KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS

Robotų programavimo technologijos (T125B114)

Inžinerinio projekto ataskaita

Atliko:

Audrius Puzinas IFF-8-1

Gabriele Butnoriūtė IFF-8-7

2021 m. gruodžio 21 d.

Priėmė:

doc. Rasa Brūzgienė

doc. Tomas Adomkus

KAUNAS 2021

Turinys

1. Užduotis

Projektas bus atliekamas CoppeliaSim simuliatoriuje. Robotas sukurtame labirinte ieškos kelio iki išėjimo. Labirinte robotas išvenginės kliūčių (sienų) bei kito roboto. Sutikus kitą robotą jie apsikeis žinutėmis ir seks sutikto roboto judėjimą. Pasiekęs labirinto galą robotas aptiks liniją ir seks ją iki jos pabaigos.

Naudojami sensoriai:

- Ultragarso(kliūties aptikimas)
- IR(žinutėms tarp robotų)
- Šviesos sensorius(linijos sekimas)
- IR(atstumo iki sienos palaikymas)

2. Reikalavimų užduočiai atlikti analizė ir roboto valdymo (misijos) scenarijus

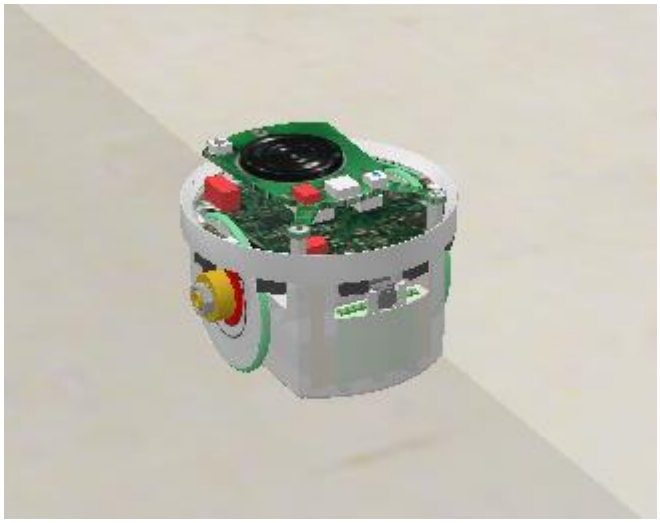
Užduotis susideda iš kelių pagrindinių etapų:

- Robotas turi ieškoti kelio iš labirinto
- Sutikęs linija robotas turi ją sekti
- Sutikės draugą turi jį sekti

Roboto misija - sekant sieną ištrūkti iš labirinto, surasti liniją ją sekti iki galo ir toliau (jei įmanoma) sekti sieną. Sutikus draugą robotas turi jį sekti, kol jį pames ir tuomet grįžti prie sienos sekimo.

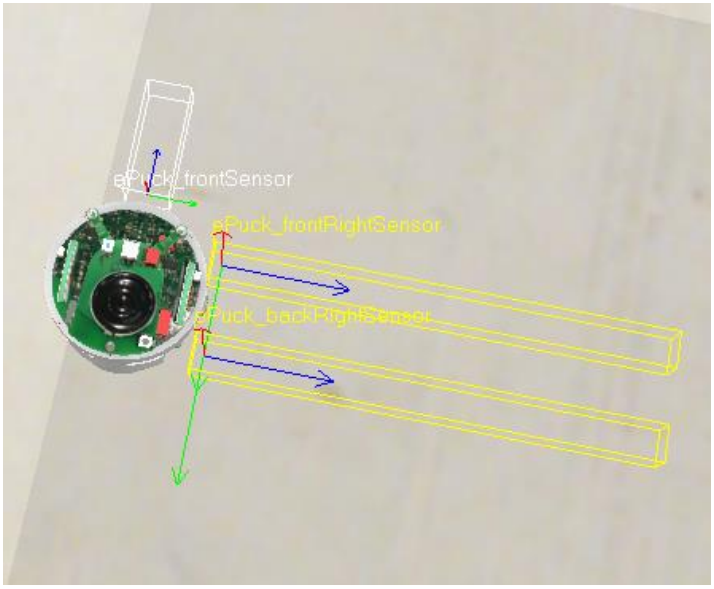
3. Modelio aprašymas

Kaip roboto bazė naudotas jau simuliacinėje aplinkoje paruoštas modelis „e-puck“ (pav.1) . Modelis turi du ratus.



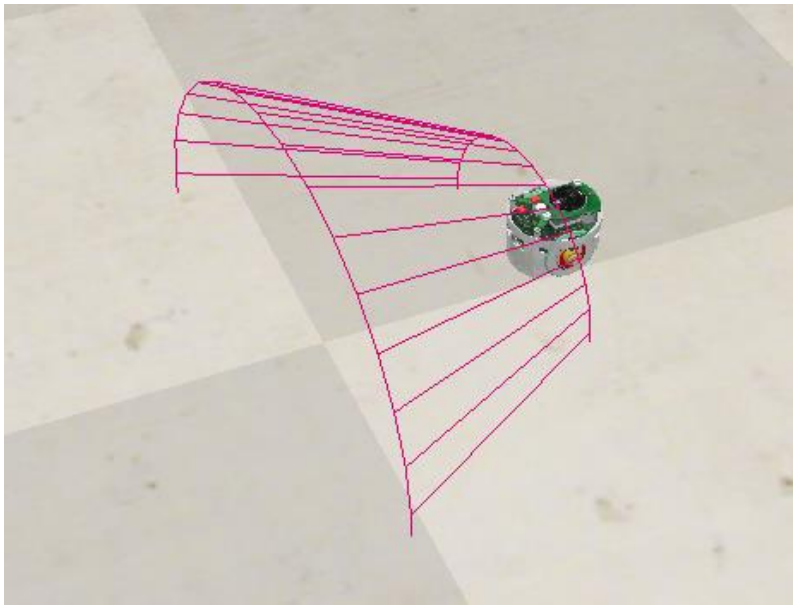
Pav. 1 Naudotas roboto modelis

Nuo duoto modelio buvo nuimti nereikalingi sensoriai bet pašalintas kodas. Pridėti du IR sensoriai dešinėje pusėje sienos sekimui (pav. 2) ir vienas ultragarso sensorius priekyje (pav.2). Ant roboto paliktas ir šviesos sensorius kuris funkcionuos kaip IR sensorius linijos sekimui(bus naudojami 3 pikseliai vietoj trijų atskirų sensorių).



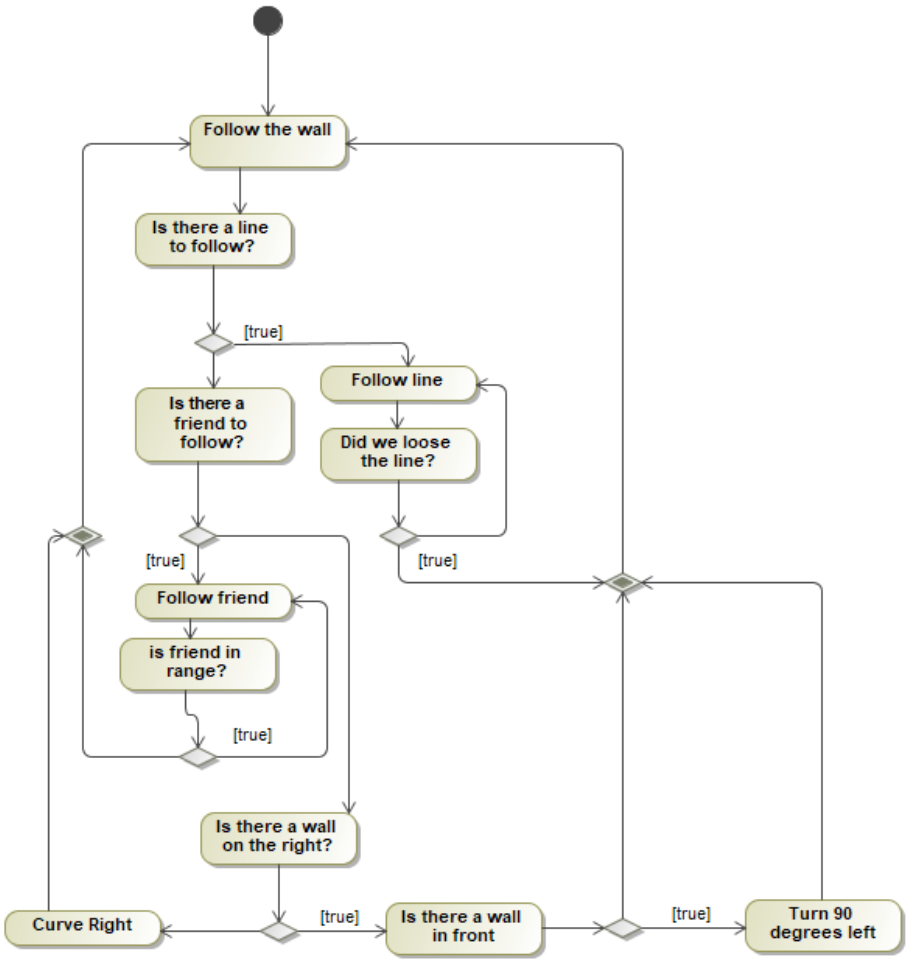
Pav. 2 Sensorių vietos

Kito roboto aptikimui panaudotas ultragarso sensorius, pritvirtintas priekyje roboto, aptinkantis tik kitą robotą (pav. 3).



Pav. 3 Priekinis sensorius kito roboto aptikimui

4. Roboto valdymo algoritmas ir jo aprašymas



Pav. 4 Roboto algoritmo veiklos diagrama

Robotas veikia pagal nustatytas būsenas:

- Sienos sekimas (Follow the wall) - robotas seka dešinę sieną
- Linijos sekimas (Follow line) - robotas seka liniją
- Draugo sekimas (Follow friend) - robotas seka sutiktą draugą
- Nuožulnus sukimasis į dešinę (Curve right) - robotas pasisuka dešinėn kai nebeturi sienos dešinėje
- 90 laipsnių pasisukimas į kairę (Turn 90 degrees left) - robotas pasisuka į kairę sutikęs sieną priekyje

Robotas turi būsenų prioritetą sudėliotą tokia tvarka: linijos sekimas, draugo sekimas, sienos sekimas ir pagal situacija nuožulnus sukimasis į dešinę arba 90 laipsnių pasisukimas į kairę (pav. 4), tai yra, jei yra linija kurią galima sekti, draugas nebus sekamas, jei yra draugas, nebus sekama siena.

5. Roboto valdymo programinis kodas

Roboto sensoriai, motorai bei konstantos nustatomos sysCall_init() funkcijoje, ji kviečiama vieną karta.

Toliau kiekvienos iteracijos metu kviečiama sysCall_sensing() funkcija kuri pagal roboto sensorius nustato koks roboto rėžimas (būsena) turi būti nustatytas ir jį nustato. Tuomet kviečiama sysCall_actuation() funkcija kurioje pagal nustatytą būseną nustatomi roboto ratų greičiai. Funkcijos sysCall_sensing() ir sysCall_actuation() viena po kitos kviečiamos tol kol simuliacija nėra sustabdoma.

```
function sysCall_init()
    corout = coroutine.create(coroutineMain)
    -- get handles of various components
    colorSensor = sim.getObjectHandle('ePuck_lightSensor')

    leftMotor = sim.getObjectHandle("ePuck_leftJoint")
    rightMotor = sim.getObjectHandle("ePuck_rightJoint")

    fRightSensor = sim.getObjectHandle("ePuck_frontRightSensor")
    bRightSensor = sim.getObjectHandle("ePuck_backRightSensor")

    frontSensor = sim.getObjectHandle("ePuck_frontSensor")
    backSensor = sim.getObjectHandle("ePuck_backSensor")

    beacon = sim.getObjectHandle("BEACON")
    beaconSensor = sim.getObjectHandle("ePuck_beaconSensor")

    --set inicial velocities
    sim.setJointTargetVelocity(leftMotor, 0.0)
    sim.setJointTargetVelocity(rightMotor, 0.0)
    -- set speeds
    maxSpeed = 2
    minSpeed = maxSpeed * 0.8
    direction = 1
    turnLeftSpeed, turnRightSpeed, minTurnTime = getTurnSpeeds()
    curveLeftSpeed, curveRightSpeed, minCurveTime = getCurveSpeeds()

    --create required constants
    distanceToWall = 2
    detectionDistance = 0.2
    state = 4
    lastTime = 0
end

getLightSensors = function()
    local img = sim.getVisionSensorImage(colorSensor)
    return {img[1], img[22], img[94]}
```

```
end

function sysCall_actuation() --main method of robot control, sets wheel velocities
wL and wR
    local wL, wR
    if (state == 1) then
        wL, wR = followWall()
    elseif (state == 2) then
        wL = turnLeftSpeed
        wR = turnRightSpeed
    elseif (state == 3) then
        wL = curveLeftSpeed
        wR = curveRightSpeed
    elseif (state == 4) then
        wL, wR = followLine()
    elseif (state == 5) then
        local targetInRange
        wL, wR, targetInRange = followOther()
        -- print("Target in range", targetInRange)
        if not targetInRange then
            state = 1
        end

    else
        wL = 0
        wR = 0
    end

    sim.setJointTargetVelocity(leftMotor, wL)
    sim.setJointTargetVelocity(rightMotor, wR)

    if coroutine.status(corout) ~= 'dead' then
        local ok, errorMsg = coroutine.resume(corout)
        if errorMsg then
            error(debug.traceback(corout, errorMsg), 2)
        end
    end
```

```
end

function lineDetected(lightSens)
    if ((lightSens[1] < 0.5) or (lightSens[2] < 0.5) or (lightSens[3] < 0.5)) then
        return true
    end
    return false
end

function beaconDetected() -- method to check if robot firend is detected
    local detected = sim.checkProximitySensor(beaconSensor, beacon)
    return detected == 1
end

function sysCall_sensing() -- main method to set states of robot. Uses sensors to
determine state and then sets it
    local isOnLine = lineDetected(getLightSensors())
    local wallInFront = isWallInFront()
    local wallOnSide = isWallOnRightSide()

    if not isOnLine then
        handleStateResetAfterLine(isOnLine)

        if (state == 1) then
            handleFirstState(wallInFront, wallOnSide)
        elseif (state == 2) then
            handleSecondState(wallOnSide)
        elseif (state == 3) then
            handleThirdState(wallOnSide)
        elseif (state == 5) then
            handleFifthState()
        end
    else
        handleForthState()
    end
end
```

```
function isWallOnRightSide()
    rightSensorVal = getDistance(fRightSensor)
    if (rightSensorVal > distanceToWall - 0.001) then
        return false
    end
    return true

end

function isWallInFront()
    local frontSensorVal = getDistance(frontSensor)

    if (frontSensorVal < distanceToWall) then
        return true
    end
    return false
end

function handleStateResetAfterLine(onLine)
    if (state == 4 and not onLine) then
        state = 1
    end
end

function handleFirstState(wallInFront, wallOnSide)
    lastTime = sim.getSimulationTime()
    local isBeaconDetected = beaconDetected()

    if isBeaconDetected then
        state = 5
        return
    end

    if wallInFront then
        state = 2
    end
```

```
    if not wallOnSide then
        state = 3
    end
end

-- the wolowing two methods use lastTime global variable to determine if the turn
is complete and the control should be given to state 1(follow wall)
function handleSecondState(wallOnSide)
    local timeElapsed = ((sim.getSimulationTime() - lastTime) > minTurnTime)
    if wallOnSide and timeElapsed then
        state = 1
    end
end

function handleThirdState(wallOnSide)
    local timeElapsed = ((sim.getSimulationTime() - lastTime) > minCurveTime)
    if wallOnSide and timeElapsed then
        state = 1
    end
end

function handleForthState()
    state = 4
end

function handleFifthState()
end

function followLine() -- method to determine speeds for line following
    local wL = sim.getJointTargetVelocity(leftMotor)
    local wR = sim.getJointTargetVelocity(rightMotor)
    lightSens = getLightSensors()
    if lightSens and (lineDetected(lightSens)) then
        if (lightSens[1] > 0.5) then
            wL = maxSpeed
        else
            wL = maxSpeed * 0.25
```

```
        end
        if (lightSens[3] > 0.5) then
            wR = maxSpeed
        else
            wR = maxSpeed * 0.25
        end
    end
    return wL, wR
end

function getSpeedsBasedOnVector(baseSpeed, vector) -- helper method to detrmine
speeds from a given vector
    local wL = baseSpeed
    local wR = baseSpeed
    local x = vector[2]
    local y = vector[3]
    local mult = 5
    wR = wR + baseSpeed * -x * mult
    wL = wL + baseSpeed * x * mult

    if (y > 0.05) then
        wR = wR + wR * y * 0.5
        wL = wL + wL * y * 0.5
    else
        wR = 0
        wL = 0
    end
    return wL, wR
end

function followOther() -- method to determine speeds for friend following
    local wL, wR
    minDist = 0.5
    local distace, target = getDistanceToTarget(beaconSensor, beacon)
    wL, wR = getSpeedsBasedOnVector(maxSpeed, target)
    local targetIsInRange = false
    if (distace > 0) then
```

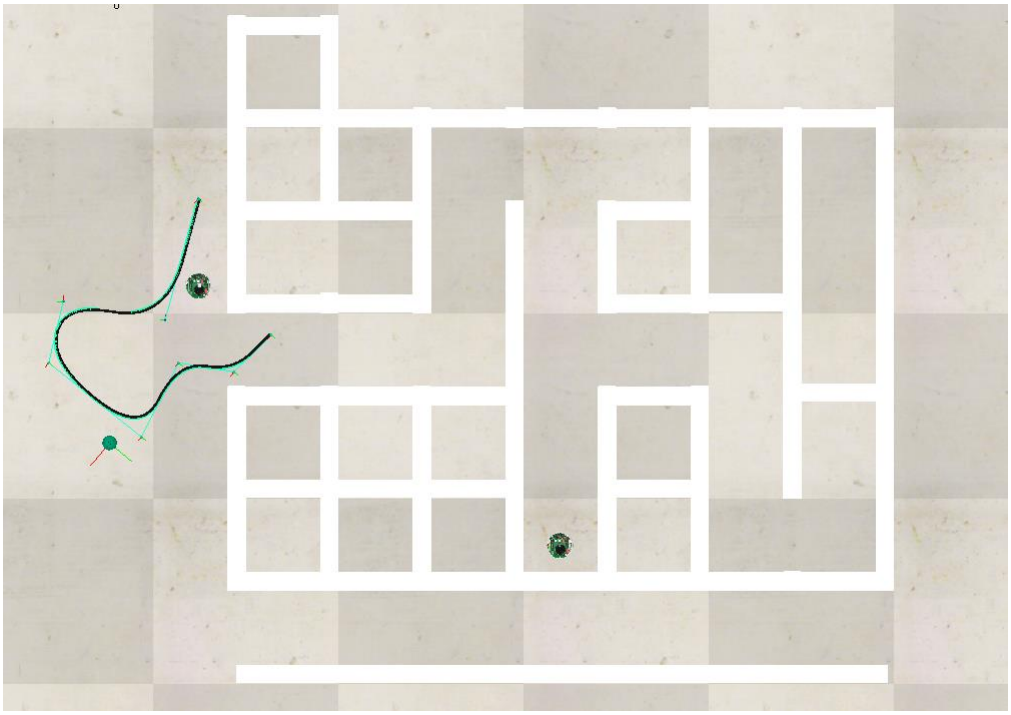
```
        targetIsInRange = true
    end
    return wL, wR, targetIsInRange
end

function getDistance(sensor) --helper method to get ditance from sensor
    local d, dist
    d, dist = sim.readProximitySensor(sensor)
    if (d < 1) then
        dist = distanceToWall
    end
    return dist
end

function getDistanceToTarget(sensor, target) -- helper function to determine
distance to target
    local detected, dist, point = sim.checkProximitySensor(sensor, target)
    if (detected < 1) then
        return -1, {0, 0, 0}
    end
    return dist, point
end

function getSpeed(handle) -- wrapper for API call
    local speed = sim.getJointTargetVelocity(handle)
    return speed
end

function followWall() -- fucntion to determine speed for wall following
    local desiredDistance = 0.06
    local phi, d, alpha
    local wL = getSpeed(leftMotor)
    local wR = getSpeed(rightMotor)
    local dFR = getDistance(fRightSensor)
    local dBR = getDistance(bRightSensor)

    local diff = math.abs(dFR - dBR)
```

```
    local mult = 10 + diff * 10
    local frontVal = (dFR - desiredDistance) * mult
    local backVal = (dBR - desiredDistance) * mult

    if (direction == 1) then
        if (frontVal > 0) then
            wL = minSpeed + frontVal * maxSpeed
            wR = minSpeed - frontVal * maxSpeed
        elseif (frontVal < 0) then
            wR = minSpeed - frontVal * maxSpeed
            wL = minSpeed + frontVal * maxSpeed
        end
    end
    return wL, wR
end
-- helper functions to set speeds
function getTurnSpeeds()
    local wL, wR, t
    if (direction > 0) then
        wL = -maxSpeed
        wR = maxSpeed
    else
        wL = maxSpeed
        wR = -maxSpeed
    end
    t = 0.8
    return wL, wR, t
end

function getCurveSpeeds()
    local wL, wR, t
    if (direction > 0) then
        wL = maxSpeed * 1
        wR = maxSpeed * 0.6
    else
        wL = maxSpeed * 0.2
```

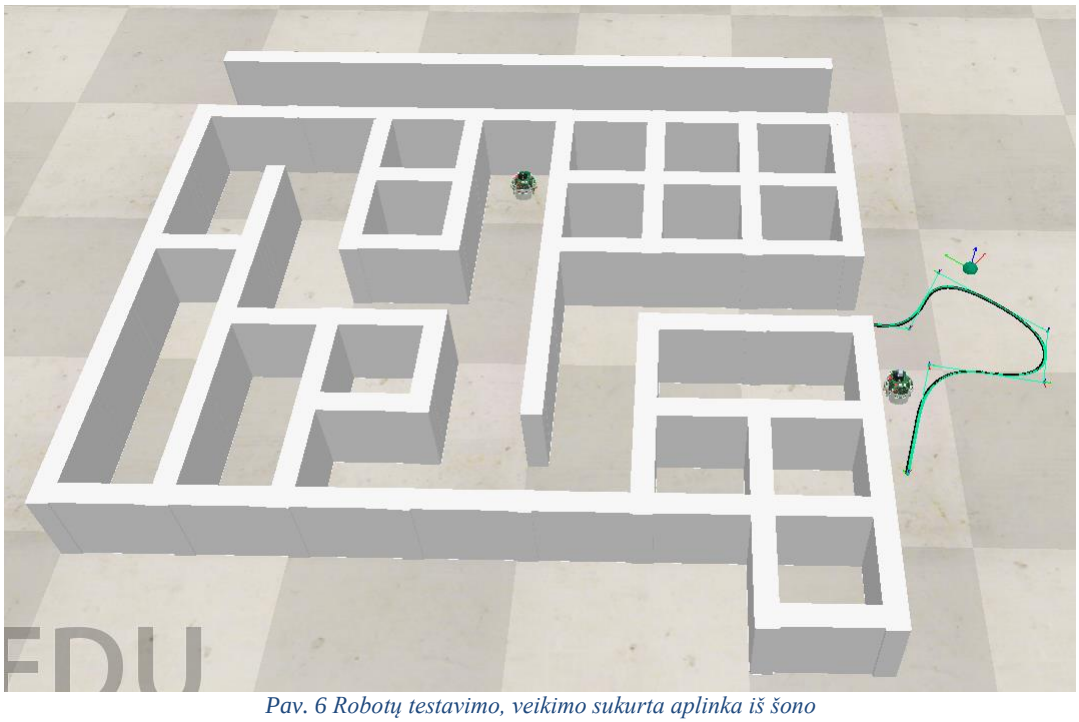
```
        wR = maxSpeed
    end
    t = 3.1
    return wL, wR, t
end

function coroutineMain()
    while true do
        sim.switchThread() -- Don't waste too much time in here (simulation time
will anyway only change in next thread switch)
    end
end

function sysCall_cleanup()
    -- Put some clean-up code here:
end
```

6. Roboto modeliavimo arba tyrimo rezultatai ir jų aprašymas

Siekiant ištestuoti ir patikrinti robotų veikimo modelius CoppeliaSim simuliacinėje aplinkoje sukūrėme, sudarėme iš elementų labirinto modelį. Pagrindinio veikimo roboto pradžios veikimo tašką parinkome labirinte. Robotą draugo pradžios veikimo tašką parinkome už labirinto taip siekiant kuo sklandžiau pereiti visus pagrindinio roboto rėžimus. Robotų greičiai skiriasi (robotas draugas yra lėtesnis nei pagrindinis robotas) siekiant, kad simuliacinėje aplinkoje pagrindinis robotas tikrai pasivytų judantį draugą ir galėtų atlikti draugo sekimo misiją. Robotas šiame sudarytame labirinte seks dešinę sieną ieškodamas kelio iš labirinto. Labirinto pabaigoje buvo sudaryta juoda vingiuota linija, kurią pagavęs robotas seka iki tol kol pames. Robotui atlikus linijos sekimą jis toliau seks pagavęs sieną ir vysis draugą robotą. Aptikęs draugą robotą pereis į draugo sekimo rėžimą, sulėtės ir atkartos visu draugo judesius iki tol kol pames draugą iš sensoriaus.



Pav. 5 Robotų testavimo, veikimo sukurta aplinka iš viršaus



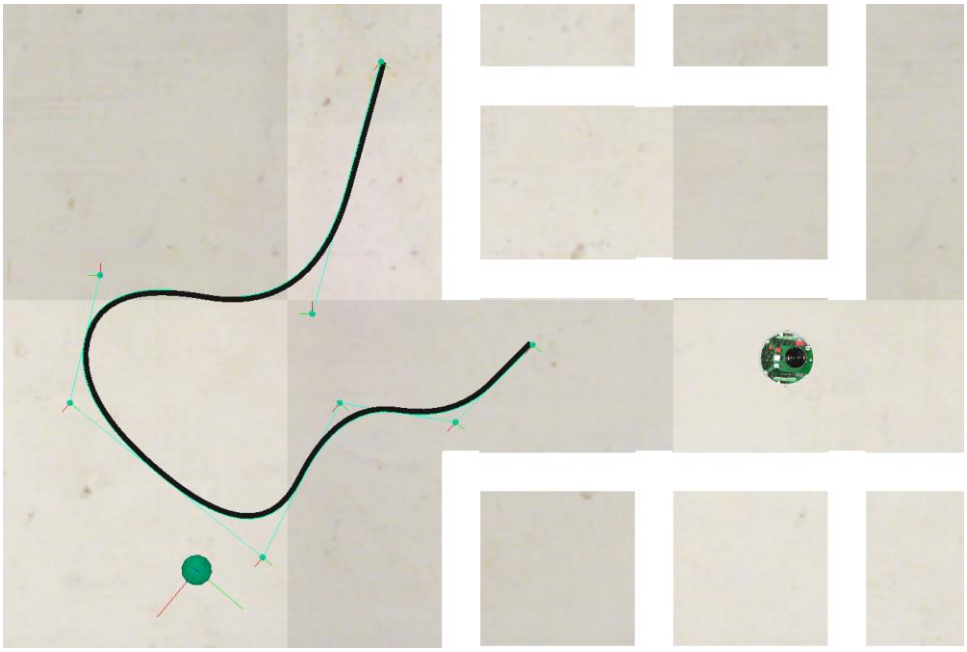
Pav. 6 Robotų testavimo, veikimo sukurta aplinka iš šono

Paleidžiama simuliacija. Simuliacijos pagrindinio demonstracinio įrašo pavadinimas ***„demontracinis.mp4“***. Įvykdžius simuliaciją matome, kad labirinto trasa buvo įveikta per 3 minutes. Tiek sienos sekimas, tiek linijos sekimas bei roboto draugo pasivijimas ir sekimas yra įvykdomas tvarkingai. Robotas įvykdo visas iškeltas misijas. Siekiant dar geriau parodyti roboto draugo sekimo veikimą paleidžiama antra simuliacija, demonstracinio įrašo pavadinimas ***„demo_sekimo.mkv“***, kurioje rankiniu būdų robotas draugas palaukia pagrindinio roboto, kuri įveikia labirintą, ir kuomet jį pradeda sekti, jis yra tempiamas demonstruojant kaip pagrindinis robotas seka.

7. Roboto valdymo eksperimentiniai testavimai

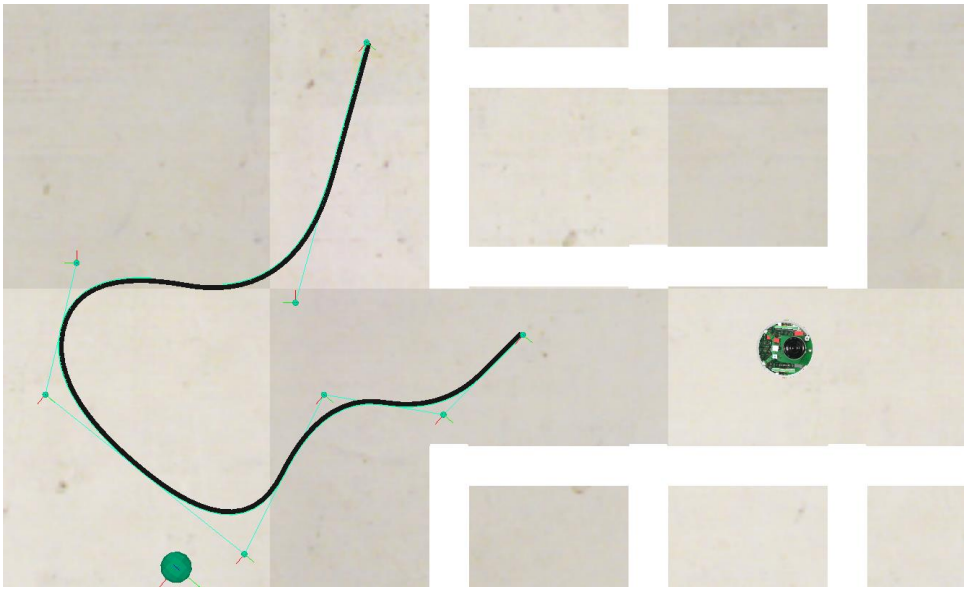
Pirmąjį eksperimentą, su pagrindinę sukurtą demonstracine aplinka, labirintų ir elementų išdėliojimu, atliksime linijos sekimo eigą keičiant sekamos linijos plotį ir stebima kada roboto sensorius pradės sunkiau gaudyti liniją.

Pagrindinėje demonstracinėje versijoje, įrašo pavadinimas ***„demontracinis.mp4“***, linijos storis yra 0.007. Detalesniame testavimo įrašę ***„linij_0.007.mp4“***, stebimas tik šios linijos sekimas. Šiame testavimo įraše matome, kad liniją robotas seka nuo pat pradžių iki galo niekad jos nepamesdamas. Liniją su minimaliu privažiavimu iki jos įveikia per maždaug 51 sekundę.



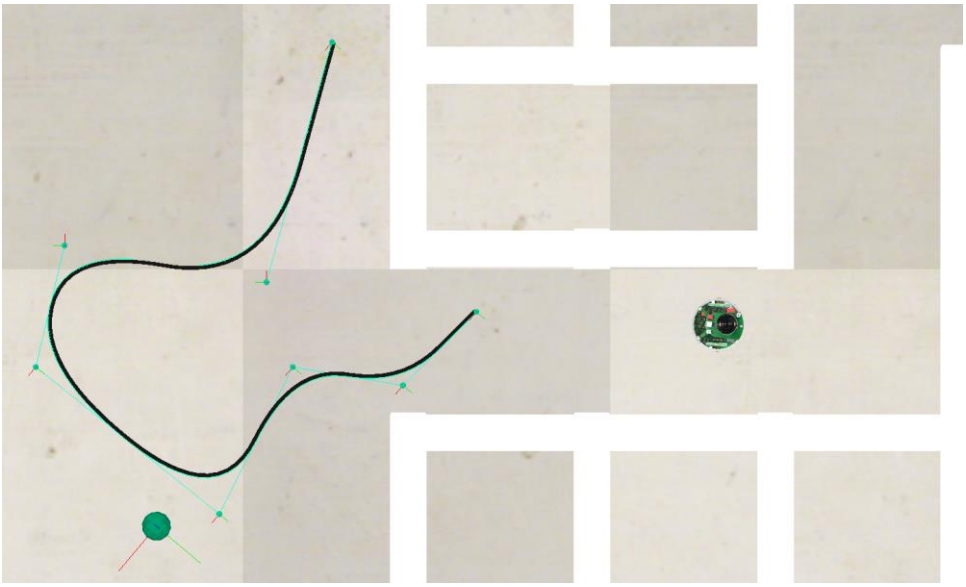
Pav. 7 Roboto linijos testavimas, linijos plotis 0.007

Sekančiu testavimo etapu linijos ploti sumažiname 0.001 reikšme. Linijos plotis nustatomas 0.006. Demonstraciniame įrašę, pavadinimas ***„linija_0.006.mp4“***, stebima kaip šio pločio liniją įveiks robotas. Šiame testavimo įraše matome, kad liniją robotas taip pat seka nuo pat pradžių iki galo niekad jos nepamesdamas. Liniją su minimaliu privažiavimu iki jos įveikia per maždaug 51 sekundę.



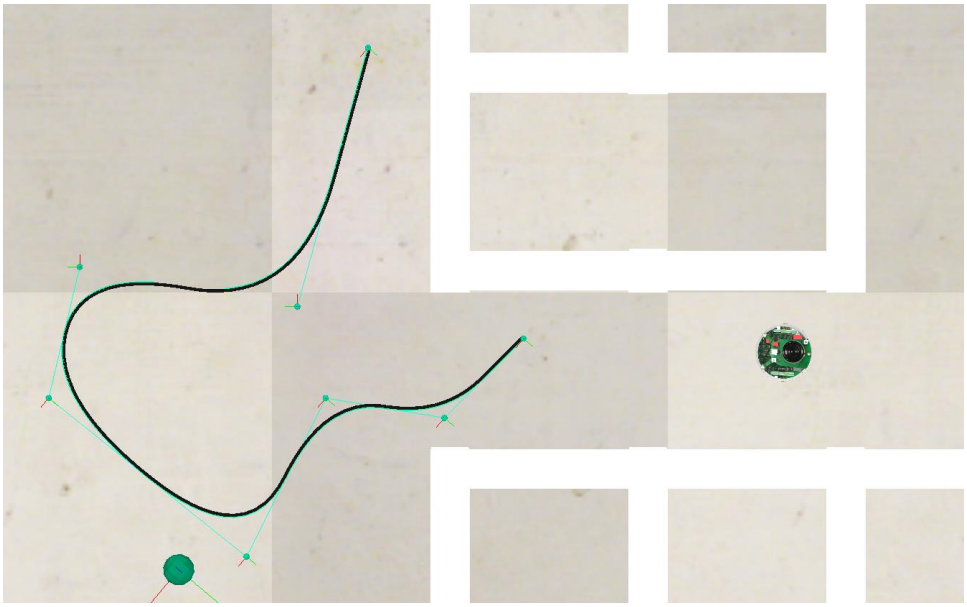
Pav. 8 Roboto linijos testavimas, linijos plotis 0.006

Kitame testavimo etapu linijos ploti dar sumažiname 0.001 reikšme. Linijos plotis nustatomas 0.005. Demonstraciniame įrašę, pavadinimas ***„linija_0.005.mp4“***, stebima kaip šio pločio liniją įveiks robotas. Šiame testavimo įraše matome, kad liniją robotas taip pat seka nuo pat pradžių iki galo niekad jos nepamesdamas. Liniją su minimaliu privažiavimu iki jos įveikia per maždaug 51 sekundę.



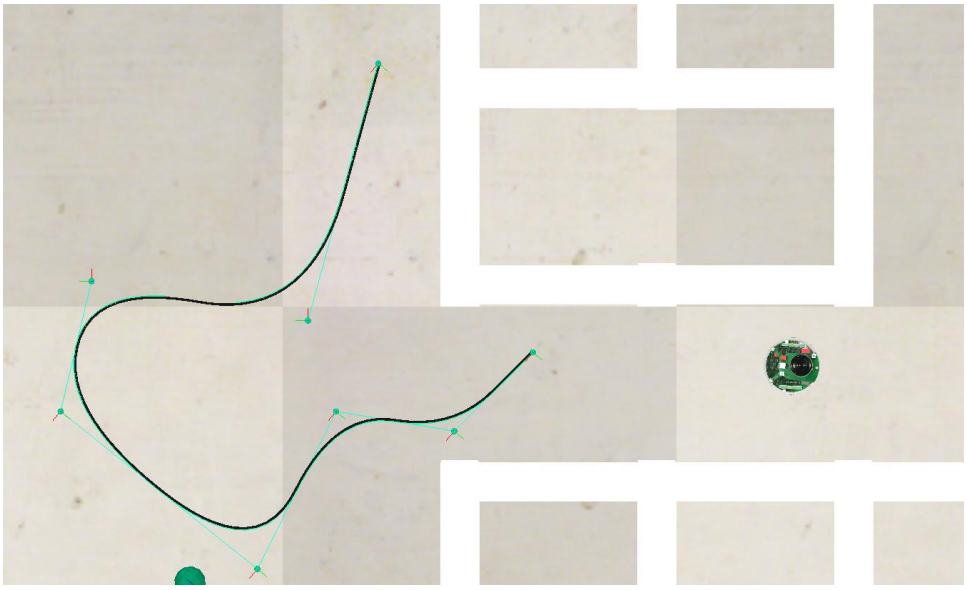
Pav. 9 Roboto linijos testavimas, linijos plotis 0.005

Sekančiu testavimo etapu linijos ploti dar sumažiname 0.001 reikšme. Linijos plotis nustatomas 0.004. Demonstraciniame įrašę, pavadinimas ***„linija_0.004.mp4“***, stebima kaip šio pločio liniją įveiks robotas. Šiame testavimo įraše matome, kad liniją robotas taip pat seka nuo pat pradžių iki galo niekad jos nepamesdamas. Liniją su minimaliu privažiavimu iki jos įveikia per maždaug 51 sekundę.



Pav. 10 Roboto linijos testavimas, linijos plotis 0.004

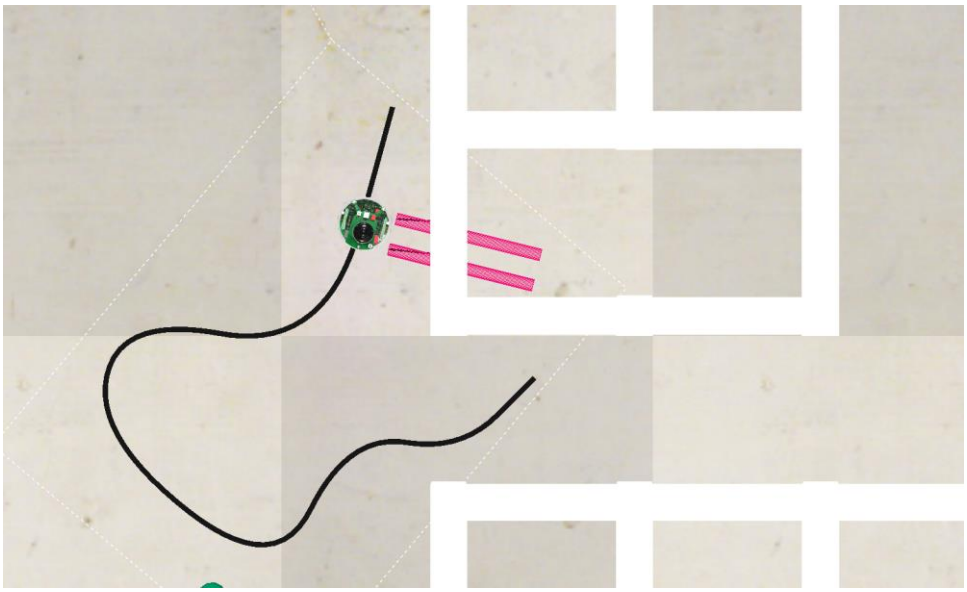
Kitame testavimo etapu linijos ploti dar sumažiname 0.001 reikšme. Linijos plotis nustatomas 0.003. Demonstraciniame įrašę, pavadinimas ***„linija_0.003.mp4“***, stebima kaip šio pločio liniją įveiks robotas. Šiame testavimo įraše matome, kad liniją robotas ties antra linijos dalimi liniją pameta. Galima daryti išvadą, kad linijos lygios 0.003 ar mažesnio pločiau nėra optimalios naudoti roboto linijos sekime.



Pav. 10 Roboto linijos testavimas, linijos plotis 0.003

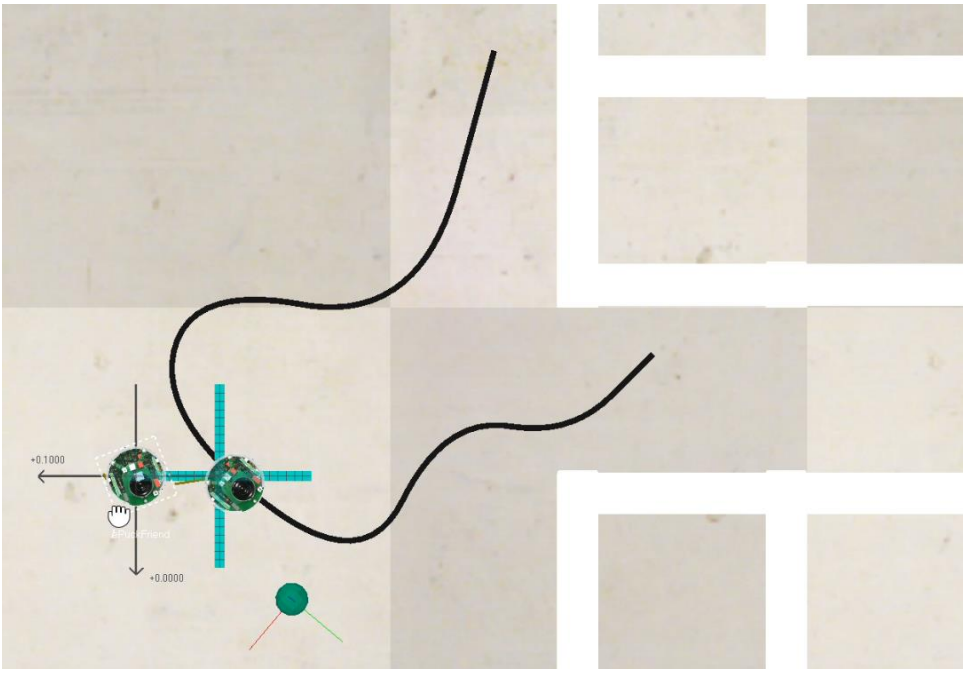
Antrąjį eksperimentą, su pagrindinę sukurtą demonstracine aplinka, labirintų ir elementų išdėliojimu, atliksime stebėti ar tinkamai veikia roboto veiksmų prioritetai. Svarbiausio prioriteto yra linijos sekimo veiksmas, tuomet draugo sekimo ir galiausiai sienos sekimas.

Pirmame demonstraciniame įraše ***„sekimas_01.mp4“***, testuojame ar robotas sekdamas linija ir aptikęs šalia esančią sieną nepameta linijos ir nepradeda sekti sienos. Paleidę simuliaciją matome, kad net robotui aptikus sieną sensoriai robotas nenustoja sekti linijos iki pat jos galo ir tik pasiekus linijos pabaigą pradeda reaguoti į sienos aptikimą.



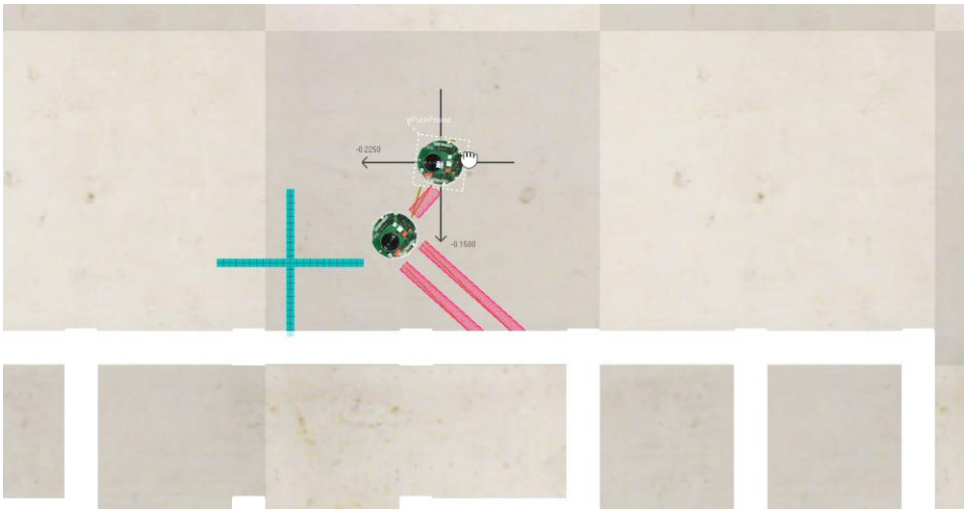
Pav. 11 Roboto linijos sekimo išlaikymo testavimas, aptinkant sieną

Antrame demonstraciniame įraše ***„sekimas_02.mp4“***, testuojame ar robotas sekdamas linija ir sutikęs važiuojanti draugą robotą nepameta linijos ir nepradeda jo sekti. Paleidę simuliaciją matome, kad net tuo atveju kaip roboto kelyje pasitaiko ir yra aptinkamas draugas robotas jis nepameta linijos ir toliau seka iki linijos pabaigos. Į draugą robotą sureaguoja tik įvykdžius roboto sekimą.



Pav. 12 Roboto linijos sekimo išlaikymo testavimas, aptinkant draugą

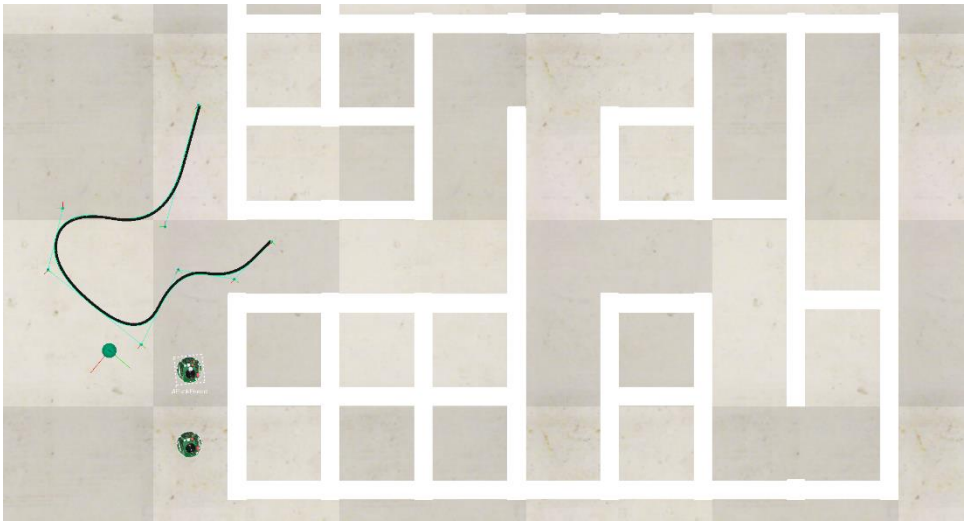
Trečiame demonstraciniame įraše ***„sekimas_03.mp4“***, testuojame ar robotas aptikęs roboto draugą nustoja sekti sieną ir seka tik draugą robotą. Paleidę simuliaciją matome, kad vos aptikus robotą draugą jis yra sekamas nepaisant ar siena šalia yra ar ne. Tik draugą pametus robotas grįžta į sienos sekimą.



Pav. 13 Roboto draugo sekimo testavimas, aptinkant sieną

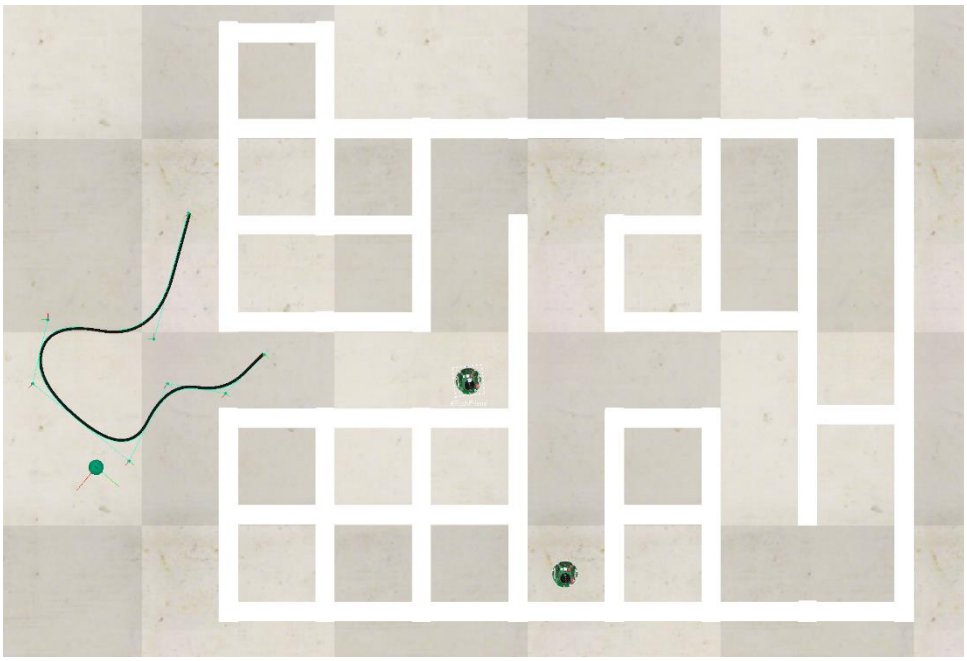
Trečiajame eksperimente pagrindinėje sukurtoje demonstracinėje aplinkoje keitėme robotų išdėliojimo pradines pozicijas. Stebima kokia trajektorija robotai juda ir kai reaguoja į susidariusias aplinkybes.

Pirmuoju bandymu demonstraciniame įraše ***„pozicija_01.mp4“***, abu robotus pastatome už labirinto judėjimo link labirinto išėjimo kryptimi. Paleidus simuliaciją matome, kad pagrindinis robotas pasiviję draugą robotą, pradeda jį sketi. Pasiekus labirinto išėjimo dalį, abu robotai įsuka į labirintą ir toliau seka dešinę sieną.



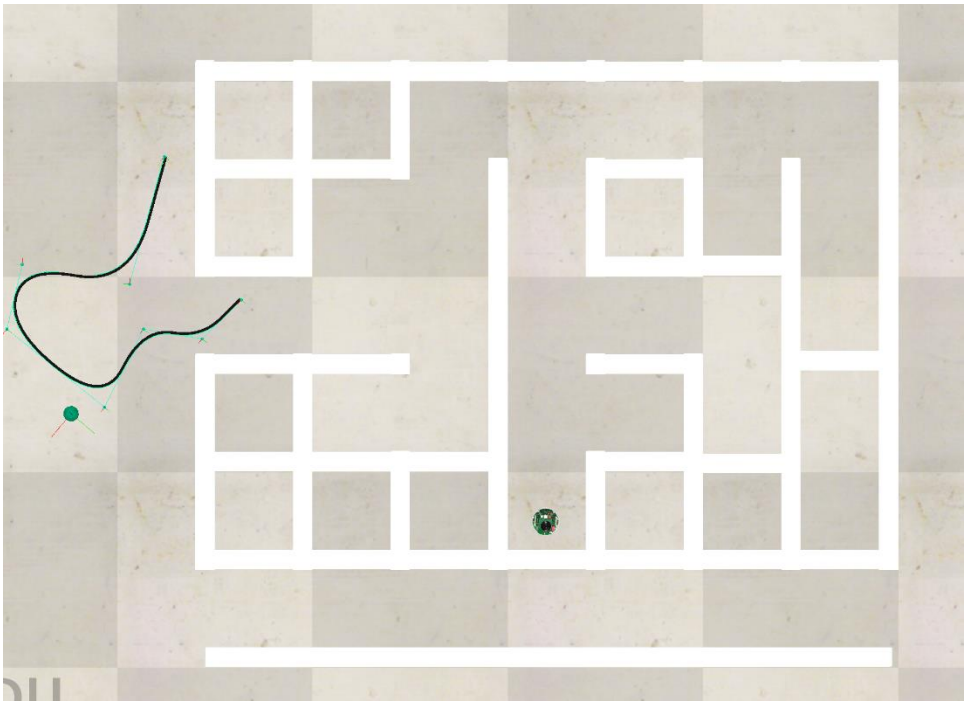
Pav. 14 Robotai pradeda judėti iš panašių tašku

Pirmuoju bandymu demonstraciniame įraše ***„pozicija_02.mp4“***, pagrindinį robotą paliekame pagrindinėje demonstracinėje pozicijoje, o draugą robotą perstatome ne už labirinto išėjimo o prieš. Paleidus simuliaciją stebime, kaip robotai reaguos susidūrę vienas su priešprieša. Matome, kad susidūrus draugas robotas apsisuka pagrindinio roboto judėjimo kryptin ir abu robotai pradeda judėti kartu.



Pav. 15 Robotai pradeda judėti vienas priešais kitą

Ketvirtajame eksperimente pakeičiame labirinto išdėliojimą ir stebime kaip robotas įveikia naująjį labirintą ir kiek laiko užtrunka, ***„labirintas_2.mp4“***. Pagrindinį demonstracinį labirintą (Pav. 5) robotas įveikė per 3 minutes. Pirmąjį naujai sudaryta labirintą (Pav. 16) robotas įveikia maždaug per 3 minutes 46 sekundes, apie 46 sekundes lėčiau nei demonstracinį.



Pav. 16 Robotų testavimui, veikimui sukurtas nauja lairintas iš viršaus

8. Išvados

Gabrielė:

Susipažinome su CoppeliaSim simuliacine aplinka, jos paleidimu, elementų kūrimu. Atlikus testavimo eksperimentus, galime daryti išvadą, kad robotas seka liniją aukščiausiu prioritetu. Robotas seka aptiktą robotą draugą net ir pagavus šalia esančia sieną sensoriais. Atlikus sekimo linijos pločio testavimus, darome išvada, kad kuomet linijos plotis yra lygus 0.003 arba mažesnis, roboto šviesos sensorius linija aptinka sunkiau ir gali ją pamesti. Antrąjį sudarytą labirintą robotas įveikia lėčiau nei pagrindinį demonstracinį labirintą. Vertinant visus atliktus testavimus galime daryti išvadą, kad robotas įgyvendina visas iškeltas misijas.

Prisidėjau prie:

- Ataskaitos
- Eksperimentinių testavimų
- Skaidrių
- Programavimo

Audrius:

Susipažinta su coppelia sim simuliacine aplinka bei sukurtas roboto modelis įvykdantis užduotį. Mano manymu darbas mums pavyko - robotas įgyvendina užsibrėžtas funkcijas, tiesa roboto sekimas nėra įvykdytas tikslai pagal užduotį. Nepavyko rasti protingo būdo simuliacijoje robotams bendrauti per IR sensorius todėl buvo pasirinktas apėjimas- naudojama simuliacijos API funkcija nusakanti sekamo roboto vieta ir atstumą.

Prisidėjau prie:

- Ataskaitos
- Skaidrių
- Eksperimentinių testavimų
- Programavimo

9. Literatūra

- CompeliaSim User Manual, [žiūrėta 2021–12]. Prieiga per internetą: https://www.coppeliarobotics.com/helpFiles/index.html
- How to Use Proximity Sensors To Stay Between Walls, (2020-02-12) [žiūrėta 2021–12]. Prieiga per internetą: https://www.youtube.com/watch?v=hqZ3YRW16KA#fromHistory&ab_channel=LeopoldoArmesto
- How To Solve a Maze with a Wall Follower Algorithm, (2020-03-03). Prieiga per internetą:

https://www.youtube.com/watch?v=DkOXpZtEZwg&t=589s&ab_channel=LeopoldoArmesto
- CompeliaSim FORUM, [žiūrėta 2021–12]. Prieiga per internetą: https://forum.coppeliarobotics.com/viewtopic.php?t=2355