UNIVERSITATEA “POLITEHNICA” DIN TIMIȘOARA FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE DEPARTAMENTUL AUTOMATICĂ ŞI INFORMATICĂ APLICATĂ

TAMAGOTCHI

PROIECT SINCRETIC I

AUTORI: **Lucian-Florin GODEANU**

Alexandra-Raluca BOGDAN

Ligia TRIF

Coordonatori: **Conf. dr. ing. Florin DRĂGAN, As. ing. Emil VOIȘAN, Andrei-Ioan PRODANIUC**

Anul III AIA - an universitar 2022 / 2023

CUPRINS

I. Introducere

Un robot mobil este o mașină controlată de software care utilizează senzori și alte tehnologii pentru a identifica împrejurimile și a se mișca în jurul lor. Roboții mobili funcționează folosind o combinație de inteligență artificială (AI) și elemente robotice fizice, cum ar fi roți, șenile și picioare. Roboții mobili devin din ce în ce mai populari în diferite sectoare de afaceri. Ele sunt folosite pentru a ajuta la procesele de lucru și chiar pentru a îndeplini sarcini care sunt imposibile sau periculoase pentru oameni [WWW 01].

Există mai multe tipuri de navigare pentru robotul mobil: manual remote or tele-op, guarded tele-op, line-following car, autonomously randomized robot, autonomously guided robot și sliding autonomy.

Un robot teleoperat (tele-op) manual este complet sub controlul unui șofer cu un joystick sau alt dispozitiv de control. Dispozitivul poate fi conectat direct la robot, poate fi un joystick fără fir sau poate fi un accesoriu pentru un computer fără fir sau alt controler. Un robot tele-op este de obicei folosit pentru a ține operatorul departe de pericol.

Un robot de tele-operație păzit are capacitatea de a detecta și de a evita obstacolele, dar în caz contrar va naviga pe măsură ce este condus, ca un robot cu tele-operație manuală. Foarte puțini roboți mobili oferă doar operațiuni de televiziune păzite [WWW 02].

Unele dintre cele mai vechi vehicule cu ghid automat (AGV) au fost roboți mobili. Acestea pot urma o linie vizuală pictată sau încorporată în podea sau tavan sau un fir electric în podea. Cei mai mulți dintre acești roboți au operat un algoritm simplu de „ține linia în senzorul central”. Nu puteau ocoli obstacolele; s-au oprit și au așteptat când ceva le-a blocat calea. Multe exemple de astfel de vehicule sunt încă vândute de către Transbotics, FMC, Egemin, HK Systems și multe alte companii. Aceste tipuri de roboți sunt încă foarte populare în societățile robotice bine cunoscute, ca un prim pas către colțurile și colțurile de învățare ale roboticii.

Roboții autonomi cu mișcare aleatorie, practic sar peste pereți, indiferent dacă acești pereți sunt detectați sau nu.

Un robot ghidat autonom știe cel puțin câteva informații despre unde se află și cum să atingă diferite obiective și/sau puncte de referință pe parcurs. „Localizarea” sau cunoașterea locației sale curente, se calculează prin unul sau mai multe mijloace, folosind senzori precum codificatori de motor, viziune, Stereopsis, lasere și sisteme de poziționare globală.

Roboții mai capabili combină mai multe niveluri de navigare sub un sistem numit autonomie de alunecare. Majoritatea roboților ghidați autonom, cum ar fi robotul de spital HelpMate, oferă, de asemenea, un mod manual care permite robotului să fie controlat de o persoană [Hu21].

Figura 1 – Robotul mobil HelpMate

II. Prezentarea temei

Tema proiectului nostru este simularea acțiunilor unui Tamagotchi cu ajutorul robotului mobil TurtleBot3 Burger.

Tamagotchi („Egg Watch”) este un animal de companie extraterestru digital de mână care a fost creat în Japonia de Akihiro Yokoi de la WiZ și Aki Maita de la Bandai. A fost lansat de Bandai pe 23 noiembrie 1996 în Japonia și în SUA la 1 mai 1997, devenind rapid unul dintre cele mai mari trend-uri de jucării de la sfârșitul anilor 1990 și începutul anilor 2000. În martie 2021, peste 83 de milioane de unități au fost vândute în întreaga lume. Majoritatea Tamagotchi sunt găzduite într-un mic joc video portabil în formă de ou, cu o interfață formată din trei butoane, Tamagotchi Pix adăugând un obturator în partea de sus pentru a activa camera.

Jucătorul poate avea grijă de animalul de companie cât de mult sau cât de puțin vrea, iar rezultatul depinde de acțiunile jucătorului. Animalele de companie au un contor de foame, un contor de fericit și un contor de dresaj pentru a determina cât de sănătos și de bine comportat este animalul de companie. Există, de asemenea, o funcție de verificare a vârstei și greutății pentru vârsta și greutatea curentă a animalului de companie. În modelele recente, jucătorul câștigă monedă numită Puncte Gotchi în timp ce joacă jocuri și poate folosi moneda

într-un magazin din joc pentru a cumpăra diferite alimente, jucării, accesorii sau chiar decorațiuni de cameră pentru animalul de companie [Ban98].

Așadar, pornind de la ideea de a îngriji un animal de companie, am abstractizat acțiunile unui Tamagotchi la acțiunile unui cățeluș. Acesta are șapte moduri de lucru:

- default():
  - sta pe loc
  - afișează "puppy waiting for your attention" în consolă
- eat():
  - detectează poza cu culoarea galben
  - executa 2 rotiri 360* pe loc
  - afișează "happy puppy eating" în consolă
- poop():
  - se execută doar dupa ce se execută eat()
  - după 5s după ce a mancat, poop() se va executa automat
  - se va mișca de 2 ori față-spate rapid
  - afișează “sad puppy wants to poo" în consolă
- play():
  - detectează poza albastră
  - aleargă rapid în cerc
  - afișează “happy puppy running" în consolă
- sleep():
  - se execută doar dupa ce se execută play()
  - după 5s după ce s-a jucat, sleep() se va executa automat
  - afișează “shhh puppy is sleeping" în consolă
- love():
  - detectează poza portocalie
  - se deplasează încet către culoare cât timp o detectează
  - afișează “puppy loves you so much" în consolă
- attack():
  - detectează poza roșie
  - urmărește foarte rapid culoarea cât timp o detectează
  - afișează “run, puppy is in attack mode" în consolă

Figura 2 – Modele Tamagotchi

III. Tehnologii utilizate

Pentru a realiza proiectul am utilizat următoarele tehnologii: TurtleBot3 Burger, Ubuntu 20.04, ROS Noetic, Python și OpenCV.

Ubuntu este o distribuție Linux bazată pe Debian și compusă în mare parte din software gratuit și open-source. Python este un limbaj de programare de nivel înalt, cu scop general. Filosofia sa de design pune accent pe lizibilitatea codului prin utilizarea unei indentări semnificative. Python este tastat dinamic și folosește garbage-collector. Suportă mai multe paradigme de programare, inclusiv programarea structurată, orientată pe obiecte și funcțională. OpenCV este o bibliotecă de funcții de programare, dezvoltat inițial de Intel, care vizează în principal viziunea computerizată în timp real.

TurtleBot3 este un robot mobil de nouă generație, care este modular, compact și personalizabil. Cu ajutorul acestui robot putem să creăm aplicații interesante pentru educație, cercetare și dezvoltare de produse. Dispozitivul dispune de:

- 360 LiDAR pentru SLAM și navigație
- Structură scalabilă
- Computer cu o singură placă (Raspberry Pi 3)
- OpenCR (32-bit ARM Cortex-M7)
- DYNAMIXEL x2 pentru roți
- Roți dințate pentru anvelopă și șenile
- Baterie Li-Po 11.1V 1.800mAh [WWW 03]

Sistemul de operare robot (ROS) este un set de biblioteci software și instrumente care ne ajută să construim aplicații robot. De la drivere la algoritmi de ultimă generație și cu instrumente puternice de dezvoltare, ROS are ceea ce avem nevoie pentru următorul nostru proiect de robotică, iar totul este open source [WWW 04].

Procesele ROS sunt reprezentate ca noduri într-o structură de graf, conectate prin muchii numite subiecte. Nodurile ROS pot transmite mesaje unul altuia prin topic-uri, pot efectua apeluri de serviciu către alte noduri, pot oferi un serviciu pentru alte noduri sau pot seta sau prelua date partajate dintr-o bază de date comună numită server de parametri. Un proces numit ROS Master face toate acestea posibile prin înregistrarea nodurilor în sine, stabilirea comunicației nod-la-nod pentru subiecte și controlând actualizările serverului de parametri. Mesajele și apelurile de serviciu nu trec prin master, mai degrabă masterul stabilește comunicarea peer-to-peer între toate procesele nodului după ce acestea se înregistrează la master. Această arhitectură descentralizată se pretează bine roboților, care constau adesea dintr-un subset de hardware de computer în rețea și pot comunica cu computerele externe pentru heavy computing sau diverse comenzi [Mu19].

IV. Ghidul programatorului

Am început dezvoltarea aplicației prin a scrie un cod în limbajul Python care inițializează nodul “puppy_node”, după care în funcție de culorile detectate de catre cameră va executa mișcări specifice publicând mesaje de tipul “twist” in topic-ul “cm1d_vel”:

```
#!/usr/bin/env python3
# IMPORTS
import time
import rospy
import cv2
from geometry_msgs.msg import Twist
# CAMERA CONFIGURATION
cap = cv2.VideoCapture(0)
cap.set(3, 640)
cap.set(4, 480)
# COLOR RANGES IN HSV
# EAT
lower_yellow = (22, 93, 0)
upper_yellow = (45, 255, 255)
# PLAY
```

```
lower_blue = (78, 158, 124)
upper_blue = (138, 255, 255)
# LOVE
lower_orange = (5, 50, 50)
upper_orange = (15, 255, 255)
# ATTACK
lower_red = (0, 100, 20)
upper_red = (10, 255, 255)
jmp = 0
eat = False
play = False
rospy.init_node('puppy_node')
try:
    while True:
        _, frame = cap.read()
        hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
        # COLOR MASKS
        # EAT
        mask_yellow = cv2.inRange(hsv, lower_yellow, upper_yellow)
        # PLAY
        mask_blue = cv2.inRange(hsv, lower_blue, upper_blue)
        # LOVE
        mask_orange = cv2.inRange(hsv, lower_orange, upper_orange)
        # ATTACK
        mask_red = cv2.inRange(hsv, lower_red, upper_red)
        cv2.imshow("camera", frame)
        # 0. DEFAULT
        if cv2.countNonZero(mask_yellow) <= 0 and cv2.countNonZero(mask_blue) <= 0 and
cv2.countNonZero(mask_orange) <= 0 and cv2.countNonZero(mask_red) <= 0 and jmp == 0:
            # if cv2.countNonZero(mask_yellow) > 0 and cv2.countNonZero(mask_blue) > 0 and
cv2.countNonZero(mask_orange) > 0 and cv2.countNonZero(mask_red) > 0 and jmp == 0:
            print("puppy waiting for your attention")
            jmp = 1
        # 1. EAT
        if cv2.countNonZero(mask_yellow) > 0:
            # cv2.imshow("detect_yellow", mask_yellow)
            # Setam publisher ul pentru cmd_vel topic
            cmd_vel_pub = rospy.Publisher('cmd_vel', Twist, queue_size=10)
            num_mov = 0
            print("happy puppy eating\n")
            while True:
                twist = Twist()
                if num_mov == 3:
```

```
            twist.linear.x = 0
            twist.angular.z = 0
            cmd_vel_pub.publish(twist)
            break
        else:
            linear_velocity = 0.1  # m/s
            angular_velocity = 3.4  # rad/s
            # Cream un mesaj de tipul Twist si il publicam
            twist.linear.x = linear_velocity
            twist.angular.z = angular_velocity
            cmd_vel_pub.publish(twist)
            time.sleep(2)
            num_mov += 1
    eat = True
    jmp = 0
# 2. POOP
if eat == True:
    cmd_vel_pub = rospy.Publisher('cmd_vel', Twist, queue_size=10)
    num_mov = 0
    print("sad puppy is going to poop in about 3 seconds\n")
    while True:
        twist = Twist()
        if num_mov == 7:
            twist.linear.x = 0.0
            twist.angular.z = 0.0
            cmd_vel_pub.publish(twist)
            break
        elif num_mov == 6:
            twist.linear.x = -0.8
            twist.angular.z = 0.0
            cmd_vel_pub.publish(twist)
            num_mov = num_mov+1
            time.sleep(0.5)
        elif num_mov == 5:
            twist.linear.x = 0.0
            twist.angular.z = 0.0
            cmd_vel_pub.publish(twist)
            num_mov = num_mov+1
            time.sleep(0.5)
        elif num_mov == 4:
            twist.linear.x = 0.8
            twist.angular.z = 0.0
            cmd_vel_pub.publish(twist)
```

```
            num_mov = num_mov+1
            time.sleep(0.5)
        elif num_mov == 3:
            twist.linear.x = 0.0
            twist.angular.z = 0.0
            cmd_vel_pub.publish(twist)
            num_mov = num_mov+1
            time.sleep(0.5)
        elif num_mov == 2:
            twist.linear.x = -0.8
            twist.angular.z = 0.0
            cmd_vel_pub.publish(twist)
            num_mov = num_mov+1
            time.sleep(0.5)
        elif num_mov == 1:
            twist.linear.x = 0.0
            twist.angular.z = 0.0
            cmd_vel_pub.publish(twist)
            num_mov = num_mov+1
            time.sleep(0.5)
        elif num_mov == 0:
            time.sleep(0.5)
            twist.linear.x = 0.8
            twist.angular.z = 0.0
            cmd_vel_pub.publish(twist)
            num_mov = num_mov+1
            time.sleep(0.5)
    print("puppy has pooped\n")
    eat = False
    jmp = 0
# 3. PLAY
if cv2.countNonZero(mask_blue) > 0:
    # cv2.imshow("detect_blue", mask_blue)
    cmd_vel_pub = rospy.Publisher('cmd_vel', Twist, queue_size=10)
    num_mov = 0
    print("happy puppy running\n")
    # rate=rospy.Rate(10)
    while True:
        twist = Twist()
        if num_mov == 1:
            twist.linear.x = 0.0
            twist.angular.z = 0.0
            cmd_vel_pub.publish(twist)
```

```
            break
        else:
            time.sleep(0.5)
            linear_velocity = 0.5
            angular_velocity = 1.0
            twist.linear.x = linear_velocity
            twist.angular.z = angular_velocity
            cmd_vel_pub.publish(twist)
            time.sleep(6.9)
            # rate.sleep()
            num_mov += 1
    play = True
    jmp = 0
# 4. SLEEP
if play == True:
    print("puppy is really tired\n")
    time.sleep(5)
    print("shhh puppy is sleeping\n")
    jmp = 0
# 5. LOVE
if cv2.countNonZero(mask_orange) > 0:
    # cv2.imshow("detect_orange", mask_orange)
    cmd_vel_pub = rospy.Publisher('cmd_vel', Twist, queue_size=10)
    num_mov = 0
    print("puppy loves you so much\n")
    while True:
        twist = Twist()
        if num_mov == 1:
            twist.linear.x = 0.0
            twist.angular.z = 0.0
            cmd_vel_pub.publish(twist)
            break
        else:
            time.sleep(0.5)
            linear_velocity = 0.5
            angular_velocity = 0.0
            distance = 1.0
            movement_duration = distance / linear_velocity  # seconds
            twist.linear.x = linear_velocity
            twist.angular.z = angular_velocity
            cmd_vel_pub.publish(twist)
            rospy.sleep(movement_duration)
            num_mov += 1
```

```
            jmp = 0
        # 6. ATTACK
        if cv2.countNonZero(mask_red) > 0:
            # cv2.imshow("detect_red", mask_red)
            cmd_vel_pub = rospy.Publisher('cmd_vel', Twist, queue_size=10)
            num_mov = 0
            print("run, puppy is in attack mode\n")
            while True:
                twist = Twist()
                if num_mov == 1:
                    twist.linear.x = 0.0
                    twist.angular.z = 0.0
                    cmd_vel_pub.publish(twist)
                    rospy.sleep(0)
                    break
                else:
                    time.sleep(0.5)
                    linear_velocity = 1.5
                    angular_velocity = 0.0
                    distance = 2.0
                    movement_duration = distance / linear_velocity
                    twist.linear.x = linear_velocity
                    twist.angular.z = angular_velocity
                    cmd_vel_pub.publish(twist)
                    rospy.sleep(movement_duration)
                    num_mov += 1
            jmp = 0
        # FORCE STOP CV
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
except KeyboardInterrupt:
    pass
cap.release()
# ROS MAIN LOOP
rospy.spin()
```

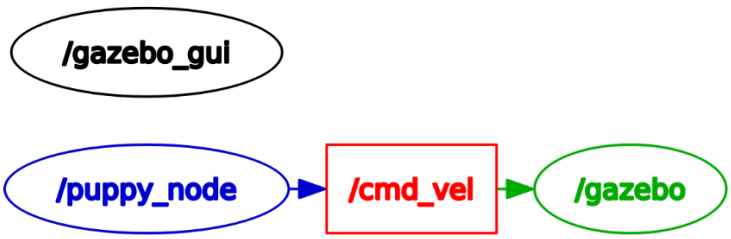

Figura 3 – Structura de noduri

V. Ghidul utilizatorului

Pentru a rula aplicația va trebui ca mai întâi să instalăm/folosim Ubuntu 20.04, ROS Noetic, Pyhton3 și OpenCV2, după care vom urma pașii de mai jos:

- Pas 1: pornim nodul master prin comanda roscore
- Pas 2: rulăm fișierul de tip launch customizat pentru Tamagotchi care:
  - pornește nodul turtlebot3_gazebo ce va genera mediul de simulare
  - pornește rqt_graph în care se observă structura graf-ului de noduri si topic-uri
  - rulează fișierul python
- Pas 4: scriem comanda rostopic echo /cmd_vel pentru a observa în timp real coordonatele robotului din viteza liniară și cea unghiulară
- Pas 5: Ghid de culori pe care le putem arăta robotului:
  - Nimic/Negru – așteaptă să îi oferim atenție
  - Galben – hrănim cățelușul
  - Albastru – se joacă
  - Portocaliu – ne oferă afecțiune și ne iubește
  - Roșu – atacă

Pentru oprirea de urgență a camerei apasăm litera “q”, iar pentru oprirea totală a robotului apasăm “Ctrl-C”.

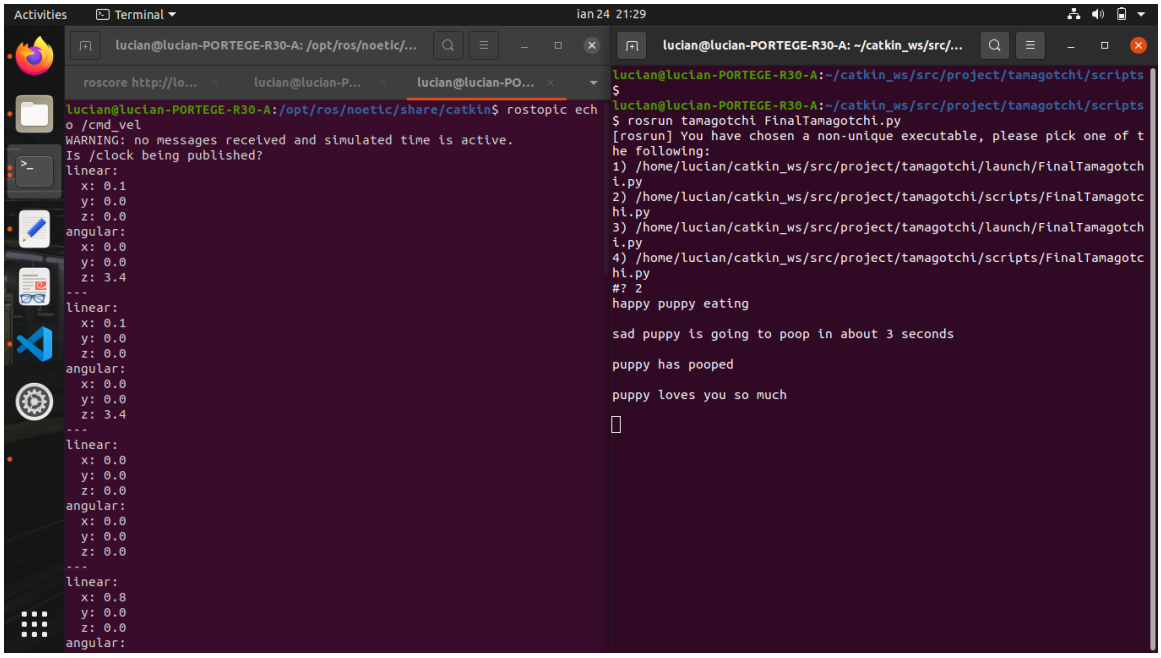

Figura 4 – Testare funcționalități

VI. Testare și punere in funcțiune

Procedurile de testare și punere în funcțiune au fost detaliate mai sus în ghidul programatorului și cel al utilizatorului. Pe scurt, se verifică existența următoarelor tehnologii: TurtleBot3 Burger package, Ubuntu 20.04, ROS Noetic, Python și OpenCV, după care se pornește nodul master (roscore), se lansează fișierul cu extensia launch (roslaunch) și începem să îi aratăm diverse culori robotului în funcție de ce dorim să facă.

VII. Prezentarea firmei

- Lucian-Florin GODEANU
  - ✓ Implementarea funcțiilor default, eat, play
  - ✓ Realizarea documentației
- Alexandra-Raluca BOGDAN
  - ✓ Implementarea funcțiilor attack, poop
  - ✓ Realizarea documentației
- Ligia TRIF
  - ✓ Implementarea funcțiilor love, sleep
  - ✓ Realizarea documentației

VIII. Concluzii

În concluzie, folosind facilitățile TurtleBot-ului am reușit printr-un cod de python și o rețea de noduri să manipulăm un robot mobil care să îndeplinească funcțiile unui animal de companie relativ la culorile pe care le detectează.

IX. Bibliografie

[WWW 01] https://www.techtarget.com/iotagenda/definition/mobile-robot-mobile-robotics

[Hu21] Hu, J.; Bhowmick, P.; Lanzon, A., "Group Coordinated Control of Networked Mobile Robots with Applications to Object Transportation" IEEE Transactions on Vehicular Technology, 2021.

[WWW 02] https://en.wikipedia.org/wiki/Mobile_robot#cite_note-tvt2-1

[Ban98] "Just Another Day's Work". CNN. 1997-07-25. Retrieved 2021-10-19. "Bandai America's Tamagotchi Toys for 1998!". Bandai. Archived from the original on 1998-07-09. Retrieved 2021-10-19.

[WWW 03] https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/

[WWW 04] https://www.ros.org/

[Mu19] Muhammad Luqman, "ROS/Tutorials/UnderstandingNodes – ROS Wiki". ROS.org. Open Robotics. Retrieved 29 April 2019.