МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ государственное БЮДЖЕТНОЕ

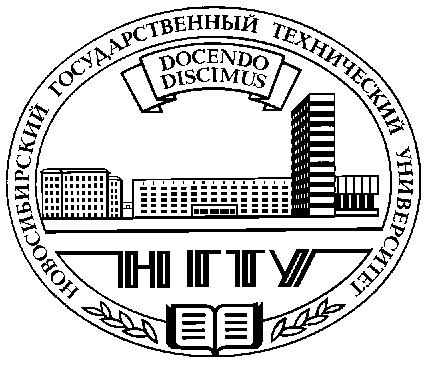
образовательное учреждение

высшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра вычислительной техники



**ОТЧЁТ**

**по лабораторной работе №1-2**

**«Управление движением робота по разметке»**

**по дисциплине: «***Автономные роботы и многоагентные системы***»**

Выполнил: Преподаватель:

Студент гр. АММ2-21, АВТФ Яковина И. Н.

Назьмов А. А.

Новосибирск

2022

**Задание****:**

1. Познакомиться с основными понятиями симулятора CoppeliaSim: сцена, примитивы, соединения, типы объектов.
2. Загрузить модели робота и полигона.
3. Освоить базовые приемы при работе на сцене в симуляторе CoppeliaSim.
4. Познакомиться с приемами и способами управления движением модели мобильного робота с двумя моторами и двумя визуальными датчиками
5. Получить значения средней яркости по кадру с визуальных датчиков. Результаты занести в таблицу.
6. Создать собственный полигон.
7. Составить алгоритм управления движением мобильного робота на основе:
   1. переключателя
   2. П-регулятора
   3. ПД-регулятора
8. Выполнить тестирование разработанных алгоритмов на сцене и занести результаты проведенных экспериментов в таблицу.

**Выполнение работы:**

Двигательная платформа робота (тележка) — трехопорная и включает левый и правый независимые двигатели. Благодаря такой схеме возможен разворот робота на месте, что подтверждает высокую маневренность робота, и соответственно, возможность проходить крутые трассы с острыми углами поворота достаточно стабильно. Визуальные датчики имеют разрешение 8х8, но используется только среднее показание яркости с сенсора. Точнее, используется разность левого и правого сенсоров, благодаря чему число управляющих сигналов сводится к одному, а алгоритм управления движением по разметке — к передаточной функции от разности показаний датчиков к разности скоростей двух двигателей, которая отдается на вход простой функции руления.

В симуляторе сигналы с визуальных датчиков изменяются в диапазоне 0-1, соответственно, управляющий сигнал — их разность — от -1 до 1. Примеры показаний датчиков, полученных в симуляторе, можно найти в таблице 1.

Из различных загруженных элементов собран полигон: кольцевая трасса из белой разметки на черном фоне; трасса изображена на рис. 1.

Для управления роботом и прохождения им трассы с наибольшей скоростью и стабильностью реализованы и настроены (подобраны коэффициенты) несколько алгоритмов. Результаты экспериментов с алгоритмами управления приведены в таблице 2.

Простейший алгоритм — двухпозиционный переключатель — очевидно проигрывает остальным по всем параметрам. Во-первых, это связано с увеличением проходимого роботом пути в 2-3 раза, так как он совершает рыскающие движения и проезжает по зигзагу. Во-вторых, разность в модулях скоростей левого и правого двигателя должна быть большой для возможности разворота практически на месте при прохождении элемента с резким поворотом (более 90 градусов), что делает этот самый зигзаг очень плотным. Кроме того, резкая смена направления вращения колес приводит к проскальзыванию (а значит, увеличению времени прохождения) и опрокидыванию тележки, что произошло, например, в экспериментах при 40 и 60% мощности от максимальной — поэтому трасса помечена непройденной совсем. Добавление третьего режима — движения по прямой, сильно смягчает перечисленные недостатки и позволяет пройти трассу средней сложности.

Алгоритмы П-регулятора и ПД-регулятора имеют идентичные показатели. Предполагалось, что более сложный из них должен дать небольшой прирост относительно простого в скорости и стабильности, исключив рывки, резкие изменения скорости вращения двигателей; но и для самого П-регулятора на практике не наблюдается подобных рывков. Вероятно, такое сглаживание ПД — регулятора на резких участках трассы компенсируется более долгим разгоном на прямых участках. То есть более попробовать добиться более высоких результатов ПД-регулятора, чем у П-регулятора, можно на более извилистых трассах без прямых участков.

**Результаты:**



Рисунок 1. Вид поля с кольцевой трассой и начальное положение робота

Таблица 1. Численная оценка показаний визуальных датчиков

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ЛЕВЫЙ** | | **Значение ОШИБКИ**  **(Лев - Прав)** | **ПРАВЫЙ** | |
| **Внешний вид показаний визуального датчика** | **Численное значение с датчика** | **Численное значение с датчика** | **Внешний вид показаний визуального датчика** |
| Полностью БЕЛЫЙ кадр | 1.0 | 0.0 | 1.0 | Полностью БЕЛЫЙ кадр |
| Полностью БЕЛЫЙ кадр | 1.0 | 0.95 | 0.05 | Полностью ЧЕРНЫЙ кадр |
| Полностью ЧЕРНЫЙ кадр | 0.05 | -0.95 | 1.0 | Полностью БЕЛЫЙ кадр |
| Полностью ЧЕРНЫЙ кадр | 0.05 | 0.0 | 0.05 | Полностью ЧЕРНЫЙ кадр |
|  | 0.4 | -0.22 | 0.62 |  |
|  | 0.67 | 0.15 | 0.52 |  |

Таблица 2. Результаты экспериментов

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Алгоритм** | **Значения коэфф.** | **Мощн. моторов.**  **(при макс. 8 рад/с)** | **Стабильность прохождения участка пути** | | |
| **Факт прохождения полного круга** | **Внутримодельное время прохождения круга, сек** | **Название элемента на котором возникает срыв с линии** |
| Переключатель 2-позиционный | *Kl = 1/1.4*  *bound = 0.15* | 20% | *нет* | *1020* | *pb\_l3* |
| 40% | *нет* | *-* | *pb\_c1* |
| 60% | *нет* | *-* | *pb\_c6* |
| Переключатель 3-позиционный | *Kl = 1.2*  *bound = 0.15* | 20% | *да* | *132* | *-* |
| 40% | *да* | *68* | *-* |
| 60% | *да* | *55* | *-* |
| П - регулятор | *Kl = 2.2*  *Kp = 3.2* | 20% | *нет* | *-* | *pb\_z4* |
| 40% | *да* | *65* | *-* |
| 60% | *да* | *44* | *-* |
| ПД -  регулятор | *Kl = 2.6*  *Kp = 3.6*  *Kd = 0.33* | 20% | *нет* | *-* | *pb\_z4* |
| 40% | *да* | *65* | *-* |
| 60% | *да* | *44* | *-* |

Полный скрипт управления объектом тележки:

#python

algorithms = (

"switch2",

"switch3",

"Pcontroller",

"PDcontroller",

"guk\_original",

"nonee"

)

# --------------------chose here------------------

algorithm = algorithms[5]

#

max\_velocity = 8.

norm\_velocity = max\_velocity \* 0.2

# ------------------------------------------------

object\_handlers = {

"sensors": {

"L": None,

"R": None

},

"engines": {

"L": None,

"R": None

}

}

def apply\_velocity(vel):

for side in ("R", "L"):

sim.setJointTargetVelocity(

object\_handlers["engines"][side],

vel[side])

return

#

velocity = {"L": 0., "R": max\_velocity}

velocity\_prev = {"L": 0., "R": max\_velocity}

def nonee(impact):

return

def switch2(impact):

insignificance\_bound = 0.15

vel\_high = norm\_velocity

vel\_low = - vel\_high / 1.4

if abs(impact) > insignificance\_bound:

velocity["R"] = vel\_low if impact > 0. else vel\_high

velocity["L"] = vel\_low if impact < 0. else vel\_high

apply\_velocity(velocity)

return

def switch3(impact):

insignificance\_bound = 0.15

vel\_high = norm\_velocity

vel\_low = - vel\_high \* 1.2

velocity["R"] = velocity["L"] = vel\_high

if abs(impact) > insignificance\_bound:

velocity["R" if impact > 0. else "L"] = vel\_low

apply\_velocity(velocity)

return

def guk\_original(impact):

vel\_high = norm\_velocity

vel\_low = - vel\_high / 1.2

velocity["R"] = velocity["L"] = vel\_high

velocity["R" if impact > 0. else "L"] = vel\_low \* abs(impact)

apply\_velocity(velocity)

return

def Pcontroller(impact):

vel\_high = norm\_velocity

vel\_low = - vel\_high \* 2.2

velocity["R"] = velocity["L"] = vel\_high

vel\_low = vel\_high - (vel\_high - vel\_low) \* abs(impact)

velocity["R" if impact > 0. else "L"] = vel\_low

apply\_velocity(velocity)

return

def sign(val):

return -1 if val < 0 else 1

def PDcontroller(impact):

vel\_high = norm\_velocity

vel\_low = - vel\_high \* 2.6

velocity["R"] = velocity["L"] = vel\_high

vel\_low = vel\_high - (vel\_high - vel\_low) \* abs(impact)

velocity["R" if impact > 0. else "L"] = vel\_low

Kd = 0.33

for key in velocity.keys():

velocity[key] += (velocity\_prev[key] - velocity[key])\* Kd

velocity\_prev[key] = velocity[key]

apply\_velocity(velocity)

return

# ----------------------------------------------------

def sysCall\_init():

for path, key in zip(("revj\_W","vis\_sensor\_"),("engines","sensors")):

for side in ("L","R"):

tmp = sim.getObject("./" + path + side)

if tmp is None:

print(path + side + " not found!!!")

object\_handlers[key][side] = tmp

return

def sysCall\_actuation():

sensing = {"L" : None,"R" : None}

for side in sensing.keys():

res = sim.readVisionSensor(object\_handlers["sensors"][side])

if type(res) is int:

return # sometimes there is no info except single number

sensing[side] = res[1][11]

move\_control = eval(algorithm)

move\_control(sensing["R"] - sensing["L"])

return

def sysCall\_sensing():

return

def sysCall\_cleanup():

return