v_r)).$$

Ezt az egyenletet linearizálva:
Kis α esetén,

$$Ts \cdot v_r = \alpha \ll 1 \rightarrow \sin(Ts \cdot v_r) \approx Ts \cdot v_r;\ \cos(Ts \cdot v_r) \approx 1,$$
$$h' = h + \sin(\varphi) \cdot \frac{v_t}{v_r} \cdot Ts \cdot v_r + \cos(\varphi) \cdot \frac{v_t}{v_r} \cdot (1 - 1) = h + \sin(\varphi) \cdot v_t \cdot Ts.$$

Kis φ esetén,

$$\varphi \ll 1 \rightarrow \sin(\varphi) \approx \varphi;\ \cos(\varphi) \approx 1,$$
$$\boldsymbol{h' = h + \varphi \cdot v_t \cdot Ts}.$$

A megadott sebességet, v_t-t paraméterként meghagyva definiálható egy linearizált, diszkrét állapottér modell, melynek bemenete a megadott szögsebesség, v_r, kimenete a vonaltól való távolság, h. A mintavételezési idő, $Ts = 1/60\ s$.

$$\begin{pmatrix} \boldsymbol{\varphi_{k+1}} \\ \boldsymbol{h_{k+1}} \end{pmatrix} = \begin{bmatrix} \mathbf{1} & \mathbf{0} \\ \boldsymbol{Ts \cdot v_t} & \mathbf{1} \end{bmatrix} \cdot \begin{pmatrix} \boldsymbol{\varphi_k} \\ \boldsymbol{h_k} \end{pmatrix} + \begin{bmatrix} \boldsymbol{Ts} \\ \mathbf{0} \end{bmatrix} \cdot \boldsymbol{v_r}$$
$$\boldsymbol{y_k} = [\mathbf{0} \quad \mathbf{1}] \cdot \begin{pmatrix} \boldsymbol{\varphi_k} \\ \boldsymbol{h_k} \end{pmatrix}$$

A robot mozgása során a cél, hogy e modell két állapotváltozójának minél gyorsabban való nullába juttatása, ez jelenti azt, hogy a robot a vonal mentén mozog.

Ehhez terveztem állapotvisszacsatolást. Az irányíthatósági mátrix:

$$M_C = [B \quad A \cdot B] = \begin{bmatrix} Ts & Ts \\ 0 & Ts^2 \cdot v_t \end{bmatrix}, \qquad rank(M_C) = 2$$

Az irányíthatósági mátrix rangja maximális, tehát a rendszer irányítható, az állapotvisszacsatolási mátrix, az Ackermann képlettel:

$$K = [0 \quad 1] \cdot {M_C}^{-1} \cdot \phi_C(A)$$

A $\phi_C(z)$ polinomot úgy állítottam be, hogy a folytonos idejű megfelelője egységnyi csillapítású legyen, időállandója $T_f = 0{,}2\ s$, ebből a két folytonos idejű gyök:

$$s_{1/2} = -\frac{1}{T_f}.$$

A diszkrét idejű gyökök,

$$z_{1/2} = e^{s_{1/2} \cdot Ts} = e^{-\frac{Ts}{T_f}}.$$

A $\phi_C(z)$ polinom,

$$\phi_C(z) = \left(z - e^{-\frac{Ts}{T_f}}\right)^2.$$

Ezt beírva a visszacsatolási mátrix képletébe,

$$K = [0 \quad 1] \cdot \begin{bmatrix} \frac{1}{Ts} & \frac{-1}{Ts^2 \cdot v_t} \\ 0 & \frac{1}{Ts^2 \cdot v_t} \end{bmatrix} \cdot \left(\begin{bmatrix} 1 & 0 \\ Ts \cdot v_t & 1 \end{bmatrix} - \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot e^{-\frac{Ts}{T_f}} \right)^2,$$

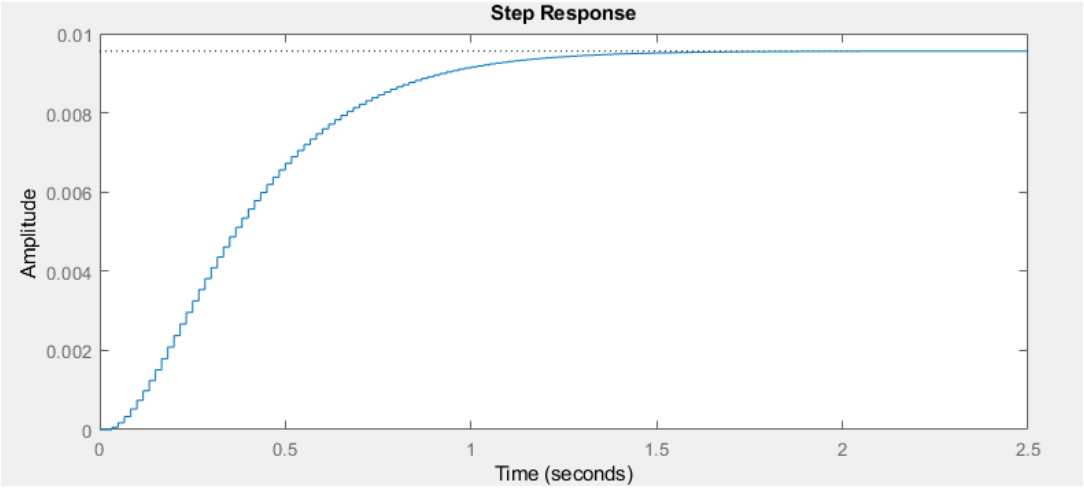
$$K = [0 \quad 1] \cdot \begin{bmatrix} \frac{1}{Ts} & \frac{-1}{Ts^2 \cdot v_t} \\ 0 & \frac{1}{Ts^2 \cdot v_t} \end{bmatrix} \cdot \begin{bmatrix} 1 - e^{-\frac{Ts}{T_f}} & 0 \\ Ts \cdot v_t & 1 - e^{-\frac{Ts}{T_f}} \end{bmatrix}^2,$$

$$K = \begin{bmatrix} 2 \cdot \frac{a}{Ts} & \frac{a^2}{Ts^2 \cdot v_t} \end{bmatrix}, \qquad ahol: a = 1 - e^{-\frac{Ts}{T_f}}$$

Ez a visszacsatolási mátrix már alkalmazható a robot irányítására, a szabályozó kimenete,

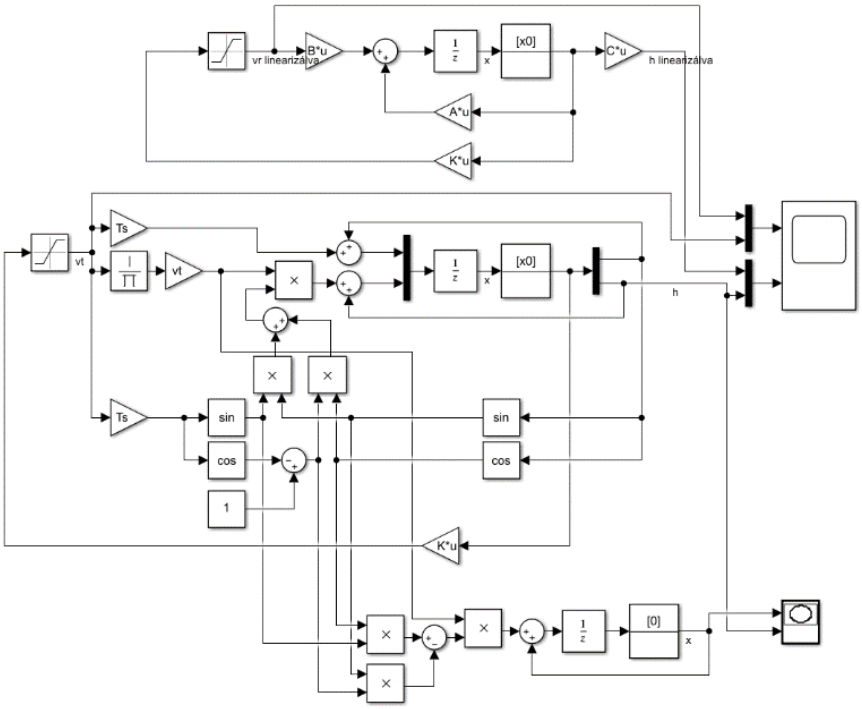
$$v_r = -K \cdot \begin{pmatrix} \varphi_k \\ h_k \end{pmatrix}.$$

A számításokat Matlab-bal elvégezve, az állapot visszacsatolt rendszer egységugrás válasza:

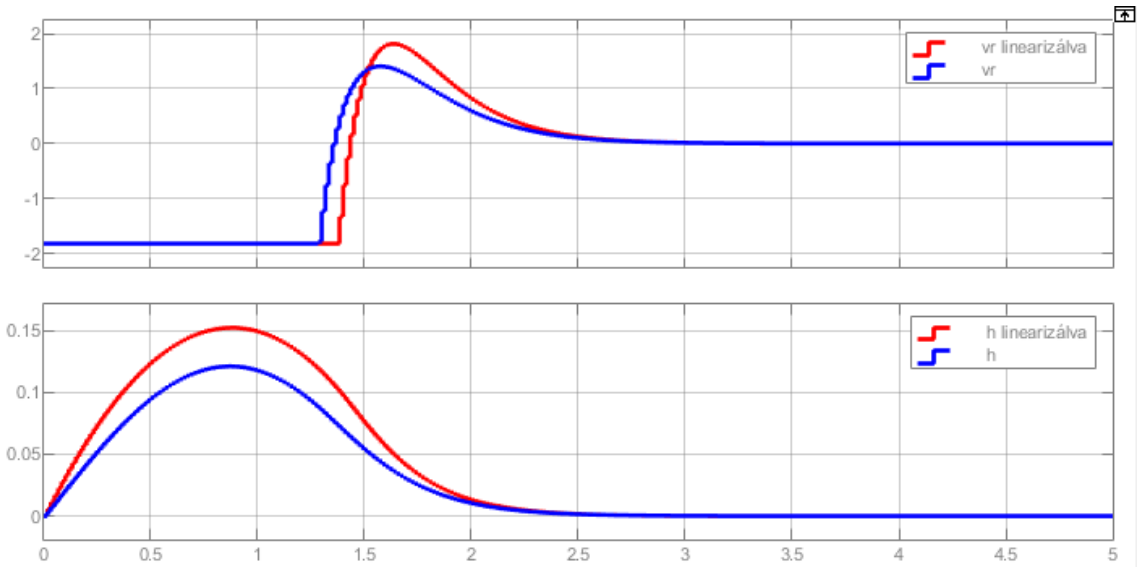


5. ábra Szabályozott rendszer egységugrás válasza.

Ezután Simulink környezetben létrehoztam a linearizált és a nemlineáris rendszer modelljét, alkalmazva az állapotvisszacsatolást, a $\begin{pmatrix}\varphi_0\\ h_0\end{pmatrix} = \begin{pmatrix}\pi/2\\ 0\end{pmatrix}$ kezdeti értékből indulva, azaz a vonalra merőlegesen indítva a robotot:



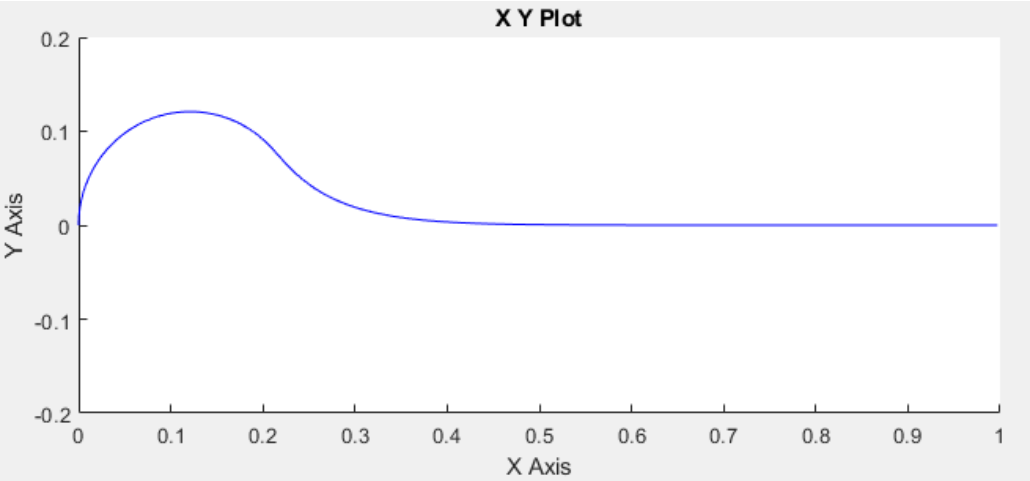
6. ábra Simulink modell.



7. ábra Simulink be- és kimenetek.

Látható, hogy a nemlineáris esetben a robot körülbelül 0,12 m-rel távolodik el a vonaltól maximálisan és nagyjából 2,5 s alatt áll rá a vonalra.

Végül kirajzoltattam a robot által a nemlineáris modellel számolt pályát, úgy, hogy a robot referencia vonala a vízszintes tengely az origóból indítva.



6. ábra Robot feltételezett pályája.

A v_t sebességet a Matlab szimulációban a robot adatlapján megadott maximális sebességgel tettem egyenlővé (0,22 m/s), hasonlóan a szögebesség bemenet is az adatlapi maximumnál van korlátozva (1,82 rad/s). Ez alapján a robot által leírt pályán a kezdeti göbületi sugár, a már ismert képlet alapján:

$$r = \frac{v_t}{v_r} = \frac{0{,}22\ m/s}{1{,}82\ rad/s} = 0{,}121\ m.$$

Ez az érték látható a feltételezett pályán is.

A ROS szimulációban a sebesség paraméter marad, ami minden lépésben a kívánt pozíciótól való távolsággal arányos a maximumáig, ez lassítja le a robotot a kívánt pozíció elérekor. Az állapotvisszacsatolási mátrix azonban ettől függ, így a visszacsatolt rendszer összességében független ettől a parmétertől.

Algoritmus részletezése

A robotra szánt programot C++ nyelven írtam, egyetlen nodeot használva.

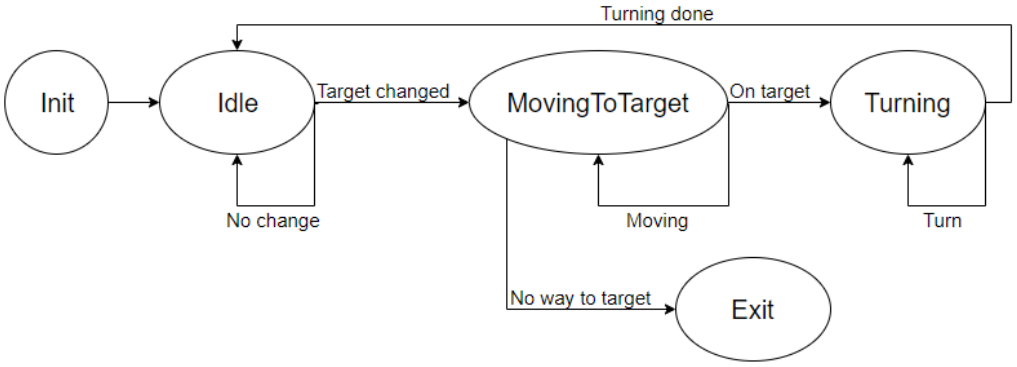
A node publisherje a motorok topicjának, a szabályozó ezen keresztül adja át a megadott sebesség és szögsebesség adatokat. Továbbá a node subsciberje a robot pozícióját egy globális koordináta rendszerben megadó, már korábban említett topicnak, a távolságérzékelő szenzor topicjának továbbá egy általam definiált topicnak, melyen keresztül a kívánt pozíció információt kapja a robot.

A programszerkezet egy inicializálási lépés után egy állapotgép, ami a szabályozó frekvenciájának megfelelően 60 Hz-es frekvencián üzemel, állapotai:

- *Idle*, ha a robot a kívánt pozícióban van.
- *MovingToTarget*, itt történik a navigáció.
- *Turning*, a kívánt pozícióban a megadott irányba fordul.
- *Exit*, kilép a program.

Az állapot átmeneteket a 9. ábra mutatja. Az inicializálást követően a robot egy ciklusba lép, a ciklus minden iterációs lépésében először kiolvassa a topicok

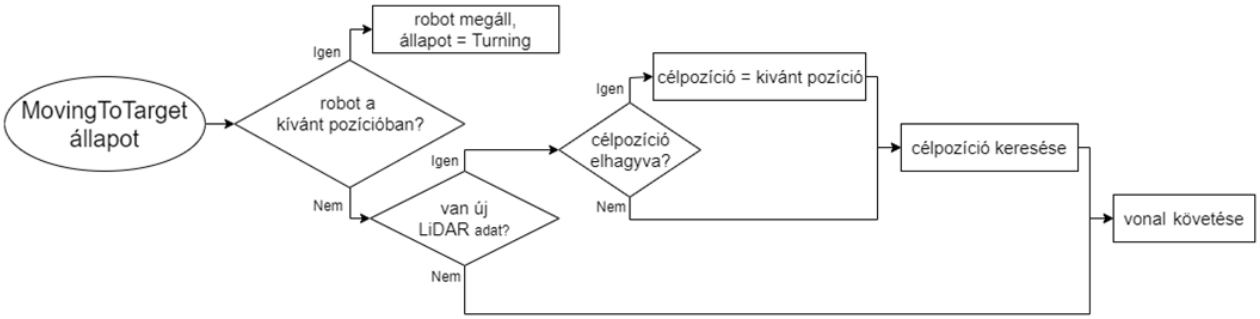
adatait, amire fel van iratkozva, majd az aktuális állapotnak megfelelő utasítások futnak le.



7. ábra Állapot átmenetek

Az *Idle* állapot az, amelyik az inicializálás után közvetlen meghívódik. Ebben az állapotban egyszerűen azt vizsgálja a robot, hogy a kívánt pozíció megegyezik-e az aktuális pozícióval. Ha igen, akkor a robot *Idle* állapotban marad a következő iterációs lépésben. Ha nem, akkor a célpozíciónak beállítja a kívánt pozíciót és átlép *MovingToTarget* állapotba.

A *MovingToTarget* állapotban először megvizsgálja a robot, hogy elérte-e már a kívánt pozíciót, ha igen, megáll a robot és átlép *Turning* állapotba.
Ha nincs a kívánt pozícióban, akkor ellenőrzi, hogy az iterációs lépésszám a LiDAR mérési frekvenciájának megfelelő, azaz kapott a robot új adatokat a szenzorból, ha nem, akkor folytatódik a navigáció.
Ha van új adat, akkor megvizsgálja, túlment-e már az aktuális célpozíción, ha igen, akkor beállítja a kívánt pozíciót a célpozíciónak.
Ezután lefut a célpozíció keresési algoritmus, mely vagy változtatás nélkül visszatér, ha a célpozíció megközelíthető, vagy a célpozíciónak beállítja az algoritmus alapján számolt újat. Ha az érzékelt akadály körbe veszi a robotot, így nem tud definiálni új célpozíciót, akkor meghívja az *Exit* állapotot.
Végül lefut a szabályozó algoritmus.



8. ábra MovingToTarget állapot

A *Turning* állapotban a robot megvizsgálja, hogy az aktuális orientációja a robotnak megegyezik-e a kívánttal, ha igen, megáll a robot és átlép *Idle* állapotba, ha nem, egy arányos szabályozót használva fordul be a kívánt szöghelyzetbe, ami a pozícióhoz hasonlóan adott az induláskor.

Az *Exit* állapot egyszerűen megállítja a robotot és leállítja a program futását.

Igyekeztem egy védelmi funkciót is beállítani, a LiDAR adatai alapján. Ha egy tárgypont túl közel kerülne a robothoz, akkor az megáll és kilép a programból. A szimulációban, azonban még ha a robot neki ment is a fizikai akadálynak a szenzor a robot méreténél nagyobb értékeket adott vissza, holott szemből a robot fizikai mérete ennél jóval kisebb. Ez a szenzor méréshatárának minimuma miatt lehet, ami az adatlap szerint *120 mm*. Így ez a funkció nem működik.

Továbbfejlesztési lehetőségek

Algoritmus szempontjából a célpozíció keresés fejlesztéséhez úgy gondolom, mindenképp valamiféle memóriával kellene rendelkeznie a robotnak a talált akadályokról, ahhoz hogy bonyolultabb környezetben is jól navigáljon. Itt felhasználhatók lehetnek különböző mesterséges intelligencia algoritmusok.

Ami a kész algoritmust illeti, a szabályozót ki lehetne szervezni külön nodeba, moduláris alkalmazás céljából.

Másik irány a szabályozó tervezésénél használt modell finomítása. Ez jelenleg nem veszi figyelembe a mozgató motorok dinamikai tulajdonságait. Ez a különbség látható a Simulink és a Gazebo szimuláció között, ugyanakkor bőven nem okoz stabilitási problémát.

Felhasznált források

http://emanual.robotis.com/docs/en/platform/turtlebot3/overview/

http://wiki.ros.org/ROS/Tutorials

http://wiki.ros.org/Books/ROS_Robot_Programming_English