COLEGIUL NAȚIONAL “EMIL RACOVIȚĂ”

PROIECT PENTRU OBȚINEREA ATESTĂRII

PROFESIONALE ÎN INFORMATICĂ

**Remote Drive**

Profesor coordonator: Elev:

Lăpuștea Anca Râpeanu George

Cluj-Napoca

Iunie 2021

**Cuprins**

1. Introducere

2. First Tech Challenge

3. Modelul aplicației

4. ZeroTier

5. Structura și conținutul proiectului

6. Funcționalitate

7. Bibliografie

1. Introducere

Pandemia COVID-19 a schimbat multe aspecte ale vieții, și a avut efecte până și în competiții. O astfel de competiție este First Tech Challenge, aceasta fiind un concurs de robotică. Datorită restricțiilor impuse pentru prevenirea răspândirii virusului, echipele de robotică nu s-au mai putut întâlni la fel de ușor pentru a lucra la robot. Aplicația făcută de mine își propune să ofere posibilitatea de a controla robotul de la distanță, și astfel să permită testarea robotului de către “driveri”(cei care, în competiția oficială, ar controla robotul).

2. First Tech Challenge

First Tech Challenge este un concurs destinat elevilor de liceu. Scopul concursului este de a promova robotica în rândul tinerilor, dar în același timp și de a-i învăța importanța unei bune cooperări dintre aceștia. Obiectivul concursului diferă de la an la an, dar, în mare, în fiecare an echipele de robotică trebuie sa construiască un robot care sa joace un obiectiv dat, primele 30 de secunde din meci fiind în autonomă(necontrolat de driveri), iar restul meciului fiind controlat de driveri. Așadar, este foarte important ca driverii să fie obișnuiți cu robotul, și au nevoie de cateva ore de antrenament pentru a obține cât mai multe puncte. Aplicația mea oferă o metodă de a controla robotul de la distanță, driverii putând să exerseze de la ei de acasă.



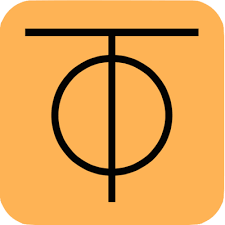
3. Modelul aplicației

La competiția oficială de robotica, driverii își controlează robotul doar prin intermediul unor controlere de tip xbox. Aplicația are 2 părți: partea de ”server”(care va fi folosită de către membrul echipei care are acces fizic la robot) și partea de ”client”(folosită de driveri). Pentru realizarea acestora, am decis să folosesc python3, datorită numeroaselor librării care permit crearea unor astfel de programe, și datorită capacității programelor lui de a fi executate de către orice sistem de operare.

Programul ”client” folosește librăria de python ”pygame” pentru a captura în timp real toate mișcările driverului de pe gamepad, pe care cu ajutorul librăriei ”socket” o trimite serverului prin intermediul unui VPN(Virtual Private Network). Serverul preia comenzile trimise de către client și le trimite mai departe prin WIFI Direct robotului, care cu ajutorul unei clase de Java scrise de mine pune în aplicare comenzile.

4. ZeroTier

Zerotier este un serviciu VPN gratis. Am ales acest VPN deoarece el suportă majoritatea sistemelor de operare.



5. Structura și conținutul proiectului

Proectul a fost realizat treptat, pronind de la cei mai simplii pași, la cei mai complecși.

**Inițializarea**

Primul pas a fost instalarea bibliotecilor necesare, folosite atât de către ”client” cât și de către ”server”. Acesta sunt: socket, time, threading, sys, os și pygame. Apoi, este nevoie și de instalarea serviciului VPN, și de conectarea clientului cu serverul prin acesta.

**Conținutul proiectului**

joystick\_mapping.py

Aceasta sursă python a fost creată cu scopul de a conține detalii legate de configurația controllerului folosit. Așadar, ea conține o serie de constante care identifica în mod unic butoane și axe de pe controler cu ajutorul librăriei pygame. Fișierul poate fi scris manual, dar poate fi și creat de către programul Config.py

A\_BUTTON = 0

B\_BUTTON = 1

X\_BUTTON = 2

Y\_BUTTON = 3

LEFT\_BUMPER\_BUTTON = 4

RIGHT\_BUMPER\_BUTTON = 5

LEFT\_STICK\_BUTTON = 9

RIGHT\_STICK\_BUTTON = 10

START\_BUTTON = 7

BACK\_BUTTON = 6

GUIDE\_BUTTON = 8

INVERT\_X\_AXIS = False

LEFT\_STICK\_X\_AXIS = 0

LEFT\_STICK\_Y\_AXIS = 1

INVERT\_Y\_AXIS = False

RIGHT\_STICK\_X\_AXIS = 3

RIGHT\_STICK\_Y\_AXIS = 4

LEFT\_TRIGGER\_AXIS = 2

LEFT\_TRIGGER\_LOW = -1.0

LEFT\_TRIGGER\_HIGH = 1

RIGHT\_TRIGGER\_AXIS = 5

RIGHT\_TRIGGER\_LOW = -1.0

RIGHT\_TRIGGER\_HIGH = 1

Config.py

Această sursă python a fost creată cu scopul de a scrie fișierul ”joystick\_mapping.py”, astfel încât aplicația să poate suporta mai multe tipuri diferite de controllere. În această sursă, aplicația detectează toate controlerele conectate la computer, și îl determina pe cel corect cu ajutorul utilizatorului, care primește instrucțiuni de folosință pe parcursul rulării programului în consolă.

*#!/bin/python3*

*#importing libraries*

import pygame

import time

import threading

import sys

*#the time an user has to press/release buttons*

TIME\_INTERVAL = 2;

*#get pressed user buttons between the last and current check*

**def** get\_pressed\_buttons(gamepad):

pygame.event.pump();

ans = [];

for i in range(0,gamepad.get\_numbuttons()):

if gamepad.get\_button(i):

ans.append(i);

return ans;

*#gets the used axis between the last and current check*

**def** get\_active\_axes(gamepad,default\_values):

pygame.event.pump();

ans = [];

for i in range(0,gamepad.get\_numaxes()):

*#print(i,gamepad.get\_axis(i),default\_values[i]);*

if abs(gamepad.get\_axis(i) - default\_values[i]) > 5e-2:

ans.append((i,gamepad.get\_axis(i)));

return ans;

**def** main():

*#intializing pygame*

pygame.init();

pygame.display.init()

pygame.joystick.init()

*#getting the number of plugged in joysticks*

joystick\_count = pygame.joystick.get\_count();

if joystick\_count == 0:

sys.exit("ERROR no joystick detected");

*#printing user instructions*

print("detected " + str(joystick\_count))

print("This will cycle through them all with a delay of " + str(TIME\_INTERVAL) + "s.");

print("Hold the A button on your gamepad when its name shows up until the script tells you to release it");

*#checking all gamepads in order to detect the used one*

idx = -1;

gamepad = "";

for i in range(0,joystick\_count):

gamepad = pygame.joystick.Joystick(i);

gamepad.init();

print("Trying joystick " + str(i));

time.sleep(TIME\_INTERVAL);

pygame.event.pump();

if len(get\_pressed\_buttons(gamepad)) > 0:

print("gamepad detected");

idx = i;

break;

gamepad.quit();

*#if none found, throw an error*

if idx == -1:

sys.exit("ERROR no gamepad could be determined");

*#getting the default values of all gamepad inputs*

pygame.event.pump();

default\_values = [];

for i in range(0,gamepad.get\_numaxes()):

default\_values.append(gamepad.get\_axis(i));

time.sleep(TIME\_INTERVAL);

*#opening the config file*

f = open("joystick\_mapping.py","w");

*#promptin user with instructions for configuring all buttons and axis*

print("\n");

print("Press the A button on your gamepad...\n")

while(True):

time.sleep(0.1);

a = get\_pressed\_buttons(gamepad);

if len(a) == 1:

f.write("A\_BUTTON = " + str(a[0]) + "\n");

break;

print("Release the A button on your gamepad...\n");

time.sleep(TIME\_INTERVAL);

print("Press the B button on your gamepad...\n")

while(True):

time.sleep(0.1);

a = get\_pressed\_buttons(gamepad);

if len(a) == 1:

f.write("B\_BUTTON = " + str(a[0]) + "\n");

break;

print("Release the B button on your gamepad...\n");

time.sleep(TIME\_INTERVAL);

print("Press the X button on your gamepad...\n")

while(True):

time.sleep(0.1);

a = get\_pressed\_buttons(gamepad);

if len(a) == 1:

f.write("X\_BUTTON = " + str(a[0]) + "\n");

break;

print("Release the X button on your gamepad...\n");

time.sleep(TIME\_INTERVAL);

print("Press the Y button on your gamepad...\n")

while(True):

time.sleep(0.1);

a = get\_pressed\_buttons(gamepad);

if len(a) == 1:

f.write("Y\_BUTTON = " + str(a[0]) + "\n");

break;

print("Release the Y button on your gamepad...\n");

time.sleep(TIME\_INTERVAL);

f.write("\n");

print("Press the LB button on your gamepad...\n")

while(True):

time.sleep(0.1);

a = get\_pressed\_buttons(gamepad);

if len(a) == 1:

f.write("LEFT\_BUMPER\_BUTTON = " + str(a[0]) + "\n");

break;

print("Release the LB button on your gamepad...\n");

time.sleep(TIME\_INTERVAL);

print("Press the RB button on your gamepad...\n")

while(True):

time.sleep(0.1);

a = get\_pressed\_buttons(gamepad);

if len(a) == 1:

f.write("RIGHT\_BUMPER\_BUTTON = " + str(a[0]) + "\n");

break;

print("Release the RB button on your gamepad...\n");

time.sleep(TIME\_INTERVAL);

print("Press the LS button on your gamepad...\n")

while(True):

time.sleep(0.1);

a = get\_pressed\_buttons(gamepad);

if len(a) == 1:

f.write("LEFT\_STICK\_BUTTON = " + str(a[0]) + "\n");

break;

print("Release the LS button on your gamepad...\n");

time.sleep(TIME\_INTERVAL);

print("Press the RS button on your gamepad...\n")

while(True):

time.sleep(0.1);

a = get\_pressed\_buttons(gamepad);

if len(a) == 1:

f.write("RIGHT\_STICK\_BUTTON = " + str(a[0]) + "\n");

break;

f.write("\n");

print("Release the RS button on your gamepad...\n");

time.sleep(TIME\_INTERVAL);

print("Press the START button on your gamepad...\n")

while(True):

time.sleep(0.1);

a = get\_pressed\_buttons(gamepad);

if len(a) == 1:

f.write("START\_BUTTON = " + str(a[0]) + "\n");

break;

print("Release the START button on your gamepad...\n");

time.sleep(TIME\_INTERVAL);

print("Press the BACK button on your gamepad...\n")

while(True):

time.sleep(0.1);

a = get\_pressed\_buttons(gamepad);

if len(a) == 1:

f.write("BACK\_BUTTON = " + str(a[0]) + "\n");

break;

print("Release the BACK button on your gamepad...\n");

time.sleep(TIME\_INTERVAL);

print("Press the GUIDE button on your gamepad...\n")

while(True):

time.sleep(0.1);

a = get\_pressed\_buttons(gamepad);

if len(a) == 1:

f.write("GUIDE\_BUTTON = " + str(a[0]) + "\n");

break;

f.write("\n");

print("Release the GUIDE button on your gamepad...\n");

time.sleep(TIME\_INTERVAL);

print("Move your left stick to the right...\n");

while(True):

time.sleep(0.1);

a = get\_active\_axes(gamepad,default\_values);

if len(a) == 1:

f.write("INVERT\_X\_AXIS = " + ("True" if a[0][1] < 0 else "False") + "\n");

f.write("LEFT\_STICK\_X\_AXIS = " + str(a[0][0]) + "\n");

break;

print("Release the left stick...\n");

time.sleep(TIME\_INTERVAL);

print("Move your left stick up...\n");

while(True):

time.sleep(0.1);

a = get\_active\_axes(gamepad,default\_values);

if len(a) == 1:

f.write("LEFT\_STICK\_Y\_AXIS = " + str(a[0][0]) + "\n");

f.write("INVERT\_Y\_AXIS = " + ("True" if a[0][1] > 0 else "False") + "\n");

break;

print("Release the left stick...\n");

time.sleep(TIME\_INTERVAL);

print("Move your right stick to the right...\n");

while(True):

time.sleep(0.1);

a = get\_active\_axes(gamepad,default\_values);

if len(a) == 1:

f.write("RIGHT\_STICK\_X\_AXIS = " + str(a[0][0]) + "\n");

break;

print("Release the right stick...\n");

time.sleep(TIME\_INTERVAL);

print("Move your right stick up...\n");

while(True):

time.sleep(0.1);

a = get\_active\_axes(gamepad,default\_values);

if len(a) == 1:

f.write("RIGHT\_STICK\_Y\_AXIS = " + str(a[0][0]) + "\n");

break;

print("Release the right stick...\n");

time.sleep(TIME\_INTERVAL);

print("Press your left trigger...\n");

while(True):

time.sleep(0.1);

a = get\_active\_axes(gamepad,default\_values);

if len(a) == 1:

f.write("LEFT\_TRIGGER\_AXIS = " + str(a[0][0]) + "\n");

f.write("LEFT\_TRIGGER\_LOW = " + str(default\_values[a[0][0]]) + "\n");

f.write("LEFT\_TRIGGER\_HIGH = " + str(1) + "\n");

break;

print("Release the left trigger...\n");

time.sleep(TIME\_INTERVAL);

print("Press your right trigger...\n");

while(True):

time.sleep(0.1);

a = get\_active\_axes(gamepad,default\_values);

if len(a) == 1:

f.write("RIGHT\_TRIGGER\_AXIS = " + str(a[0][0]) + "\n");

f.write("RIGHT\_TRIGGER\_LOW = " + str(default\_values[a[0][0]]) + "\n");

f.write("RIGHT\_TRIGGER\_HIGH = " + str(1) + "\n");

break;

print("Release the right trigger...\n");

time.sleep(TIME\_INTERVAL);

f.close();

main();

Gamepad.py

Această sursă implementează o clasă care conține ”starea” unui controller la un moment dat(sau cu alte cuvinte, ce butoane sunt apăsate și respectiv poziția la care se afla axele controllerului). De asemenea, această clasă implementează și o metoda care codifică starea într-un string, pe care ”Client.py” poate să îl trimită serverului prin ”Server.py”.

*#!/bin/python3*

*#This class serves as interface between Client.py and pygame*

import joystick\_mapping as mapping;

import pygame

import time

import threading

import sys

*#implemented with the help of https://www.pygame.org/docs/ref/joystick.html*

**class** Gamepad:

*#initializing all the fields we need(or more exactly gamepad inputs).*

**def** \_\_init\_\_(self):

self.A = 0;

self.B = 0;

self.X = 0;

self.Y = 0;

self.DPAD\_UP = 0;

self.DPAD\_DOWN = 0;

self.DPAD\_LEFT = 0;

self.DPAD\_RIGHT = 0;

self.LEFT\_BUMPER = 0;

self.RIGHT\_BUMPER = 0;

self.LEFT\_STICK\_PRESSED = 0;

self.RIGHT\_STICK\_PRESSED = 0;

self.LEFT\_TRIGGER = float(0);

self.RIGHT\_TRIGGER = float(0);

self.LEFT\_STICK\_X = float(0);

self.LEFT\_STICK\_Y = float(0);

self.RIGHT\_STICK\_X = float(0);

self.RIGHT\_STICK\_Y = float(0);

self.BACK = 0;

self.START = 0;

self.GUIDE = 0;

*#intiliazing pygame*

pygame.init();

pygame.display.init();

pygame.joystick.init();

joystick\_count = pygame.joystick.get\_count();

*#if no joystick/gamepad is found then we exit the program*

if joystick\_count == 0:

sys.exit("ERROR no joystick detected");

*#This detects the active gamepad of the driver(in case the driver has multiple plugged-in)*

print("detected " + str(joystick\_count));

print("This will cycle through them all with a delay of 3s.");

print("Hold the A button on your gamepad when its name shows up until the script tells you to release it");

idx = -1;

*#Trying every gamepad*

for i in range(0,joystick\_count):

self.gamepad = pygame.joystick.Joystick(i);

self.gamepad.init();

print("Trying joystick " + str(i));

time.sleep(3);

pygame.event.pump();

*#If the A button is pressed then this is the active gamepad*

if self.gamepad.get\_button(mapping.A\_BUTTON) == 1:

print("gamepad initialized");

idx = i;

break;

self.gamepad.quit();

if idx == -1:

sys.exit("ERROR no gamepad could be determined");

*#This asks pygame for the current gamepad inputs.*

**def** update\_inputs(self):

pygame.event.pump();

self.A = self.gamepad.get\_button(mapping.A\_BUTTON);

self.B = self.gamepad.get\_button(mapping.B\_BUTTON);

self.X = self.gamepad.get\_button(mapping.X\_BUTTON);

self.Y = self.gamepad.get\_button(mapping.Y\_BUTTON);

self.DPAD\_UP = 1 if self.gamepad.get\_hat(0)[1] == 1 else 0;

self.DPAD\_DOWN = 1 if self.gamepad.get\_hat(0)[1] == -1 else 0;

self.DPAD\_LEFT = 1 if self.gamepad.get\_hat(0)[0] == -1 else 0;

self.DPAD\_RIGHT = 1 if self.gamepad.get\_hat(0)[0] == 1 else 0;

self.LEFT\_BUMPER = self.gamepad.get\_button(mapping.LEFT\_BUMPER\_BUTTON);

self.RIGHT\_BUMPER = self.gamepad.get\_button(mapping.RIGHT\_BUMPER\_BUTTON);

self.LEFT\_STICK\_PRESSED = self.gamepad.get\_button(mapping.LEFT\_STICK\_BUTTON);

self.RIGHT\_STICK\_PRESSED = self.gamepad.get\_button(mapping.RIGHT\_STICK\_BUTTON);

self.LEFT\_TRIGGER = (self.gamepad.get\_axis(mapping.LEFT\_TRIGGER\_AXIS) - mapping.LEFT\_TRIGGER\_LOW) / (mapping.LEFT\_TRIGGER\_HIGH - mapping.LEFT\_TRIGGER\_LOW);

self.RIGHT\_TRIGGER = (self.gamepad.get\_axis(mapping.RIGHT\_TRIGGER\_AXIS) - mapping.RIGHT\_TRIGGER\_LOW) / (mapping.RIGHT\_TRIGGER\_HIGH - mapping.RIGHT\_TRIGGER\_LOW);

self.LEFT\_STICK\_X = self.gamepad.get\_axis(mapping.LEFT\_STICK\_X\_AXIS) \* (1 if mapping.INVERT\_X\_AXIS == False else -1);

self.LEFT\_STICK\_Y = self.gamepad.get\_axis(mapping.LEFT\_STICK\_Y\_AXIS) \* (1 if mapping.INVERT\_Y\_AXIS == False else -1);

self.RIGHT\_STICK\_X = self.gamepad.get\_axis(mapping.RIGHT\_STICK\_X\_AXIS) \* (1 if mapping.INVERT\_X\_AXIS == False else -1);

self.RIGHT\_STICK\_Y = self.gamepad.get\_axis(mapping.RIGHT\_STICK\_Y\_AXIS) \* (1 if mapping.INVERT\_Y\_AXIS == False else -1);

self.BACK = self.gamepad.get\_button(mapping.BACK\_BUTTON);

self.START = self.gamepad.get\_button(mapping.START\_BUTTON);

self.GUIDE = self.gamepad.get\_button(mapping.GUIDE\_BUTTON);

*#This transforms the state of the input into a string that can be sent to the server by the client.*

**def** get\_transmission\_message(self):

message\_components = [];

message\_components.append("A~" + str(self.A));

message\_components.append("B~" + str(self.B));

message\_components.append("X~" + str(self.X));

message\_components.append("Y~" + str(self.Y));

message\_components.append("D\_UP~" + str(self.DPAD\_UP));

message\_components.append("D\_DN~" + str(self.DPAD\_DOWN));

message\_components.append("D\_LT~" + str(self.DPAD\_LEFT));

message\_components.append("D\_RT~" + str(self.DPAD\_RIGHT));

message\_components.append("L\_BMP~" + str(self.LEFT\_BUMPER));

message\_components.append("R\_BMP~" + str(self.RIGHT\_BUMPER));

message\_components.append("L\_PRS~" + str(self.LEFT\_STICK\_PRESSED));

message\_components.append("R\_PRS~" + str(self.RIGHT\_STICK\_PRESSED));

message\_components.append("L\_TRG~" + str(self.LEFT\_TRIGGER));

message\_components.append("R\_TRG~" + str(self.RIGHT\_TRIGGER));

message\_components.append("L\_X~" + str(self.LEFT\_STICK\_X));

message\_components.append("L\_Y~" + str(self.LEFT\_STICK\_Y));

message\_components.append("R\_X~" + str(self.RIGHT\_STICK\_X));

message\_components.append("R\_Y~" + str(self.RIGHT\_STICK\_Y));

message\_components.append("BACK~" + str(self.BACK));

message\_components.append("START~" + str(self.START));

message\_components.append("GUIDE~" + str(self.GUIDE));

msg = message\_components[0];

for i in range(1,len(message\_components)):

msg = msg + "," + message\_components[i];

return msg;

environmental\_variables.py

Această sursă conține mai multe variabile constante, folosite atât de server, cât și de către client(de exemplu: ip-urile serverului și ale driverilor, formatul codificării stringului, lungimea maximă a unui string trimis prin ”socket”).

*#This file contains all variables that should be shared by the server or the client and/or need to be locally changed(thus this file is included in .gitignore)*

import socket;

TIMEOUT\_DURATION = 2;*#The amount of seconds that the server waits to receive a new packet. If it doesnt receive one, it will assume that the driver disconnected*

DRIVER\_ADDRESSES = ["127.0.0.1"];*#A list of all driver's ip addresses. This is a security feature used in order to prevent executing commands that are not comming from our drivers.*

DRIVER\_COUNT = 2;

ROBOT\_ADDRESS = ""*#robot WIFI-Direct ip addres, has to be the same with the one in RemoteDrive.java*

ROBOT\_PORT = 6969*#port used to communicate with robot, has to be the same with the one in RemoteDrive.java*

ROBOT = (ROBOT\_ADDRESS,ROBOT\_PORT);

FORMAT = "utf-8";*#Message encoding, has to be the same with the one int RemoteDrive.java*

HOST\_PORT = 42069*#host port*

*#Host ip address*

*#HOST\_ADDRESS = socket.gethostbyname(socket.gethostname()) #This should be used when running in LAN, mainly for testing*

HOST\_ADDRESS = "172.22.216.79"; *#This should be the host's public address*

HOST = (HOST\_ADDRESS,HOST\_PORT);

DISCONNECT = "!DISCONNECT";

MAX\_MESSAGE\_SIZE = 1024;

ERROR\_MESSAGES = ["Too many drivers, connection failed", "Address not recognized, connection failed",""];

Client.py

Această sursă inițializează conexiunea cu serverul (sau afișează o eroare în cazul în care aceasta nu este posibilă), și trimite serverului starea curentă a controlerului unui driver la fiecare schimbare a acesteia, sau la trecerea unei secunde de la ultima stare transmisă (pentru a evita pierderea de informație).

*#!/bin/python3*

import environmental\_variables as \_\_env;

import socket;

import time;

import threading;

import Gamepad;

import sys;

*#This runs on a separate thread and insures that all server messages will be printed to the driver*

*#Server messages dont actually affect the client behavior, so they can just be printed while the actual client code is running*

**def** handle\_server\_messages(driver):

while True:

msg = driver.recv(\_\_env.MAX\_MESSAGE\_SIZE).decode(\_\_env.FORMAT);

print(msg);

if msg == \_\_env.DISCONNECT:

return;

**def** start():

*#setting up our gamepad interface*

gamepad = Gamepad.Gamepad();

*#establishing a TCP connection to the server*

driver = socket.socket(socket.AF\_INET,socket.SOCK\_STREAM);

driver.connect(\_\_env.HOST);

*#Checking if the server threw an error and disconects if thats the case*

connect\_msg = driver.recv(\_\_env.MAX\_MESSAGE\_SIZE).decode(\_\_env.FORMAT);

print(connect\_msg);

if connect\_msg in \_\_env.ERROR\_MESSAGES:

return ;

*#starting the server messages thread.*

thread = threading.Thread(target=handle\_server\_messages,args = [driver]);

thread.start();

print("driver connected at ",\_\_env.HOST);

*#To prevent spamming the server and causing undefined behavior, we are only sending the input message when the actual input state changes,*

*#with a delay of 10ms between messages.*

*#However, since we are using UDP to communicate with the robot, there is a slight chance of packet loss*

*#To prevent that, the client will resend the message if a second has passed since it last sent one*

last\_transmission\_message = "";

last\_time = time.time() - 20;

while True:

time.sleep(0.01);

gamepad.update\_inputs();

message = gamepad.get\_transmission\_message() + "|";*#'|' separator between packets*

if message != last\_transmission\_message or (time.time() - last\_time >= 1):

last\_transmission\_message = message;

last\_time = time.time();

driver.send(message.encode(\_\_env.FORMAT));

start();

Server.py

Această sursă este responsabilă pentru a prelua mesajele transmise de către driver (după verificarea identității acestora și atribuirea unui loc) și a le transmite mai departe robotului.

*#!/bin/python3*

import environmental\_variables as \_\_env;

import socket;

import threading;

import os;

import sys;

import time;

*#Setting up the host socket*

host = socket.socket(socket.AF\_INET,socket.SOCK\_STREAM);

host.bind(("0.0.0.0",\_\_env.HOST\_PORT));

*#Setting up the robot socket*

robot = socket.socket(socket.AF\_INET,socket.SOCK\_DGRAM);

*#Keeps track of available driver spots. When a driver connects it will take the first available spot. If the first driver disconnects, the second will take its place.*

available\_driver\_spots = [False,True,True];

*#True if server is running*

SERVER\_RUNNING = False;

*#This handles a client connection*

**def** handle\_client(conn,addr):

**global** SERVER\_RUNNING;

**global** available\_driver\_spots;

print("attempted connection ",addr);

if addr[0] not in \_\_env.DRIVER\_ADDRESSES:*#In case that the driver is not recognized we end the connection.*

print("address not recognized, connection failed");

conn.send(\_\_env.ERROR\_MESSAGES[1].encode(\_\_env.FORMAT));

conn.close();

return ;

else:

*#Finding the first available driver spot*

driver\_id = -1;

for i in range(0,len(available\_driver\_spots)):

if available\_driver\_spots[i] == True:

driver\_id = i;

available\_driver\_spots[i] = False;

break;

*#If no spot was found it means that we can not accept this connection because we would have too many drivers.*

if driver\_id == -1:

conn.send(\_\_env.ERROR\_MESSAGES[0].encode(\_\_env.FORMAT));

conn.close();

return ;

*#Sending a welcome message to the driver and setting the timeout duration*

print("driver " + str(driver\_id) + " connected at " + str(addr));

conn.send(str("welcome driver #" + str(driver\_id)).encode(\_\_env.FORMAT));

conn.settimeout(\_\_env.TIMEOUT\_DURATION);

connected = True;

buff = "";

while connected and SERVER\_RUNNING:

try:

*#Await driver commands*

msg = conn.recv(\_\_env.MAX\_MESSAGE\_SIZE).decode(\_\_env.FORMAT);

buff += msg;

except socket.timeout:

*#If driver timed out we assume they disconnected, so we end the connection,free his spot(this allows the driver to reconnect and the second driver to take over) and output a debug message*

connected = False;

print("Driver " + str(driver\_id) + " disconnected");

available\_driver\_spots[driver\_id] = True;

if connected == False:

break;

*#Checking if a previous driver disconnected so this driver can take over*

promoted = False;

for i in range(0,driver\_id):

if available\_driver\_spots[i] == True:

avialable\_driver\_spots[i] = False;

avialable\_driver\_spots[driver\_id] = True;

driver\_id = i;

promoted = True;

break;

*#if this driver was promoted we let them know by sending a message*

if promoted:

conn.send(("Promoted to driver #" + str(driver\_id)).encode(\_\_env.FORMAT));

pieces = buff.split("|");

buff = pieces.pop();

*#We add the final piece to our message, that being the gamepad number, and send it to the robot*

for msg in pieces:

if len(msg) > 0:

msg = "G~" + str(driver\_id) + "," + msg;

print("recieved following command ");

print(msg);

robot.sendto(msg.encode(\_\_env.FORMAT),\_\_env.ROBOT);

if SERVER\_RUNNING == False and connected == True:*#if the connection is still active but the server is shutting down let the client know to disconnect*

conn.send(\_\_env.DISCONNECT.encode(\_\_env.FORMAT));

conn.close();

**def** start():

*#Starting the server*

**global** SERVER\_RUNNING;

SERVER\_RUNNING = True;

host.listen();

print("listening");

while True:

*#Waiting for a new connection*

conn,addr = host.accept();

*#When a connection established, we handle it on a separate thread in order to be able to continue accepting connections.*

thread = threading.Thread(target=handle\_client,args = (conn,addr));

thread.start();

*#Starting the server*

try:

start()

except KeyboardInterrupt:

*#Cleanly shutting down the server.*

print('Interrupted')

try:

SERVER\_RUNNING = False;

time.sleep(\_\_env.TIMEOUT\_DURATION + 1);

host.shutdown(socket.SHUT\_RDWR)

host.close()

sys.exit(0)

except SystemExit:

os.\_exit(0)

CustomGamepad.java

Echivalentul în java a ”Gamepad.py”, scopul acestei surse este de a implementa o clasă care să conțină starea controlerului unui driver.

package **org.firstinspires.ftc.teamcode**;

*//Implements a custom gamepad class that uses the same variables as the FTC one, but doesnt implement the other methods*

**public** **class** CustomGamepad {

**boolean** a;

**boolean** b;

**boolean** x;

**boolean** y;

**boolean** dpad\_up;

**boolean** dpad\_down;

**boolean** dpad\_left;

**boolean** dpad\_right;

**boolean** left\_stick;

**boolean** right\_stick;

**boolean** left\_bumper;

**boolean** right\_bumper;

**double** left\_trigger;

**double** right\_trigger;

**double** left\_stick\_x;

**double** left\_stick\_y;

**double** right\_stick\_x;

**double** right\_stick\_y;

**boolean** back;

**boolean** start;

**boolean** guide;

*//Setting all fields to a neutral state*

CustomGamepad() {

a = false;

b = false;

x = false;

y = false;

dpad\_up = false;

dpad\_down = false;

dpad\_left = false;

dpad\_right = false;

left\_stick = false;

right\_stick = false;

left\_bumper = false;

right\_bumper = false;

left\_trigger = 0;

right\_trigger = 0;

left\_stick\_x = 0;

left\_stick\_y = 0;

right\_stick\_x = 0;

right\_stick\_y = 0;

back = false;

start = false;

guide = false;

}

}

RemoteDrive.java

Această sursă de java implementează o clasă care folosește librăriile FTC necesare oricum pentru programarea robotului, și care folosind ”CustomGamepad.java” preia și oferă programatorilor robotului un mod ușor de a folosi informațiile oferite de către server.

*/\* Copyright (c) 2017 FIRST. All rights reserved.*

*\**

*\* Redistribution and use in source and binary forms, with or without modification,*

*\* are permitted (subject to the limitations in the disclaimer below) provided that*

*\* the following conditions are met:*

*\**

*\* Redistributions of source code must retain the above copyright notice, this list*

*\* of conditions and the following disclaimer.*

*\**

*\* Redistributions in binary form must reproduce the above copyright notice, this*

*\* list of conditions and the following disclaimer in the documentation and/or*

*\* other materials provided with the distribution.*

*\**

*\* Neither the name of FIRST nor the names of its contributors may be used to endorse or*

*\* promote products derived from this software without specific prior written permission.*

*\**

*\* NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED BY THIS*

*\* LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS*

*\* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,*

*\* THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE*

*\* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE*

*\* FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL*

*\* DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR*

*\* SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER*

*\* CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,*

*\* OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE*

*\* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.*

*\*/*

package **org.firstinspires.ftc.teamcode**;

import **android.util.Log**;

import **com.qualcomm.robotcore.eventloop.opmode.LinearOpMode**;

import **com.qualcomm.robotcore.eventloop.opmode.TeleOp**;

import **com.qualcomm.robotcore.hardware.DcMotor**;

import **com.qualcomm.robotcore.util.ElapsedTime**;

import **com.qualcomm.robotcore.util.Range**;

import **java.net.DatagramPacket**;

import **java.net.DatagramSocket**;

import **java.nio.charset.StandardCharsets**;

*/\*\**

*\* This file contains an minimal example of a Linear "OpMode". An OpMode is a 'program' that runs in either*

*\* the autonomous or the teleop period of an FTC match. The names of OpModes appear on the menu*

*\* of the FTC Driver Station. When an selection is made from the menu, the corresponding OpMode*

*\* class is instantiated on the Robot Controller and executed.*

*\**

*\* This particular OpMode just executes a basic Tank Drive Teleop for a two wheeled robot*

*\* It includes all the skeletal structure that all linear OpModes contain.*

*\**

*\* Use Android Studios to Copy this Class, and Paste it into your team's code folder with a new name.*

*\* Remove or comment out the @Disabled line to add this opmode to the Driver Station OpMode list*

*\*/*

*//Implements a LinearOpMode that gets its gamepad inputs from the wifi-direct packets*

@**TeleOp**(name="RemoteDriveTest", group="Linear Opmode")

@**Disabled**

**public** **class** RemoteDrive **extends** LinearOpMode {

*//Wifi-direct socket*

**private** **DatagramSocket** socket;

**private** **boolean** canRunGamepadThread;

*//the thread which awaits WIFI-DIRECT packets and parses them and updates the gamepad's fields*

**private** **Thread** gamepadHandler;

*//The two custom gamepads, one for each driver*

**CustomGamepad** gamepad1,gamepad2;

**private** **void** startGamepadHandlerThread() {

*// Creating the gamepadHandlerThread*

gamepadHandler = new Thread(new Runnable() {

@**Override**

**public** **void** run() {

while (canRunGamepadThread) {

**String** gamepadAction = "";

*// Awaiting WIFI-DIRECT packet*

Log.d("THREAD","ok, awaiting message");

try {

**byte**[] buffer = new **byte**[1024];

**DatagramPacket** response = new DatagramPacket(buffer, buffer.length);

socket.receive(response);

gamepadAction = new String(buffer,0,response.getLength(), StandardCharsets.UTF\_8);

Log.d("THREAD","received " + gamepadAction);

telemetry.addData("received ",gamepadAction);

telemetry.update();

} catch (**Exception** e) {

Log.d("EXCEPTION",e.getMessage());

}

if(gamepadAction.isEmpty() == false){

**int** gamepad\_id = -1;

**CustomGamepad** curr = new CustomGamepad();

**String**[] parts = gamepadAction.split(",");

Log.d("RECEIVED ", gamepadAction);

*//Parsing the received action*

for(**int** i = 0;i < parts.length;i++){

**String**[] command = parts[i].split("~");

Log.d("COMMAND\_HEAD: ",command[0]);

Log.d("COMMAND\_STUFF: ",command[1]);

if(command[0].equals("G")){

gamepad\_id = Integer.parseInt(command[1]);

}else if(command[0].equals("A")){

curr.a = (Integer.parseInt(command[1]) == 1);

}else if(command[0].equals("B")){

curr.b = (Integer.parseInt(command[1]) == 1);

}else if(command[0].equals("X")){

curr.x = (Integer.parseInt(command[1]) == 1);

}else if(command[0].equals("Y")){

curr.y = (Integer.parseInt(command[1]) == 1);

}else if(command[0].equals("D\_UP")){

curr.dpad\_up = (Integer.parseInt(command[1]) == 1);

}else if(command[0].equals("D\_DN")){

curr.dpad\_down = (Integer.parseInt(command[1]) == 1);

}else if(command[0].equals("D\_LT")){

curr.dpad\_left = (Integer.parseInt(command[1]) == 1);

}else if(command[0].equals("D\_RT")){

curr.dpad\_right = (Integer.parseInt(command[1]) == 1);

}else if(command[0].equals("L\_BMP")){

curr.left\_bumper = (Integer.parseInt(command[1]) == 1);

}else if(command[0].equals("R\_BMP")){

curr.right\_bumper = (Integer.parseInt(command[1]) == 1);

}else if(command[0].equals("L\_PRS")){

curr.left\_stick = (Integer.parseInt(command[1]) == 1);

}else if(command[0].equals("R\_PRS")){

curr.right\_stick = (Integer.parseInt(command[1]) == 1);

}else if(command[0].equals("L\_TRG")){

curr.left\_trigger = Double.parseDouble(command[1]);

}else if(command[0].equals("R\_TRG")){

curr.right\_trigger = Double.parseDouble(command[1]);

}else if(command[0].equals("L\_X")){

curr.left\_stick\_x = Double.parseDouble(command[1]);

}else if(command[0].equals("L\_Y")){

curr.left\_stick\_y = Double.parseDouble(command[1]);

}else if(command[0].equals("R\_X")){

curr.right\_stick\_x = Double.parseDouble(command[1]);

}else if(command[0].equals("R\_Y")){

curr.right\_stick\_y = Double.parseDouble(command[1]);

}else if(command[0].equals("BACK")){

curr.back = (Integer.parseInt(command[1]) == 1);

}else if(command[0].equals("START")){

curr.start = (Integer.parseInt(command[1]) == 1);

}else if(command[0].equals("GUIDE")){

curr.guide = (Integer.parseInt(command[1]) == 1);

}

}

*//updating the required gamepad*

if(gamepad\_id == 1){

gamepad1 = curr;

}else if(gamepad\_id == 2){

gamepad2 = curr;

}

}

}

gamepadHandler.interrupt();

}

});

gamepadHandler.setName("Gamepad Handler Thread");

gamepadHandler.setPriority(Thread.NORM\_PRIORITY);

gamepadHandler.start();

}

*// Initializes the WIFI-DIRECT socket*

**public** **void** \_init(){

*//Initializing gamepads*

this.gamepad1 = new CustomGamepad();

this.gamepad2 = new CustomGamepad();

**String** address = "192.168.49.1"; *//Check "Program and Manage" tab on the Driver Station and verify the IP address*

**int** port = 6969; *//Change as needed*

canRunGamepadThread = false;

*//Trying to start listening to the specified port*

try {

this.socket = new DatagramSocket(port);

} catch (**Exception** ex) {

Log.d("EXCEPTION",ex.getMessage());

}

Log.d("INIT","Initialized");

Log.d("INIT","Connect your server to " + address + ":" + port);

telemetry.update();

}

*// Starts the gamepadHandleThread*

**public** **void** after\_start(){

Log.d("START","this just started");

canRunGamepadThread = true;

startGamepadHandlerThread();

}

*// Cleanly closes the gamepadHandleThread and the socket*

**public** **void** \_end(){

Log.d("END","teleop ended");

canRunGamepadThread = false;

socket.close();

}

*// runOpMode implementation example*

@**Override**

**public** **void** runOpMode(){

this.\_init();

waitForStart();

this.after\_start();

if (opModeIsActive()) {

while (opModeIsActive()) {

;

}

}

this.\_end();

}

}

6. Funcționalitate

Utilitatea acestui proiect intervine în cazul în care întâlnirea membrilor unei echipe este îngreunată de diverși factori externi. Pe viitor, aplicația ar putea fi îmbunătățită prin adăugarea posibilității de a controla robotul prin tastatură, sau printr-un feedback video oferit tot de aceasta (el fiind necesar pentru driveri, dar care momentan poate fi obținut doar prin utilizarea altor aplicații precum Zoom sau Discord).

7. Bibliografie

* Documentația FTC:<https://ftctechnh.github.io/ftc_app/doc/javadoc/index.html>
* Documentația socket: <https://docs.python.org/3/library/socket.html>
* Documentația pygame: https://www.pygame.org/docs/