МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Факультет информационных систем и технологий

Кафедра «Вычислительная техника»

Дисциплина «Программирование робототехнических комплексов»

**Лабораторная работа №2**

«Построение карт с использованием SLAM»

Вариант 12

Выполнил:

студент гр. ИВТВМбд-31

Кондратьев П.С.

Проверил:

Святов К.В.

Ульяновск, 2019

**Техническое задание**

1. В среде V-REP необходимо с реализовать построение карты окружающего робота пространства с использованием библиотеки любой библиотеки SLAM (например, BreezySLAM) со сценой, разработанной в лаборатрной работе №1 в соответствии с вариантом.

Задание было обговорено с преподавателем и были внесены изменения.

Вместо библиотеки BreezySLAM, была взята библиотека под windows, построение карты окружающего робота пространства.

Задача построения сводилась к получению данных с лидара, подсчета Одометрии и передача полученных данных в отрисовке карты движения робота.

**Данные с лидара:**

Был переписан исходный код сенсора fastHokuyo, для того чтобы получать две отдельные переменные (lidarMeasuredData, distances), это было сделано для удобства получения данных с лидара, т.к. первоначально они собрались в одну закодированную строку, которую нужно было разбивать по разделителю и декодировать.

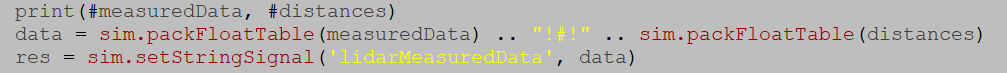


Рис.1. Исходный код сенсора fastHokuyo

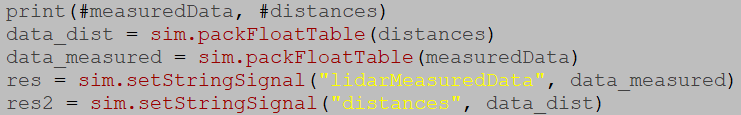


Рис.2. Измененный код сенсора fastHokuyo

Затем в коде можно с легкостью, без каких-либо разделителей получать данные слидара, после чего переведя их в читабельный вид, использую API Vrep, метод simxUnpackFloats и хранение их в массивах.

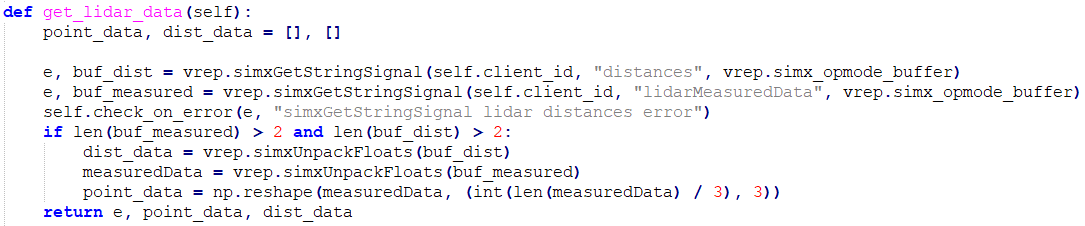
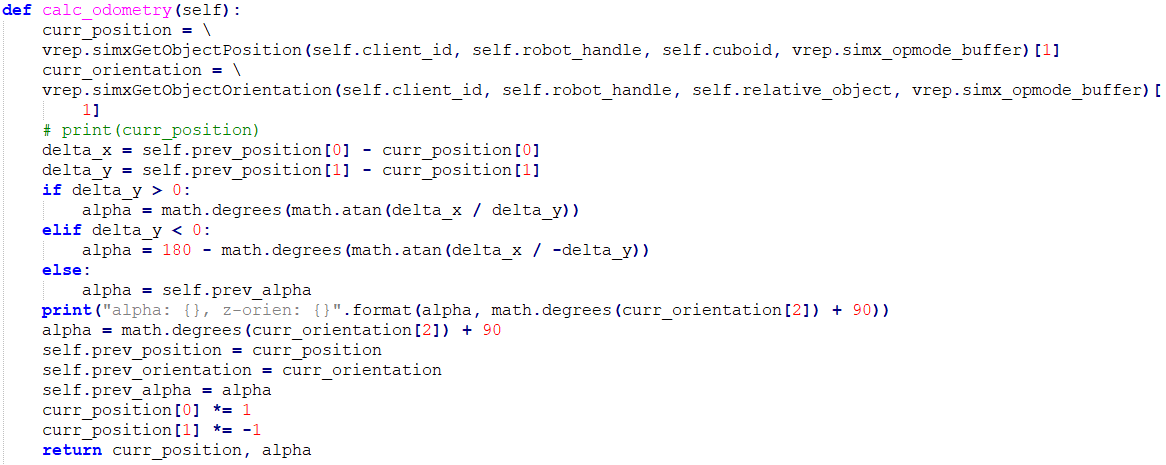


Рис.3. Метод получения данных с лидара

Затем по полученным данным подсчитываем Одометрия, для отслеживания данных о движении приводов для оценки перемещения и отслеживания положения робота.



Затем все полученные данные отправляем в метод отрисовки карты. Эти методы описаны в классе под названием MAP, там задаются начальные параметры экрана цвет экрана, цвет лучей лидара. Существует 3 метода:

1. Update - Метод обновляет карту
2. show\_map - Показывает карту. При нажатии на кнопку "s" сохраняет карту в директорию скрипта.
3. draw\_lidar\_data - Производит отрисовку данных в виде лучей, в случае если длина луча меньше максимальной (<95) значит, он врезается в стену и нужно ее сделать немного жирнее, чтобы точно детектилась.

Исходный код библиотеки смотреть в Приложении 1.

**Получившийся результат**



**Приложении 1**

**Код программы**

**class Map – для отрисовки карты**

import cv2

import math

import numpy as np

class Map(object):

def \_\_init\_\_(self, width, height, meters):

self.width = width

self.height = height

self.start\_pos\_x = width // 2

self.start\_pos\_y = height // 5

self.image = np.zeros((height, width, 3), np.uint8)

self.line\_color = (255, 255, 255)

self.robot\_color = (0, 255, 0)

self.ignore\_dist = 300

self.scale\_effect = 20

# Метод обновляет карту.

def update(self, pos, angle, lidar\_data):

pos = [p \* self.scale\_effect for p in pos]

lidar\_data = [dt \* self.scale\_effect for dt in lidar\_data]

pos[0] += self.start\_pos\_x

pos[1] += self.start\_pos\_y

pos = (int(pos[0]), int(pos[1]))

# drawing lidar lines

self.draw\_lidar\_data(pos, angle, lidar\_data)

# draw the vehicle as a circle

self.draw\_robot(pos)

# show map

self.show\_map()

# Показывает карту. При нажатии на кнопку "s" сохраняет карту в директорию скрипта.

def show\_map(self):

cv2.imshow('SLAM', self.image)

k = cv2.waitKey(1) & 0xFF

# saving the map if 's' is pressed

if k == ord('s'):

cv2.imwrite('map.png', self.image)

# Производит отрисовку данных в виде лучей, в случае если длина луча меньше максимальной (< 95) значит

# он врезается в стену и нужно ее сделать немного жирнее, чтобы точно детектилась

def draw\_lidar\_data(self, pos, ang, lidar\_data):

print("fir - {}, mid - {}, last - {}".format(lidar\_data[0], lidar\_data[684 // 2], lidar\_data[684 - 3]))

angle = -30 - ang

angle\_step = 240.0 / len(lidar\_data)

for dist in lidar\_data:

x = int(pos[0] + dist \* math.cos(math.radians(angle)))

y = int(pos[1] + dist \* math.sin(math.radians(angle)))

cv2.line(self.image, pos, (x, y), self.line\_color, 1)

if dist < 95:

cv2.circle(self.image, (x, y), 5, (0, 0, 0), cv2.FILLED)

angle += angle\_step