KLAIPĖDOS UNIVERSITETAS

Jūros technologijų ir gamtos mokslų fakultetas

Informatikos ir statistikos katedra

**AUTONOMINĖS dviratės TRANSPORTO PRIEMONĖS INFORMACINĖS SISTEMoS TYRIMAS IR PLĖTROS GALIMYBĖS**

Studijų dalyko „Dirbtinis intelektas informacinėse sistemose“ projektinio darbo aprašymas

Darbą atliko JMNTS-22 gr. stud. Lukas Jankauskas

JMNTS-22 gr. stud. Valdas Piekus

Darbą tikrino doc. dr. Darius Drungilas

Klaipėda, 2023

**TURINYS**

[1. PROJEKTO VIZIJA 4](#_Toc135840505)

[2. ROBOTŲ OPERACINĖS SISTEMOS GALIMYBĖS 6](#_Toc135840506)

[2.1. Sistemos paruošimas 7](#_Toc135840507)

[2.1.1. Operacinės sistemos įrašymas 7](#_Toc135840508)

[2.1.2. ROS įrašymas 12](#_Toc135840509)

[2.2. RPLidar ir Hector\_SLAM integracija 13](#_Toc135840510)

[2.3. Paketų kūrimas 17](#_Toc135840511)

[3. JUDĖJIMO VALDYMAS 19](#_Toc135840512)

[4. JUTIKLIŲ VALDYMAS 26](#_Toc135840513)

[4.1. Informacija apie IR jutiklius 26](#_Toc135840514)

[4.2. Informacija apie ultragarsinius jutikliu 27](#_Toc135840515)

[4.3. Mikrovaldiklio pasirinkimas 27](#_Toc135840516)

[4.4. Jutiklių sujungimas į Raspberry Pi Pico 28](#_Toc135840517)

[4.5. Duomenų apdorojimas Raspberry Pi Pico valdiklyje 28](#_Toc135840518)

[4.6. Duomenų priėmimas Raspberry Pi 4B valdiklyje 31](#_Toc135840519)

[5. TOLIMESNĖ PLĖTRA 34](#_Toc135840520)

**LENTELIŲ SĄRAŠAS**

[1 lentelė. HBridge ir 4B sujungimas su kintamųjų pavadinimais. 19](#_Toc135843302)

[2 lentelė. HBridge jungčių elgesys pagal sroves 19](#_Toc135843303)

[3 lentelė. Valdiklio komandos roboto ratams valdyti 26](#_Toc135843304)

[4 lentelė. Jutiklių sujungimas Pico mikrovaldiklyje 28](#_Toc135843305)

**PAVEIKSLĖLIŲ SĄRAŠAS**

[1 pav. Transporto priemonės bazė 4](#_Toc135843445)

[2 pav. Drono sistemos diagrama 6](#_Toc135843446)

[3 pav. SD kortelės pasirinkimas Raspberry Pi Imager programoje 7](#_Toc135843447)

[4 pav. Operacinės sistemos pasirinkimas Raspberry Pi Imager programoje 8](#_Toc135843448)

[5 pav. Prisijungimas prie operacinės sistemos 8](#_Toc135843449)

[6 pav. Prisijungimas prie Raspberry Pi 4B pasinaudojant Putty programa 10](#_Toc135843450)

[7 pav. X11 programų paleidimo nustatymai Putty programoje 11](#_Toc135843451)

[8 pav. X11 programos pavyzdys 12](#_Toc135843452)

[9 pav. TurtleSim GUI pavyzdys 13](#_Toc135843453)

[10 pav. Naudojami Putty terminalai 16](#_Toc135843454)

[11 pav. Hector\_SLAM sukurtas aplinkos žemėlapis 17](#_Toc135843455)

[12 pav. Pavyzdinės Catkin modulio programos kodas bei failų struktūra 18](#_Toc135843456)

[13 pav. Raspberry Pi 4B jungimo prie H Bridge pavyzdys 20](#_Toc135843457)

[14 pav. Infraraudonųjų spindulių jutiklis 26](#_Toc135843458)

[15 pav. Jutiklio atstumo ir voltažo ryšio kreivė 27](#_Toc135843459)

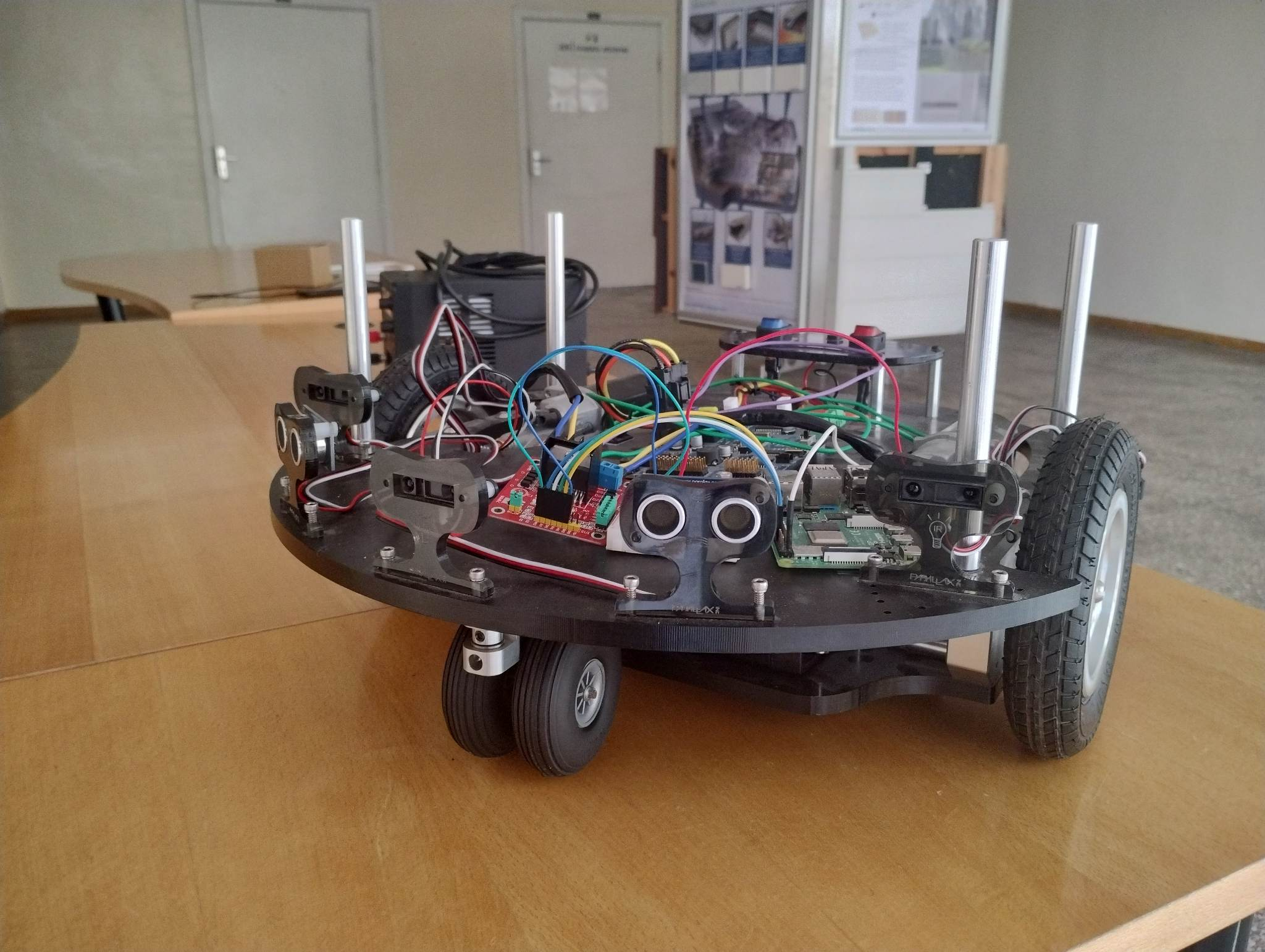
[16 pav. Ultragarsinis jutiklis 27](#_Toc135843460)

1. PROJEKTO VIZIJA

Suplanuoti ir sukurti autonominę dviratę transporto priemonę (toliau - *dronas*), kuri galėtų aptikti kliūtis esančias aplinkoje bei pagal nurodytą maršrutą arba judėjimo režimą atliktų važiavimo manevrus.

Būtų naudojama iš anksto pagaminta transporto priemonės bazė kuri susideda iš:

* medinės plokštumos;
* dviejų „Parralax #28962 DC w/ Quadrature Encoders“ variklių su kontroleriais;
* dviejų varomųjų ratų;
* dviejų stabilumą palaikančių ratų;
* dviejų apsukas stebinčių jutiklių;
* dviejų „Parralax SLA 752-00007“ (12V, 7.2Ah) akumuliatorių;
* vienos „Parralax Eddye Control Board Rev. B“ pagrindinės plokštės;
* trijų„Sharp GP2Y0A21YK0F“ infraraudonųjų spindulių (toliau - *IR*) jutiklių;
* dviejų „PING)))“ ultragarsinių jutiklių.



1. Transporto priemonės bazė

Taip pat naudojama papildoma įranga (aprašyta šiame dokumente):

* Raspberry Pi 4B (toliau - *4B*) mikrovaldiklis;
* Raspberry Pi Pico (toliau - *Pico*) mikrovaldiklis;
* RPLidar A1M8 (toliau - *Lidar*);
* HBridge Duino L298N

1. ROBOTŲ OPERACINĖS SISTEMOS GALIMYBĖS

Robotų Operacinė Sistema (toliau - *ROS*) yra robotikos srityje plačiai naudojamas programinės įrangos bibliotekų ir įrankių rinkinys. Drone esantys komponentai gali būti atskirai valdomi ROS Catkin sukompiliuotais moduliais (angl. *Catkin Package*, toliau - *moduliai*), kurie tarpusavyje gali bendrauti žinutėmis (angl. *topics*) – dėl šios priežasties sistema yra *paskirstytos architektūros*. Šio projekto kontekste galima apibrėžti kaip: komunikacija tarp skirtingus komponentus valdančių C++ programų.

Kaip ir internetinėse svetainėse, paremtose moderniomis technologijomis, į ROS sistemas galima integruoti jau sukurtus modulius, kurie atliktų tam tikrus procesus – ar tai būtų duomenų atvaizdavimo, ar lazerinio įrenginio valdymas.

Dėl šitų funkcionalumų, drono informacinę sistemą yra galima apibrėžti diagrama, kurioje pagrindinis įrenginių/duomenų valdymas yra atliekamas ROS posistemėje.

A diagram of a flowchart

Description automatically generated with low confidence

1. Drono sistemos diagrama

Diagramoje galima pastebėti, jog ROS posistemei įgalinti bus naudojamas Raspberry Pi 4B – taip buvo nuspręsta, kadangi tai buvo jau turimas, pakankamai galingas mikrovaldiklis šiai sistemai palaikyti.

* 1. Sistemos paruošimas

Šiame skyriuje yra aprašomas operacinės sistemos bei ROS įrašymas į 4B.

Oficialias instrukcijas galima rasti čia: <http://wiki.ros.org/noetic/Installation/Ubuntu>

* + 1. Operacinės sistemos įrašymas

Turi būti naudojama naujausia ROS versija – Noetic Ninjemys, tam kad būtų turimas geriausias palaikymas naudojamiems ir potencialiems naudojimo/sukūrimo moduliams.

Svarbu pabrėžti, jog reikia naudoti Ubuntu 20.04 versiją, kadangi ROS Noetic Ninjemys yra kurtas būtent šiai operacinei sistemai (šaltinis: <http://wiki.ros.org/noetic>).

Atliktas įsirašymo procesas (instrukcijos yra papildytas oficialių instrukcijų variantas susidūrus su problemomis ir jas išsprendus):

1. Parsisiųsti Raspberry Pi Imager programą: <https://www.raspberrypi.com/software/>.

2. Įdėti 4B SD kortelę į kompiuterį.

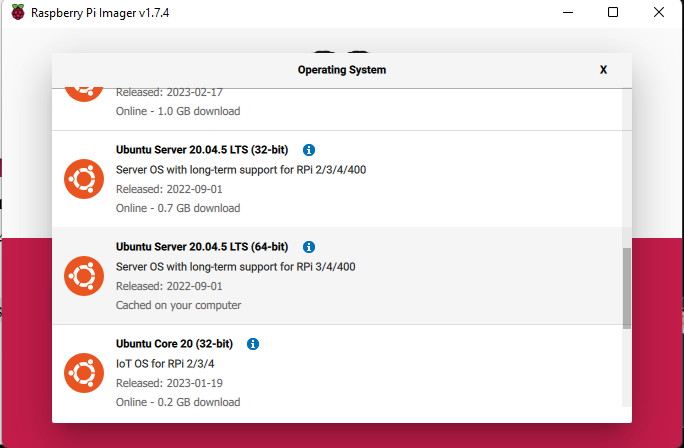
3. Atsidarius programą pasirinkti SD kortelę:

A screenshot of a computer

Description automatically generated with medium confidence

1. SD kortelės pasirinkimas Raspberry Pi Imager programoje

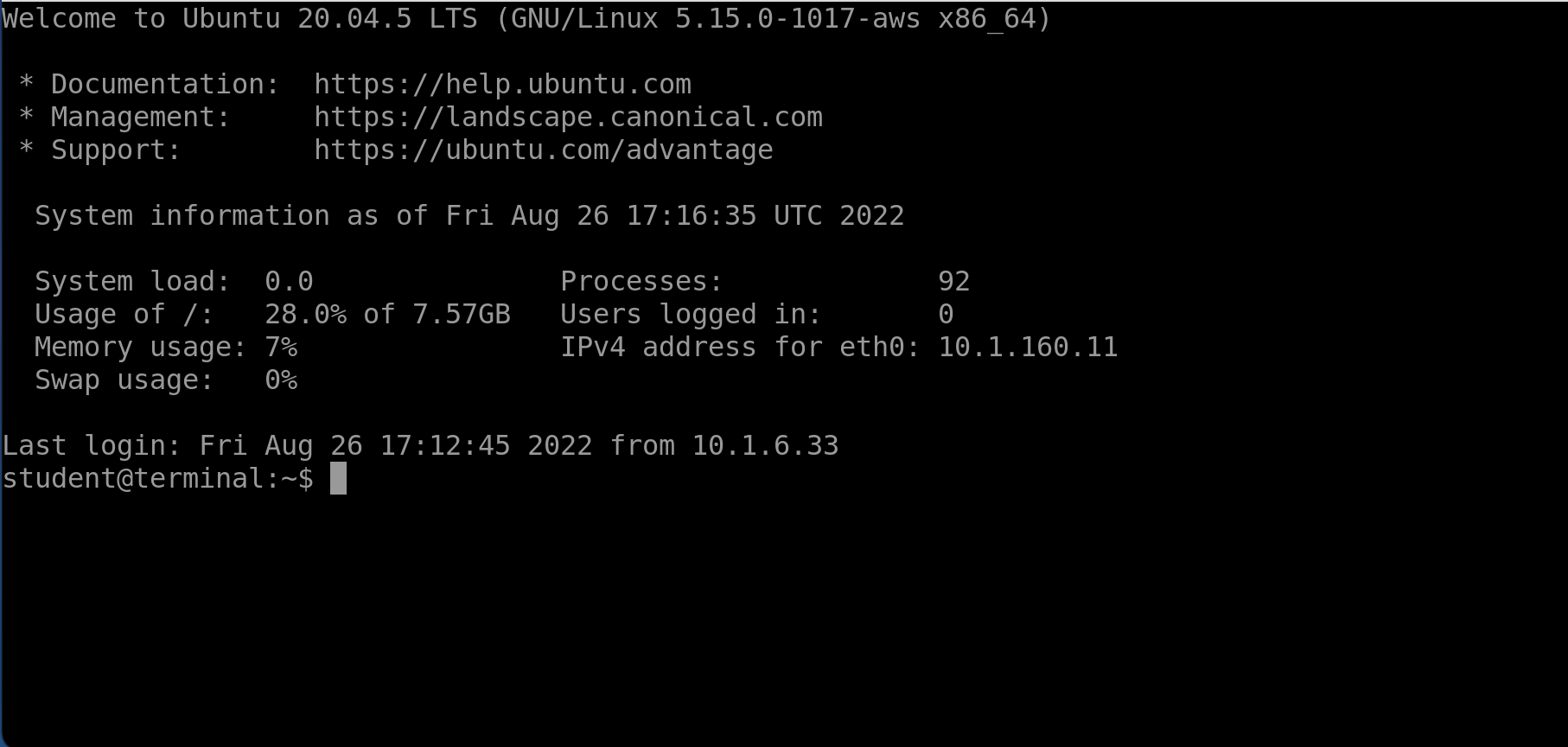
4. Pasirinkti Ubuntu 20.04 x64 operacinę sistemą:



1. Operacinės sistemos pasirinkimas Raspberry Pi Imager programoje

**Pastebėjimas**: šio dokumento GitHub repozitorijoje yra pateiktas operacinės sistemos atvaizdas, kurį įsirašius būtų įrašyta viskas kas yra atlikta šiose instrukcijose.

5. Įsidėjus kortelę ir pasileidus 4B, prireiks kelių momentų kol sistemos naudotojas su administracinėmis privilegijomis bus sukurtas. Tada prisijungimas yra: ubuntu ir slaptažodis: ubuntu (GitHub atvaizde prisijungimas: ubuntu; slaptažodis: raspberrypi).



1. Prisijungimas prie operacinės sistemos

6. Reikia atrasti Wifi modulio pavadinimą:

sudo lshw

7. Nueiti į interneto ryšio nustatymų aplanką:

cd /etc/netplan/

8. Atsidaryti interneto ryšio nustatymų failą:

sudo nano 50-cloud-init.yaml

9. Prisidėti Wifi modulį ir prisijungimo duomenis:

# This file is generated from information provided by

# the datasource. Changes to it will not persist across an instance.

# To disable cloud-init's network configuration capabilities, write a file

# /etc/cloud/cloud.cfg.d/99-disable-network-config.cfg with the following:

# network: {config: disabled}

network:

version: 2

ethernets:

eth0:

optional: true

dhcp4: true

# add wifi setup information here ...

wifis:

wlan0:

optional: true

access-points:

"YOUR-SSID-NAME":

password: "YOUR-NETWORK-PASSWORD"

dhcp4: true

10. Su šia komanda galima patikrinti ar teisingai užpildyta:

sudo netplan –debug try

11. Sugeneruoti interneto ryšio konfigūraciją:

sudo netplan --debug generate

12. Sugeneruotą konfigūraciją įgalinti:

sudo netplan --debug apply

13. Pritaikius konfigūraciją reikia perkrauti 4B:

sudo reboot

14. Pavykus 4B persikrauti ir prisijungti prie interneto, reikia atnaujinti operacinės sistemos modulius (toliau visos terminalo komandos kurios yra pateikiamos kelios turi būti paleidžiamos po vieną ir eilės tvarka):

sudo apt-get update

sudo apt-get upgrade

15. Reikia įgalinti SSH, kad butų galima prisijungti prie 4B terminalo naudojantis Putty:

sudo ssh-keygen -A

sudo systemctl enable ssh

sudo systemctl start ssh

16. Nuo dabar galima 4B valdyti nuotoliniu būdu, per internetą ir pasinaudojant SSH, Putty:

A screenshot of a computer

Description automatically generated

1. Prisijungimas prie Raspberry Pi 4B pasinaudojant Putty programa

17. Jeigu 4B yra 2 arba 4 GB, reikėtų sukurti „swap file“ – tai yra failas, kuris apibrėžia kiek SD kortelėje galima išnaudoti atminties kaip RAM:

free -h

sudo fallocate -l 2G /swapfile

ls -lh /swapfile

sudo chmod 600 /swapfile

ls -lh /swapfile

sudo mkswap /swapfile

sudo swapon /swapfile

sudo swapon --show

echo '/swapfile none swap sw 0 0' | sudo tee -a /etc/fstab

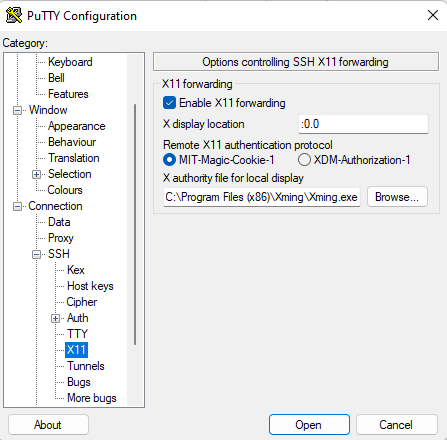
sudo reboot

18. Kad kompiuteryje būtų galima matyti ROS modulių duomenis per GUI, reikia įsirašyti X11 aplikacijas:

sudo apt install x11-apps

19. Kompiuteryje reikia parsisiųsti XMing, kad šias aplikacijas atvaizduoti asmeniniame kompiuteryje: <https://sourceforge.net/projects/xming/>

20. Iš naujo paleidus Putty programą, prieš jungiantis prie 4B reikia nurodyti šiuos X11 nustatymus:



1. X11 programų paleidimo nustatymai Putty programoje

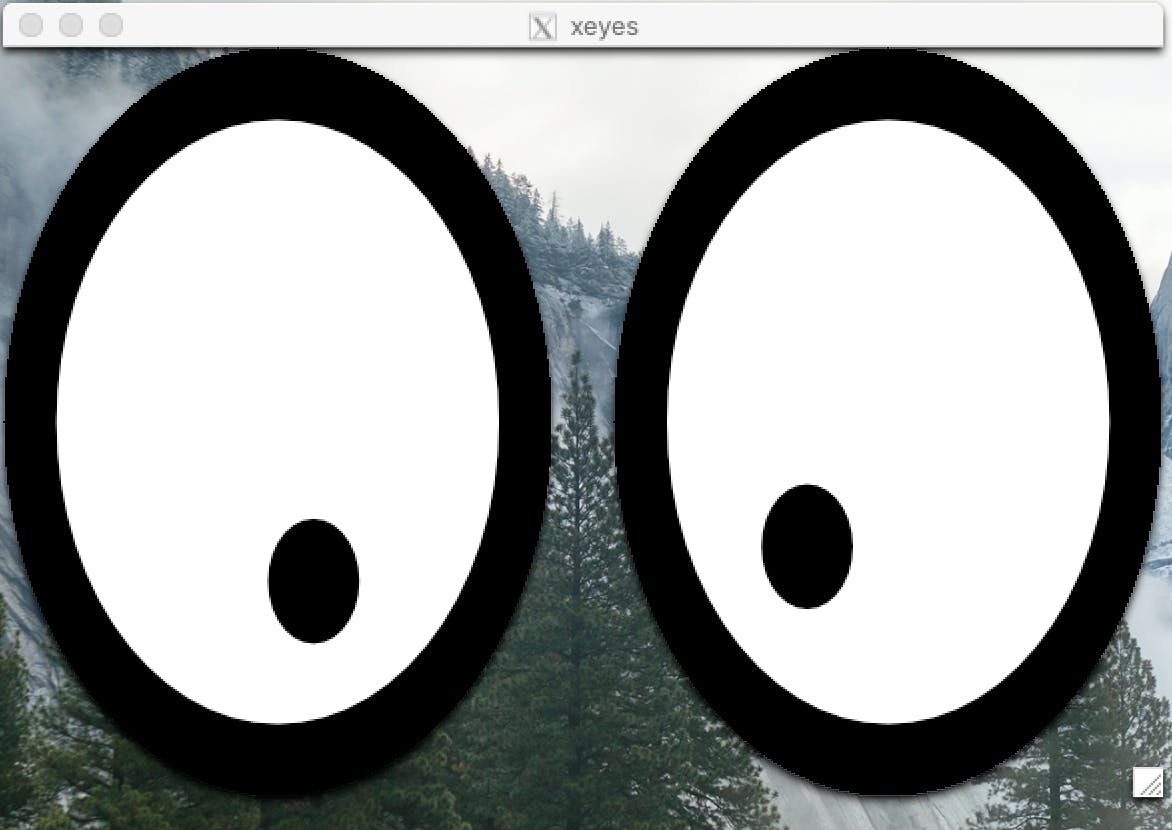
21. Dabar prisijungus prie 4B, galima įvesti šią komandą į terminalą ir bus atvaizduota sąsaja:

xeyes

22. Jeigu išmetama klaida „cannot open display“, reikia nustatyti DISPLAY kintamąjį:

export DISPLAY='localhost:0.0'

23. Viską sėkmingai atlikus turėtume matyti naujame lange šį vaizdą:



1. X11 programos pavyzdys

Sėkmingai buvo įsirašyta Ubuntu operacinė sistema, sukonfigūruoti interneto ryšio nustatymai bei paruošta sistema atvaizduoti programas per SSH.

* + 1. ROS įrašymas

ROS įsirašymo žingsniai:

1. Nustatyti 4B, kad galėtų priimti programinę įrangą iš *packages.ros.org*:

sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb\_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'

2. Nustatyti raktus parsisiųsti operacinei sistemai:

sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654

3. Parsisiųsti reikalingus atnaujinimus:

sudo apt update

4. Įsirašyti ROS Noetic Desktop Full (su visomis simuliacijos programomis):

**Pastebėjimas**: procesas gali trukti iki valandos priklausomai nuo interneto greičio.

sudo apt install ros-noetic-desktop-full

5. Tada reikia įsirašyti reikalingas papildančias bibliotekas:

sudo apt install python3-rosdep python3-rosinstall python3-rosinstall-generator python3-wstool build-essential  
sudo apt install python3-rosdep

sudo rosdep init

rosdep update

Sėkmingai įsirašius ROS, jo veikimą galima patikrinti taip:

1. Pirmame terminalo lange įvesti:

echo "source /opt/ros/noetic/setup.bash" >> ~/.bashrc

source ~/.bashrc

roscore

2. Antrame terminalo lange įvesti:

rosrun turtlesim turtlesim\_node

3. Turėtų atsidaryti naujas GUI langas su vėžliuku:

A screenshot of a computer

Description automatically generated

1. TurtleSim GUI pavyzdys
   1. RPLidar ir Hector\_SLAM integracija

Aplinkoje kliūčių nustatymui naudojamas yra Lidar A1M8, jo dokumentaciją galima rasti čia: <https://bucket-download.slamtec.com/e680b4e2d99c4349c019553820904f28c7e6ec32/LM108_SLAMTEC_rplidarkit_usermaunal_A1M8_v1.0_en.pdf>

Lidar prie 4B bus jungiamas per USB jungtį. Jo veikimą įgalins RPLidar ROS modulis, kurio dokumentaciją galima rasti čia: <https://github.com/Slamtec/rplidar_ros>

SLAM aplinkos nustatymui yra naudojamas Hector\_SLAM modulis, kurio dokumentaciją galima rasti čia: <http://wiki.ros.org/hector_slam>

Toliau nurodyti Lidar prisijungimo bei modulių įrašymo žingsniai:

1. Jungiame Lidar pasinaudojant USB į 4B.

**Pastebėjimas**: jis pradės suktis, tai yra įprastas veikimas bei įrodymas jog įrenginys funkcionuoja.

2. Tada reikia nustatyti, jog tik naudotojai su administratoriaus teisėmis gali valdyti sensorių:

ls -l /dev | grep ttyUSB  
sudo chmod 666 /dev/ttyUSB0  
ls -l /dev | grep ttyUSB

3. Tada susikuriame Catkin darbo aplanką (Catkin yra ROS programinė įranga skirta sukompiliuoti modulius):

mkdir -p ~/catkin\_ws/src

cd ~/catkin\_ws/

4. Tada inicijuojame darbo aplinką:

catkin\_init\_workspace

5. Kiekvieną kartą dirbant su moduliais Catkin aplinkoje reikia nurodyti Catkin nustatymų failą:

echo "source $HOME/catkin\_ws/devel/setup.bash" >> ~/.bashrc

6. Pereiname į */src* aplanką bei parsisiunčiame RPLidar\_ROS github aplanką:

cd src

sudo git clone https://github.com/Slamtec/rplidar\_ros.git  
cd ~/catkin\_ws/

7. Svarbu: šis žingsnis nėra įtrauktas oficialiose instrukcijose, tačiau */catkin\_ws* aplanke yra sukurtas CMakeLists.txt failas, dėl kurio nepavyks sukompiliuoti modulių. Jį yra saugu pašalinti:

sudo rm -rf CMakeLists.txt

8. Sukompiliuojame modulius:

**Pastebėjimas**: tai gali užtrukti iki valandos laiko.

catkin\_make

9. Iš naujo nurodome nustatymų šaltinį:

source devel/setup.bash

10. Šiame terminale paleidžiame ROS pagrindinį procesą:

roscore

11. Naujame terminale atsidarome Catkin darbo aplinkos aplanką:

cd ~/catkin\_ws/

12. Nurodome nustatymų šaltinį:

source devel/setup.bash

13. Paleidžiame RPLidar ROS skenavimo procesą:

roslaunch rplidar\_ros rplidar.launch

14. Naujame terminale galima paleisti GUI rezultatų atvaizdavimą:

source /opt/ros/noetic/setup.bash

rviz

15. Įsirašome Hector\_SLAM:

cd ~/catkin\_ws/

cd src

sudo git clone https://github.com/tu-darmstadt-ros-pkg/hector\_slam.git

sudo nano /rplidar\_hector\_slam/hector\_slam/hector\_mapping/launch/mapping\_default.launch

16. Šiame faile pakoreguojame eilutes:

Paskutinė nuo galo:

<node pkg="tf" type="static\_transform\_publisher" name="base\_to\_laser\_broadcaster" args="0 0 0 0 0 0 base\_link laser 100" />

Trečia eilutė:

<arg name="base\_frame" default="base\_link"/>

Ketvirta eilutė:

<arg name="odom\_frame" default="base\_link"/>

17. Išsaugome failą.

18. Tada koreguojame kitą failą:

sudo nano /rplidar\_hector\_slam/hector\_slam/hector\_slam\_launch/launch/tutorial.launch

Ir koreguojame trečią eilutę:

<param name="/use\_sim\_time" value="false"/>

19. Išsaugome failą.

20. Atsidarome tris terminalo langus:

Terminalas 1:

cd ~/catkin\_ws/

source devel/setup.bash

roscore

Terminalas 2:

cd ~/catkin\_ws/

source devel/setup.bash

roslaunch rplidar\_ros rplidar.launch

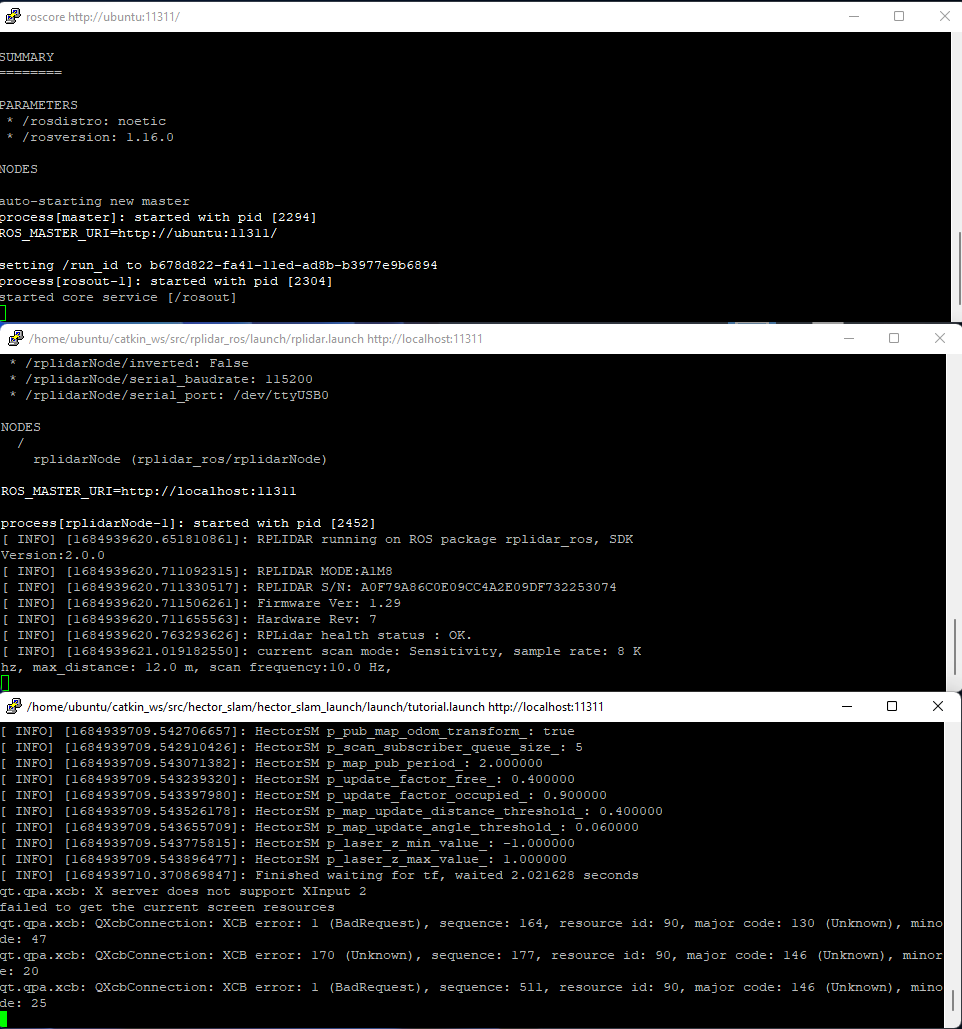
Terminalas 3:

cd ~/catkin\_ws/

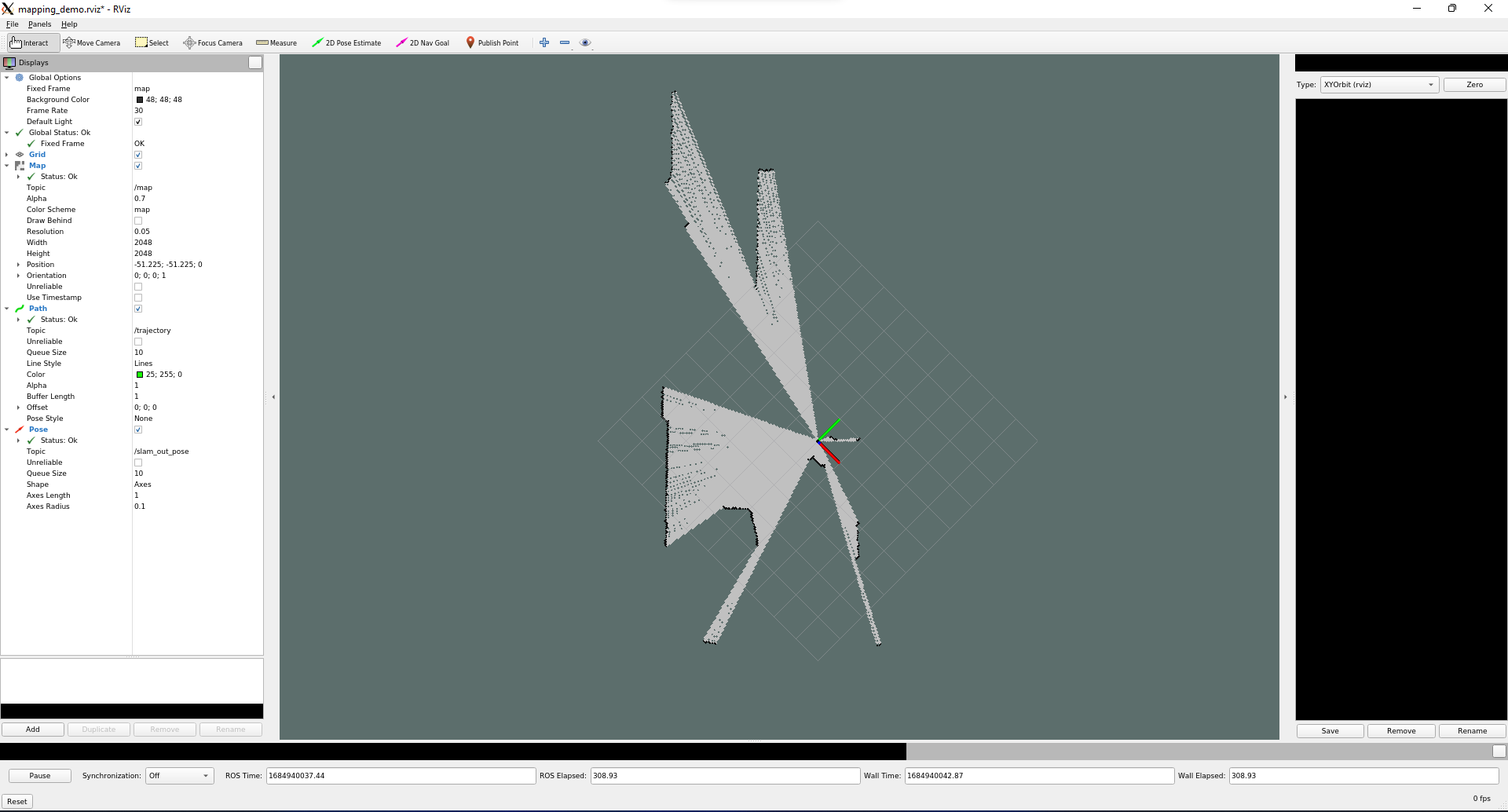
source devel/setup.bash

roslaunch hector\_slam\_launch tutorial.launch

Padarius šiuos veiksmus turėtų atsidaryti naujas GUI langas, kuriame matomas aplinkos žemėlapio kūrimas pasinaudojant Hector\_SLAM moduliu ir Lidar sensoriumi.



1. Naudojami Putty terminalai



1. Hector\_SLAM sukurtas aplinkos žemėlapis
   1. Paketų kūrimas

Atlikus sistemos paruošimo veiksmus, sekantis tikslas yra sukurti savo ROS modulius. Buvo rasta, jog pasinaudojant Visual Studio Code programa, galima per SSH prisijungti prie 4B, atsidaryti Catkin darbo aplinkos aplankus ir juose kurti naujus failus bei koreguoti esamus.

Pasinaudojant žemiau esančiais informacijos šaltiniais mums pavyko pasiruošti Visual Studio Code aplinką, sukurti naują modulį, bei jį paleisti, kad vykdytų procesą:



1. Pavyzdinės Catkin modulio programos kodas bei failų struktūra

Visual Studio Code panaudojimas kuriant ROS modulius: <https://github.com/lzptr/VS_Code_ROS>

Kaip sukurti ROS modulį:

<https://varhowto.com/cpp-ros-catkin-package/>

<https://medium.com/swlh/7-simple-steps-to-create-and-build-our-first-ros-package-7e3080d36faa>

Oficialios instrukcijos kaip sukurti ROS modulį:

<http://wiki.ros.org/ROS/Tutorials/CreatingPackage>

1. JUDĖJIMO VALDYMAS

Ratų valdymas atliekamas prijungus HBridge valdiklį prie 4B per šiuos GPIO:

|  |  |  |
| --- | --- | --- |
| **H Bdrige jungtis** | **GPIO jungtis** | **Kintamojo pavadinimas kode** |
| Enable A | GPIO 1 | enablePin |
| Enable B | GPIO 1 | enablePin |
| A1 | GPIO 4 | left1Pin |
| A2 | GPIO 5 | left2Pin |
| B1 | GPIO 21 | right1Pin |
| B2 | GPIO 22 | right2Pin |
| GND | GND (any) | N/A |
| VCC | N/A\* | N/A |

1. HBridge ir 4B sujungimas su kintamųjų pavadinimais.

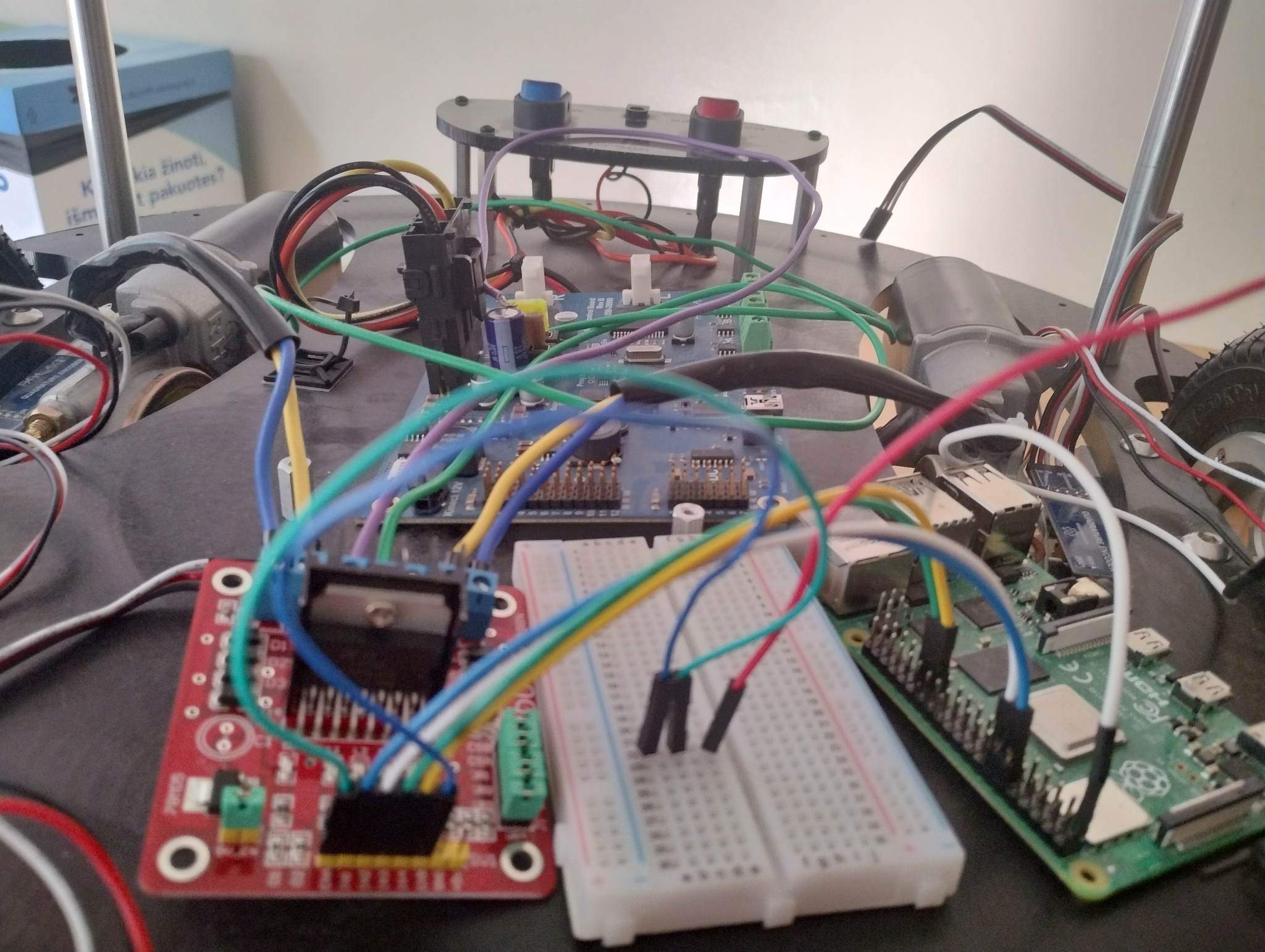
\* - VCC skirtas perduoti galiai, bet kilo problemų su H Bridge uzkaitimu, todėl naudotas atskiras galios šaltinis.

*A* ir *B* jungtys simbolizuoja atitinkamai kairį ir dešinį ratus. Jų veikimas priklauso nuo srovės į 1 ir 2 jungtis:

|  |  |  |
| --- | --- | --- |
| **1 jungties srovė** | **2 jungties srovė** | **Elgesys** |
| Nėra | Nėra | Ratas nejuda |
| Yra | Nėra | Sukasi palei laikrodžio rodyklę |
| Nėra | Yra | Sukasi prieš laikrodžio rodyklę |
| Yra | Yra | Dokumentacijoj neapibrėžta, ratas nejuda |

1. HBridge jungčių elgesys pagal sroves

*Enable* jungtys įjungia ratų veikimą. Kadangi nebuvo rasta situacijos, kada reikėtų išjungti tik vieną ratą, abu ratai pajungti prie bendros *Enable* jungties.



1. Raspberry Pi 4B jungimo prie H Bridge pavyzdys

Žemiau esančio kode yra naudojamas WiringPi, jo dokumentaciją galima rasti čia: <https://github.com/wbeebe/WiringPi>

#include <iostream>

#include <fstream>

#include <signal.h>

#include <wiringPi.h> // to control Raspberry Pi digital pins

#include "src/datetime.h" // to have current time for logging

// Raspberry PI prerequisites:

// install WiringPi

using namespace jed\_utils; // for datetime

using namespace std;

#define enablePin 1

#define left1Pin 4

#define left2Pin 5

#define right1Pin 21

#define right2Pin 22

bool inputTextPrinted = false;

class Port {

private:

int portNumber;

bool isOn;

bool out;

bool high;

public:

Port(int portNumber, bool out, bool high) {

this->portNumber = portNumber;

this->out = out;

this->high = high;

isOn = false;

}

bool IsOn() {

return isOn;

}

bool IsHigh() {

return high;

}

void TurnOn() {

pinMode(portNumber, OUTPUT);

isOn = true;

cout << "DEBUG: " << "gpio mode " << portNumber << " " << (out ? "out" : "in") << endl;

}

void ToggleHigh() {

SetHigh(!high);

}

void SetHigh(bool high) {

if (high) {

digitalWrite(portNumber, HIGH);

}

else {

digitalWrite(portNumber, LOW);

}

this->high = high;

cout << "DEBUG: " << "gpio write " << portNumber << " " << (high ? "1" : "0") << endl;

}

};

class HBridgePair {

private:

Port\* one;

Port\* two;

public:

HBridgePair(Port\* one, Port\* two) {

this->one = one;

this->two = two;

}

void Initialize() {

one->TurnOn();

two->TurnOn();

}

void Stop() {

one->SetHigh(false);

two->SetHigh(false);

}

void Forward() {

one->SetHigh(true);

two->SetHigh(false);

}

void Reverse() {

one->SetHigh(false);

two->SetHigh(true);

}

void ChangeDirection() {

if (one->IsHigh() == two->IsHigh()) return; //changing direction means nothing if you dont move

one->ToggleHigh();

two->ToggleHigh();

}

};

class WheelControl {

private:

Port\* enable;

HBridgePair\* RightWheel;

HBridgePair\* LeftWheel;

public:

WheelControl(int EnablePort, HBridgePair\* LeftWheel, HBridgePair\* RightWheel) {

enable = new Port(EnablePort, true, true);

this->RightWheel = RightWheel;

this->LeftWheel = LeftWheel;

}

void Initialize() {

enable->TurnOn();

RightWheel->Initialize();

LeftWheel->Initialize();

}

void TurnOn() {

enable->SetHigh(true);

}

void Forward() {

RightWheel->Forward();

LeftWheel->Forward();

TurnOn();

}

void Reverse() {

RightWheel->Reverse();

LeftWheel->Reverse();

TurnOn();

}

void TurnLeft() {

RightWheel->Forward();

LeftWheel->Reverse();

TurnOn();

}

void TurnRight() {

RightWheel->Reverse();

LeftWheel->Forward();

TurnOn();

}

void Stop() {

RightWheel->Stop();

LeftWheel->Stop();

enable->SetHigh(false);

}

};

void GetInput(WheelControl\* wheelControl);

bool ctrl\_c\_pressed;

void ctrlc(int) {

ctrl\_c\_pressed = true;

}

int main(void)

{

std::cout << jed\_utils::datetime().to\_string() << " App started\n";

// set-up movement controls

wiringPiSetup();

std::cout << jed\_utils::datetime().to\_string() << " Setting up wheel controls" << endl;

WheelControl\* wheelControl;

Port\* A1 = new Port(left1Pin, true, false);

Port\* A2 = new Port(left2Pin, true, false);

HBridgePair\* A = new HBridgePair(A1, A2);

Port\* B1 = new Port(right1Pin, true, false);

Port\* B2 = new Port(right2Pin, true, false);

HBridgePair\* B = new HBridgePair(B1, B2);

wheelControl = new WheelControl(enablePin, A, B);

wheelControl->Initialize();

// create detection of ctrl + c pressed

signal(SIGINT, ctrlc);

while (1) {

GetInput(wheelControl);

if (ctrl\_c\_pressed) {

break;

}

}

// end program

std::cout << jed\_utils::datetime().to\_string() << " App cancellation initiated\n";

return 0;

}

void GetInput(WheelControl\* wheelControl) {

if (!inputTextPrinted) {

inputTextPrinted = true;

std::cout << jed\_utils::datetime().to\_string() << " Input Command:";

}

string command;

std::cin >> command;

if (command == "Forward") {

inputTextPrinted = false;

wheelControl->Forward();

return;

}

if (command == "Reverse") {

inputTextPrinted = false;

wheelControl->Reverse();

return;

}

if (command == "TurnLeft") {

inputTextPrinted = false;

wheelControl->TurnLeft();

return;

}

if (command == "TurnRight") {

inputTextPrinted = false;

wheelControl->TurnRight();

return;

}

if (command == "Stop") {

inputTextPrinted = false;

wheelControl->Stop();

return;

}

cout << jed\_utils::datetime().to\_string() << " Unrecognized. Firetruck\n";

}

Šį kodą būtų galima perdaryti į ROS palaikomą modulį, kuris laukia informacijos iš valdymo topic ir pateikus atitinkamą komandą – valdyti ratų variklius iš sistemos vidaus.

Valdiklis roboto komandas suprogramuotas priimti per command line. Palaikomos funkcijos:

|  |  |
| --- | --- |
| **Komanda** | **Elgesys** |
| Forward | Abu ratai sukasi į priekį |
| Reverse | Abu ratai sukasi atgal |
| TurnRight | Dešinys ratas sukasi į priekį, kairys - atgal |
| TurnLeft | Kairys ratas sukasi atgal, dešiny – į priekį |
| Stop | Abu ratai sustabdomi |

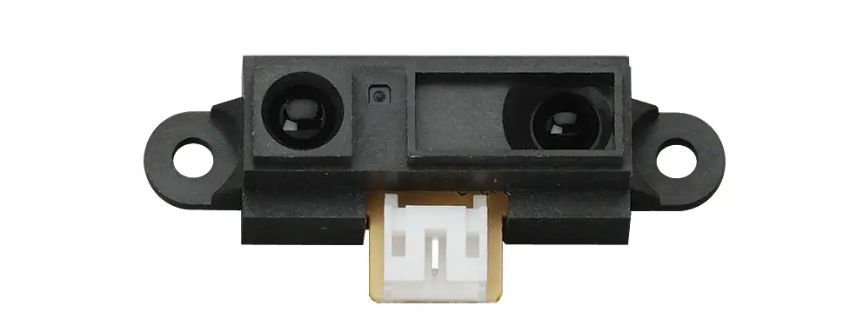
1. Valdiklio komandos roboto ratams valdyti
2. JUTIKLIŲ VALDYMAS

Šiame skyriuje aprašomi IR ir ultragarsiniai jutikliai bei jų valdymas.

* 1. Informacija apie IR jutiklius

**Pavadinimas:** Sharp GP2Y0A21YK0F

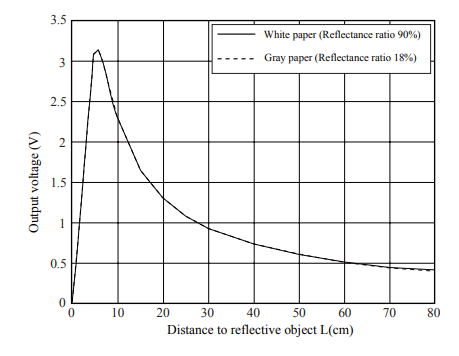
**Išvaizda:**



1. Infraraudonųjų spindulių jutiklis

**Gamintojo informacija:** <https://www.parallax.com/product/sharp-gp2y0a21yk0f-ir-distance-sensor-10-80-cm/>

Pagal tai kaip arti yra aptikta kliūtis, šis jutiklis grąžina analoginę reikšmę nuo 0.4 iki 3.3V (kuo didesnė reikšmė – tuo kliūtis yra arčiau). Atstumo ir voltažo kreivė nėra tolygi, todėl yra didelė tikimybė jog bus poreikis iššsivesti savo skaičiavimo algoritmą norint turėti tiksliausius rezultatus.



1. Jutiklio atstumo ir voltažo ryšio kreivė

Pastebėta, kad voltažas kinta ir dėl kliūties spalvos. Labiau patariama išnaudoti šį jutiklį algoritminiam aprašymui ar kliūtis yra matoma pagal nustatytus atstumo rėžius.

* 1. Informacija apie ultragarsinius jutiklius

**Pavadinimas:** PING)))

**Išvaizda:**



1. Ultragarsinis jutiklis

**Gamintojo informacija:** <https://www.parallax.com/product/ping-ultrasonic-distance-sensor/>

Ultragarsiniai jutikliai laukia informacijos perdavimo užklausos (toliau - *Ping*) iš mikrovaldiklio ir tada grąžina signalą, kurio ilgis parodo kiek truko banga atsimušti nuo kliūties ir sugrįžti. Sekant laiką ir žinant garso greitį ore, galima apskaičiuoti atstumą iki kliūties.

* 1. Mikrovaldiklio pasirinkimas

Pagrindiniai jutiklių įvesties ir išvesties (toliau – *GPIO*) reikalavimai yra šie:

* išvesties įtampa – 5V;
* skaitmeninės įvesties įtampa – iki 5V;
* analoginės įvesties įtampa – iki 3.3V.

Kadangi 4B neturi analoginių įvesčių, tenka rinktis naudotis papildomą mikrovaldiklį. Atsižvelgiant į reikalavimus, tinkami variantai yra: *Arduino Uno* arba *Raspberry Pi Pico*.

Dėl asmeninio susidomėjimo išbandyti naują mikrovaldiklį, buvo pasirinktas Pico.

* 1. Jutiklių sujungimas į Raspberry Pi Pico

Jutiklių sujungimas į GPIO jungtis yra nurodytas sekančioje lentelėje:

|  |  |  |
| --- | --- | --- |
| **Pavadinimas** | **GPIO numeris** | **Įtampa (V)** |
| VCC (all) | VBUS | 5.0 |
| Ground (common) | GND (any) | 0 |
| Infrared 1 Data | GPIO26 (A0) | 0.4-3.3 |
| Infrared 2 Data | GPIO27 (A1) | 0.4-3.3 |
| Infrared 3 Data | GPIO28 (A2) | 0.4-3.3 |
| Ultrasound 1 Data | GPIO22 | 2.5-5.0\* |
| Ultrasound 2 Data | GPIO21 | 2.5-5.0\* |

1. Jutiklių sujungimas Pico mikrovaldiklyje

\* yra laukiamas signalas iš mikrovaldiklio prieš grąžinant duomenis. Tada yra grąžinamas tokios pat įtampos elektros srovės signalas koks ir buvo signale.

* 1. Duomenų apdorojimas Raspberry Pi Pico valdiklyje

Pico apdoroja tik ultragarsinio jutiklio duomenis – pagal trukmę apskaičiuodamas atstumą centimetrais. IR jutiklio duomenų apdorojimą palieka priimančiam mikrovaldikliui išnaudoti pagal numatomas paskirtis, suapvalinant gautos elektros srovės įtampos reikšmes iki 2 skaičių po kablelio.

Jutiklių duomenys yra skaitomi sinchroniškai (vienas po kito) stabilumui garantuoti. Jeigu įmanoma, patariama asinchronizuoti šitą procesą.

Pico programa duomenis grąžina per nuoseklųjį prievadą (toliau - *Serial*), su intencija kad priimantis prietaisas būtų Pico prisijungęs per USB jungtį.

Duomenys buvo grąžinami pagal fizinį jutiklių išdėstymą ant drono, su reikšmėmis atskirtomis kableliais.

Šabloniškas aprašas: „{Infrared 1};{Ultrasound 1};{Infrared 2};{Ultrasound 2};{Infrared 3};”.

Duomenų pabaigą indikuoja specialusis simbolis: „\n“.

Siunčiamas pavyzdys būtų: „2.40;444;1.21;201;0.8;\n”.

Tai iššifruojama į:

* Infrared 1 sensorius grąžino: 2.40 V;
* Ultrasound 1 sensorius aptiko objektą už: 444 cm;
* Infrared 2 sensorius grąžino: 1.21 V;
* Ultrasound 2 sensorius aptiko objektą už: 201 cm;
* Infrared 3 sensorius grąžino: 0.8 V.

Pico kodas buvo rašomas pasitelkiant VisualGDB paketus su Visual Studio.

**Paketo puslapis:** <https://visualgdb.com>

**Algoritmas:** <https://github.com/Coolest-KU-students/Raspberry-GPIO-Wheel-control/blob/Lukas-branch/PicoSensors3%20-%20good/PicoSensors3/hello_usb.c>

#include "pico/stdlib.h"

#include "hardware/adc.h"

#include "hardware/gpio.h"

// Constants for GPIO pins

const uint LED\_PIN = PICO\_DEFAULT\_LED\_PIN; // On-board LED pin

const uint PING\_PIN\_1 = 22; // GPIO pin for PING sensor

const uint PING\_PIN\_2 = 21;

const uint IR\_PIN\_1 = 0; // GPIO26 pin for IR sensor

const uint IR\_PIN\_2 = 1; // GPIO27

const uint IR\_PIN\_3 = 2; // GPIO28

// Constants for ADC settings

const uint ADC\_RESOLUTION\_BITS = 12; // ADC resolution in bits

const float ADC\_REFERENCE\_VOLTAGE = 3.3f; // ADC reference voltage in volts

int main() {

uint32\_t IRValues[3] = {0 , 0 , 0};

float USValues[2] = {0.0, 0.0};

// Initialize standard I/O

stdio\_init\_all();

// Initialize ADC

adc\_init();

adc\_gpio\_init(IR\_PIN\_1);

adc\_gpio\_init(IR\_PIN\_2);

adc\_gpio\_init(IR\_PIN\_3);

gpio\_init(PING\_PIN\_1);

gpio\_init(PING\_PIN\_2);

gpio\_set\_dir(LED\_PIN, GPIO\_OUT);

while (true) {

for (int i = 0; i < 3; i++) { //IR inputs

adc\_select\_input(i); // Select ADC input 0 (GPIO26)

uint16\_t adc\_value = adc\_read();

IRValues[i] = adc\_value;

}

For (int i = 0; i< 2; i++) { //US inputs

uint CURRENT\_PIN = i == 0 ? PING\_PIN\_1 : PING\_PIN\_2;

gpio\_set\_dir(CURRENT\_PIN, GPIO\_OUT);

gpio\_put(CURRENT\_PIN, 1);

sleep\_us(10); // Pulse trigger for 10 us

gpio\_put(CURRENT\_PIN, 0);

gpio\_set\_dir(CURRENT\_PIN, GPIO\_IN);

uint32\_t waitingTime = time\_us\_32();

while (!gpio\_get(CURRENT\_PIN) && time\_us\_32() - waitingTime < 100000) // If it doesn't respond in 100 miliseconds - skip

{} // Wait for echo to start

uint32\_t start = time\_us\_32();

if (start - waitingTime > 100000) {

USValues[i] = 0;

continue; //next loop

}

while (gpio\_get(CURRENT\_PIN)) {} // Wait for echo to end

uint32\_t end = time\_us\_32();

uint32\_t pulse\_duration = end - start;

float distance = pulse\_duration \* 0.0343f / 2; // Calculate distance in cm

USValues[i] = distance;

}

printf("%i",IRValues[0]);

printf(";");

printf("%.2f",USValues[0]);

printf(";");

printf("%i",IRValues[1]);

printf(";");

printf("%.2f",USValues[1]);

printf(";");

printf("%i",IRValues[2]);

printf(";");

printf("\n");

}

return 0;

}

**Pastebėjimas:** Pico klaidų aptikimas (angl. *debugging*, toliau - *debugging*) yra komplikuotas, kadangi yra galimi tik du variantai: naudoti indikacinę LED lemputę (LED\_PIN), kuri yra integruota į Pico mikrovaldiklio plokštę arba atlikti nuskaitymą per GPIO prievadus ir antrą Pico (<https://www.youtube.com/watch?v=0i2gLeBal9Y>).

* 1. Duomenų priėmimas Raspberry Pi 4B valdiklyje

Duomenų priėmimas turėtų būti atliekamas per *ROS* modulį 4B mikrovaldiklyje. Žemiau aprašytas procesas nuskaito duomenis iš Serial\_0 (pirmoji USB jungtis, gali keistis nuo naudojamos jungties), juos konvertuoja į PicoData objektą, kuris toliau gali būti naudojamas duomenų perdavimui ROS sistemoje.

#define SerialPort SERIAL\_0

class PicoData {

public:

unsigned int IR\_1;

unsigned int IR\_2;

unsigned int IR\_3;

float US\_1;

float US\_2;

static PicoData\* LastData;

PicoData() {

IR\_1 = 0;

IR\_2 = 0;

IR\_3 = 0;

US\_1 = 0;

US\_2 = 0;

}

PicoData(char\* picoData) {

int placement = 1;

char\* accumulated = new char[0];

int size = 0;

for (int i = 0; i < sizeof(picoData); i++) {

char current = picoData[i];

if (current == ';') {

Assign(placement, accumulated);

placement++;

size = 0;

}

size++;

char\* newArray = new char[size];

for (int j = 0; j < size - 1; j++) {

newArray[j] = accumulated[j];

}

newArray[size - 1] = current;

accumulated = newArray;

}

}

void Assign(int placement, char\* value) {

if (placement == 1) {

IR\_1 = CharToFloat(value);

}

if (placement == 2) {

US\_1 = CharToInt(value);

}

if (placement == 3) {

IR\_2 = CharToFloat(value);

}

if (placement == 4) {

US\_2 = CharToInt(value);

}

if (placement == 5) {

IR\_3 = CharToFloat(value);

}

}

int CharToInt(char\* value) {

try {

return stoi(value);

}

catch (exception e) {

return 0;

}

}

float CharToFloat(char\* value) {

return (float)atof(value);

}

};

PicoData\* PicoData::LastData = new PicoData();

PicoData\* ReadPicoData() {

// !open serial in main!

int size = 0;

char\* returnable = new char[0];

while (Serial.available()) {

char next = Serial.read();

if (next == '\n') {

delete(PicoData::LastData);

PicoData\* newData = new PicoData(returnable);

PicoData::LastData = newData;

return newData;

}

size++;

char\* newCharArray = new char[size];

for (int i = 0; i < size - 1; i++) {

newCharArray[i] = returnable[i];

}

newCharArray[size-1] = next;

delete(returnable);

returnable = newCharArray;

}

return PicoData::LastData;

}

Pasinaudojant ReadPicoData() metodu, galima arba agreguoti/manipuliuoti duomenimis, arba juos perduoti į kitus ROS paketus. Jį galima kviesti nuolatiniame cikle, ir jei naujų duomenų nėra, jis grąžins praeito ciklo duomenis. Atmintį pravalo pats objektas, todėl trinti grąžintų PicoData negalima.

1. TOLIMESNĖ PLĖTRA
2. Sukurti ROS modulį priimti jutiklių duomenis iš Pico.
3. Iš roboto valdymo kodo sukurti modulį, kuris klausytųsi sekančių žingsnių iš ROS ir pagal tai perduotų komandas į HBridge.
4. Jeigu reikia, sukurti klausantį ROS modulį, kuris priimtų Pico ir Hector\_SLAM modulių duomenis ir juos apdorotų.
5. Sukurti/rasti algoritmą, kuris gali altikti kelio paiešką (angl. *pathfinding*) su apdorotais duomenimis ir sekančius roboto žingsnius perduotų ROS moduliams.
6. Patobulinti roboto valdymo algoritmą, atsižvelgiant į ratų apsukų jutiklių duomenis.
7. Sukurti intraneto ryšiu (local WLAN) per API prienamą duomenų posistemę, pagal kurią būtų galima kurti naudotojo sąsaja.