Вводная часть (концепт)

Платформа должна обеспечивать выполнение различных сценариев работы и отладки на ней некой группы искусственных интеллектуальных систем. В качестве потенциальных задач выступают: передвижение в помещении (лабиринте), на открытой местности, поддерживать работу с различными сенсорами очувствления, содержать базовый набор датчиков для автономного передвижения.

Рассмотрим подробнее требования, которые накладываются представленными задачами. Передвижение в помещении накладывает ограничение на габариты мобильной платформы и мощность для безопасности. Для разворота в узких пространствах внутри комнаты удобно использовать дифференциальный привод. Комфортная максимальная скорость для помещения составит 1 м/с. Пороги от открытых дверей высотой до 5 см задают минимальный диаметр колеса от 100 мм. Открытая местность, содержащая траву и ухабы, подразумевает большую площадь контакта движителя с поверхностью, а также минимальное отношение мощности привода к снаряженной массе для преодоления уклона в 30 градусов.

В качестве решения для заявленных ездовых характеристик можно предложить использовать шестиколесную конфигурацию с балочной подвеской и вентильным мотором-колесом для простоты конструкции. Для обеспечения модульности мобильной платформы стоит разбить на две отдельные части корпус с набором датчиков и движущий модуль, который будет отдельным компонентом системы управления, а также предусмотреть быстрый монтаж корпуса и его модификацию на движущий модуль. Корпус может состоять из этажей в виде пластин с набором монтажных отверстий, которые крепятся к единому туннелю, в котором проложены все шины данных и питание. Для питания датчиков необходимы разные уровни напряжений, которые должен обеспечивать движущий модуль соответственно. Для передвижения в темноте на движущий модуль должны быть установлены светодиодные фары, для удобства зарядки и работы с платформой длительное время во время отладки - блок питания AС/DC, набор стабилизаторов напряжения с разъёмами, аккумулятор, обеспечивающий работу на протяжении 4 часов, блок контроллера с драйверами и выделенным портом для связи. Движущий модуль должен иметь защиту от воды по стандарту IP20 и брызговики на колесах.

Для работы с различными датчиками необходимо иметь набор всевозможных интерфейсов: Ethernet, I2C, CAN, UART, SPI, USB и удобную среду для их коммуникации. Такой платформой может выступить одноплатный компьютер Raspberry pi под управлением ROS. Тогда для вычислений можно использовать внешний компьютер или мощный микрокомпьютер, а для стабильной связи между ними использовать роутер. Также необходимо предоставить место для их монтажа и питания. Базовый набор датчиков для автономного вождения состоит из сканирующего плоскостного лидара, четырех видеокамер с виброразвязкой, дисплея для удобства отладки, ультразвуковых дальномеров для предотвращения столкновений.

Таким образом, получили пример концепта такой платформы, показанные на рисунке 1 и рисунке 2.



Рисунок 1 – движущий модуль и мобильная платформа в сборе



Рисунок 2 – Пример конфигурации мобильной платформы

Выбор компонентов

Каркас для мобильной платформы уже имелся в наличии, габариты 200 мм на 350 мм с учетом возможности для роста в высоту достаточны для размещения всех планируемых компонентов. Мотор-редуктор со встроенным энкодером для работы одометрии для этой платформы был выбран IG-32GM с параметрами: крутящий момент 1.9 кг.см и скорость 150 об/мин, что позволит развивать платформе линейную скорость до 0.93 м/с2 с диаметром внедорожных колес 120 мм. В качестве микрокомпьютера для вычислений была выбрана доступная в продаже Nvidia Jetson TX1 с низким уровнем энергопотребления и 256 ядерным графическим процессором на архитектуре Pascal, что по производительности превосходит серверный Xeon в задачах глубокого обучения. Данный встраиваемый компьютер позволит производить обработку сложных данных на борту мобильного робота, что исключит необходимость в стабильном беспроводном подключении и позволит работать с платформой в среде с ограниченным доступом к сети. В качестве несущей платы для Jetson TX1 была выбрана J130-2k4k с компактными размерами 110 мм на 74 мм и необходимыми портами: USB3.0, CAN, UART, IMU (гироскоп, компас и акселерометр), а также встроенный преобразователь питания. Аккумулятор также имелся в наличии на 7.2 В и 5.2 А, что составляет емкость в 38 Вт.ч. Напряжение питания ниже того, на которое рассчитаны моторы, но это поспособствует меньшему износу обмоток мотора, что увеличит срок службы. Зная энергопотребление компонентов равное 13 Вт и потребление тока моторов на средней скорости при массе 5 кг равное 24 Вт, можно приблизительно оценить среднее время работы платформы – 65 мин.

Определили и остальные компоненты. Широкоугольные камеры OV5640 с углом обзора 175 градусов и разрешением 1080р в количестве четырех штук для возможности использования эффекта стереозрения подключены к Nvidia Jetson. Ультразвуковые дальномеры HC-SR04 со средним углом обзора 30 градусов для размещения по кругу будут выполнять роль низкого уровня защиты от столкновений. Они подключаются к отдельному модулю под управлением микроконтроллера Atmega328p, который обрабатывает информацию с них и передает по шине I2С ядру, которым выступает Raspberry pi 3a. Также к ядру подключен модуль управления движущий модуль, который связан с ним по отдельному UART каналу для сохранения передачи команд движения при ошибках на общих шинах. В качестве микроконтроллера модуля выбран Atmega328p, к аналоговым выходам которого подключены: датчик температуры AD7416ARZ, датчик тока ACS712, компаратор напряжения AD790JNZ. К ЦАП выводу подключен пъезодинамик для сообщения об ошибках. К пинам с внешним прерыванием подключены разъёмы для энкодеров. В качестве драйвера моторов выбраны транзисторы IRFR13N20, подключенные к пинам с ШИМ. В качестве стабилизатора напряжения модуля и питания других компонентов по 5 В линии выбран LM2678T с защитой. К ядру также подключен по USB лидар Hokuyo URG-04LX с дальностью работы до 5 м, что будет уместно для работы в помещении. Ядро соединено с Jetson TX1 и внешним компьютером по Wi-Fi под управлением ROS kinetic. К ядру подключен через GPIO-контакты разветвитель для шины I2С и SPI. Через него подключен десяти степенной модуль IMU - MPU9255, включающий в себя трех-осевой гироскоп, акселерометр и компас/магнетометр. Функциональная схема работы данных электронных компонентов показана на рисунке 3.

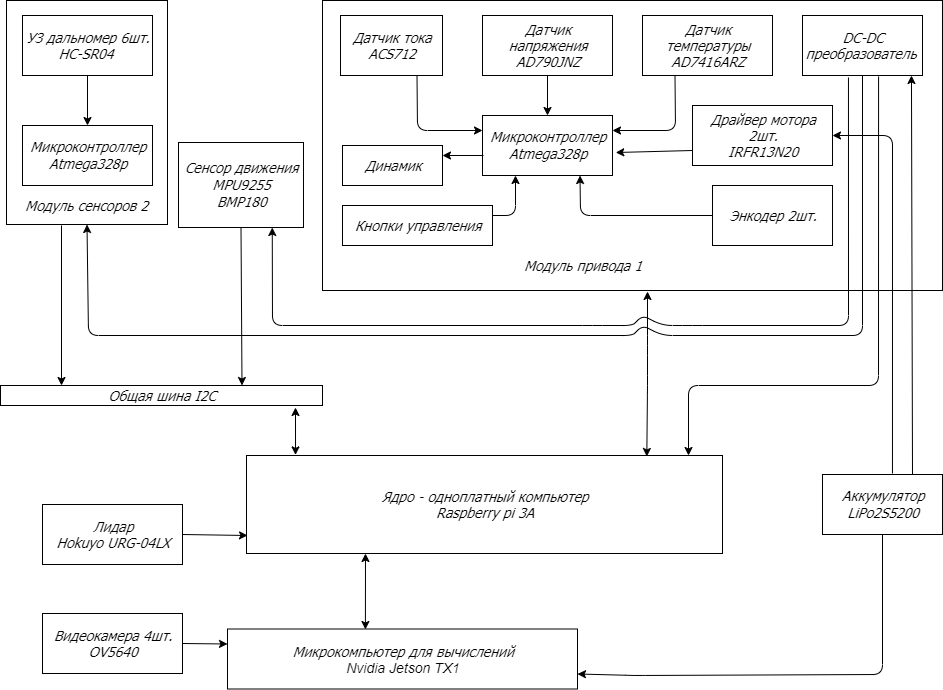


Рисунок 3 – функциональная схема электроники готовой платформы.

В итоге получили мобильную платформу в первом приближении, показанную на рисунке 4.

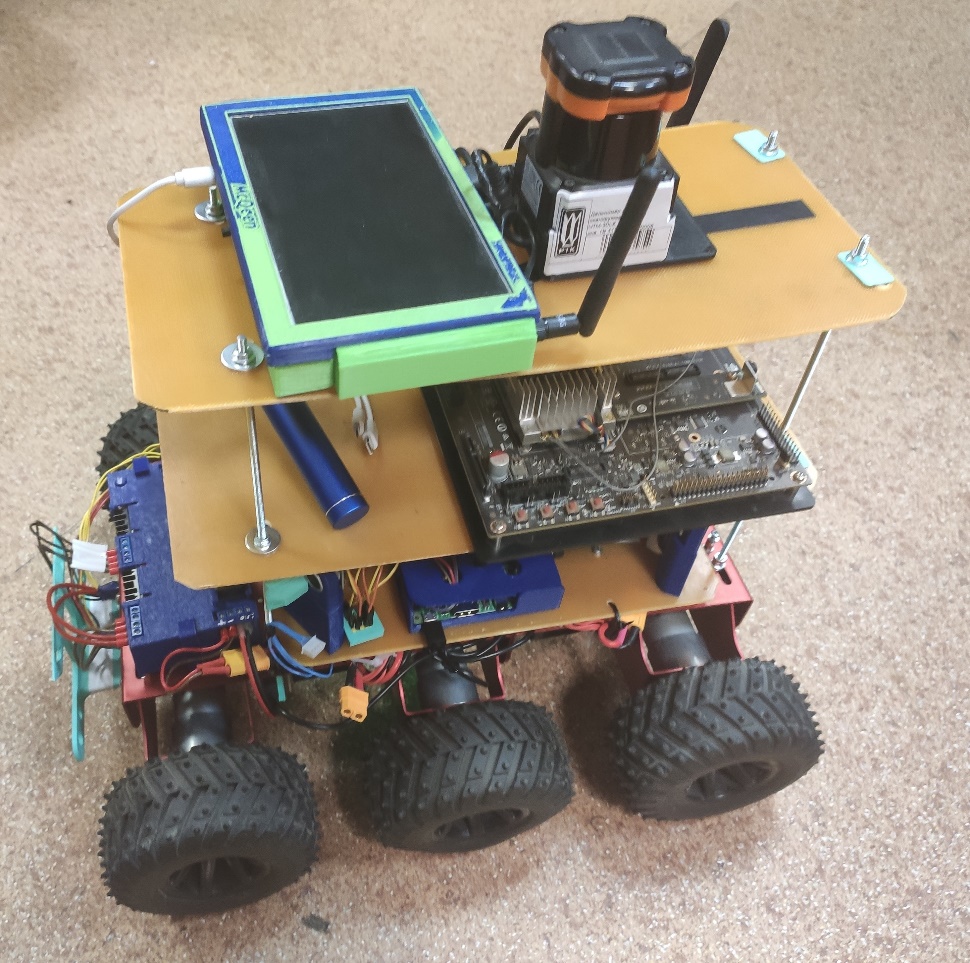


Рисунок 4 – Полученная мобильная платформа

Программная часть

Для обеспечения работы платформы был написан пакет neurobot5, файловая структура которого показана на рисунке 5, для ROS со следующими узлами: nb\_i2c\_data, nb\_core, nb\_assist, nb\_imu и nb\_pult5, схема работа которых показана на рисунке 6.

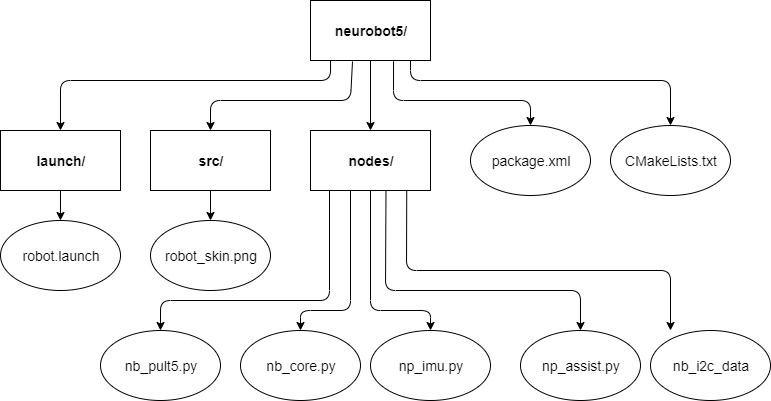


Рисунок 5 – Структура программных модулей уровня файловой системы

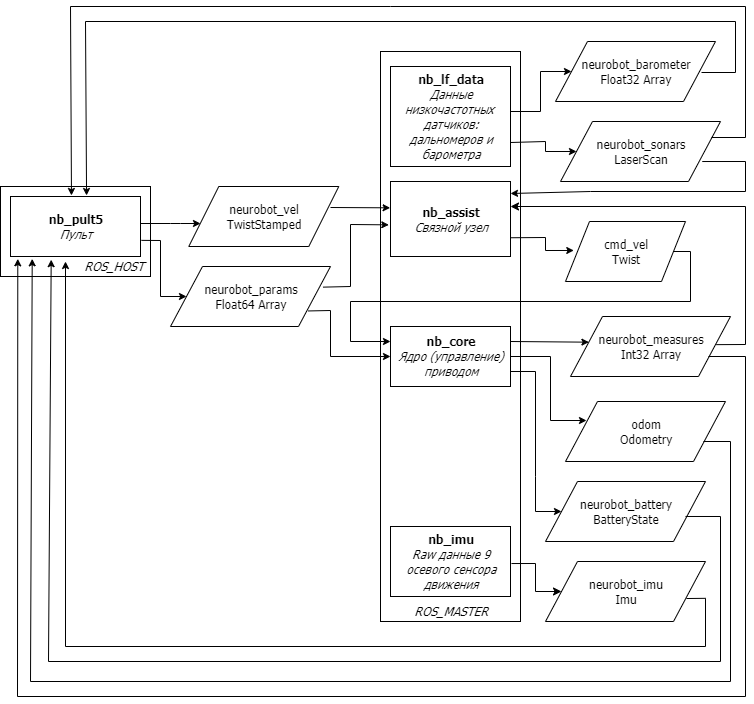


Рисунок 6 – схема работы узлов пакета neurobot5

Узел nb\_i2c\_data осуществляет выдачу данных, поступающих на шину I2C с низкой частотой. Это данные от шести ультразвуковых дальномеров HC-SR04, которые отправляются в топик neurobot\_sonars, а также показания барометра BMP180, который является частью сенсора движения: температура и давление отправляются в топик neurobot\_barometer. Настраиваемым параметром узла является частота его работы. Топик neurobot\_sonars имеет тип LaserScan() - кадр лазерного скана лидара, частота отправки равна 4 Гц, кортеж LaserScan.ranges состоит из шести элементов с диапазоном значений от 0 до 0.25 в метрах. Топик neurobot\_barometer имеет тип Float32MultiArray() – кортеж из вещественных чисел со следующим содержанием: [Температура, Давление] и обновляется с такой же частотой – 4 Гц. Код узла приведен в приложении А.

В узел nb\_core поступают данные из топиков neurobot\_params и cmd\_vel. Топик neurobot\_params имеет тип Float64MultiArray() – кортеж из вещественных чисел, представляется сообщением со следующим содержанием: [Избегание столкновений, Выключение системы, Диаметр колеса, Ширина колесной базы, Коэффициент редуктора, Максимальная скорость, Режим ПИД, Коэффициенты ПИД регулятора, Тип привода, Максимальная задержка]. Параметр «избегание столкновений» бинарный: активирует режим при значении «1», параметр «выключение системы» также бинарный, будет посылаться сигнал выключения к бортовому компьютеру при значении «1», параметр «диаметр колеса» указывается в метрах, также, как и «ширина колесной базы». Параметр «коэффициент редуктора» задает соотношение числа оборотов колеса к числу оборотов вала двигателя. Параметр «максимальная скорость» указывается в м/с, искусственно ограничивает скорость движения робота, а в случае превышения физически возможной – перестает влиять на систему. Параметр «режим ПИД» бинарный, при значении «0» скорость робота регулируется ПИД-регулятором, при значении «1» - задается коэффициент ШИМ исходя из отношения к максимальной скорости. Параметры «коэффициенты ПИД регулятора» это последовательность коэффициентов Kp, Ki, Kd - усиления пропорциональной, интегрирующей и дифференцирующей составляющих регулятора соответственно. Параметр «тип привода» бинарный, при значении «1» используется полный привод на роботе, при «0» - только средний мост. Параметр «максимальная задержка» указывается в секундах, определяет при какой задержке в сообщениях команды управления не будут учитываться. Топик cmd\_vel имеет тип сообщения Twist(), является командой движения, переменные Twist.linear.x и Twist.angular.z регламентируют линейную скорость робота в м/с и угловую в рад/с, диапазон их значений может быть любым, робот не сможет превысить физическое ограничение скорости.

Также узел выдает данные в топики neurobot\_measures, neurobot\_battery и odom. Топик neurobot\_measures имеет тип сообщения Int32MultiArray() – кортеж из целых чисел, представляется сообщением со следующим содержанием: [Кнопка1, Кнопка2, Напряжение, Ток, Заряд, Температура]. Элементы «кнопка1» и «кнопка2» бинарные, отражают состояние нажатия физических кнопок на роботе. Элементы «напряжение» и «ток» записываются в миливольтах и милиамперах, а элементы «заряд» и «температура» указываются в процентах и градусах соответственно, все они целочисленные и отражают текущее состояние робота. Топик neurobot\_battery имеет тип сообщения BatteryState(), выдает текущее состояние баттареи: BatteryState.voltage – напряжение в вольтах, BatteryState.current – ток в амперах, BatteryState.percentage – заряд в процентах в диапазоне от 0 до 1, все элементы описываются вещественными числами. Топик odom имеет тип сообщения Odometry, выдает текущую скорость робота и его положение: Odometry.twist.twist.linear.x – линейную скорость в м/с, Odometry.twist.twist.angular.z – угловую скорость в рад/с, Odometry.pose.pose.position.x – координата центра робота по оси Х в метрах, Odometry.pose.pose.position.y - координата центра робота по оси У в метрах, Odometry.pose.pose.orientation.w – абсолютный угол поворота робота в радианах, аналог рысканья. Все элементы сообщения описываются вещественными числами.

Узел nb\_core считывает данные, приходящие от модуля управления приводом по UART, такие как: скорость моторов, напряжение, ток, состояние кнопок, температура и отправляет данные в топик neurobot\_measures. Также узел принимает команды скорости из топика cmd\_vel, переводит их значения из кинематики угловой и линейной скорости в тангенциальную, подходящую для дифференциального привода, зная ширину базы из топика neurobot\_params, рассчитывает коэффициент ШИМ при помощи ПИД-регулятора, если он включен, и отправляет команду в модуль управления приводом по UART с указанием типа привода: [Тип привода, Напряжение левого мотора, Напряжение правого мотора]. Помимо этого, узел отправляет в топик neurobot\_battery данные по аккумулятору, а в топик odom значения скорости и положения центра робота на плоскости на основе данных с энкодера, предварительно вычислив, зная значения диаметра колеса и коэффициента редуктора из топика neurobot\_params. Частота работы узла равна 10 Гц. Настраиваемых параметров, помимо тех, что приходят из топика neurobot\_params нет. Блок-схема работы показана на рисунке 7. Код узла приведен в приложении Б.

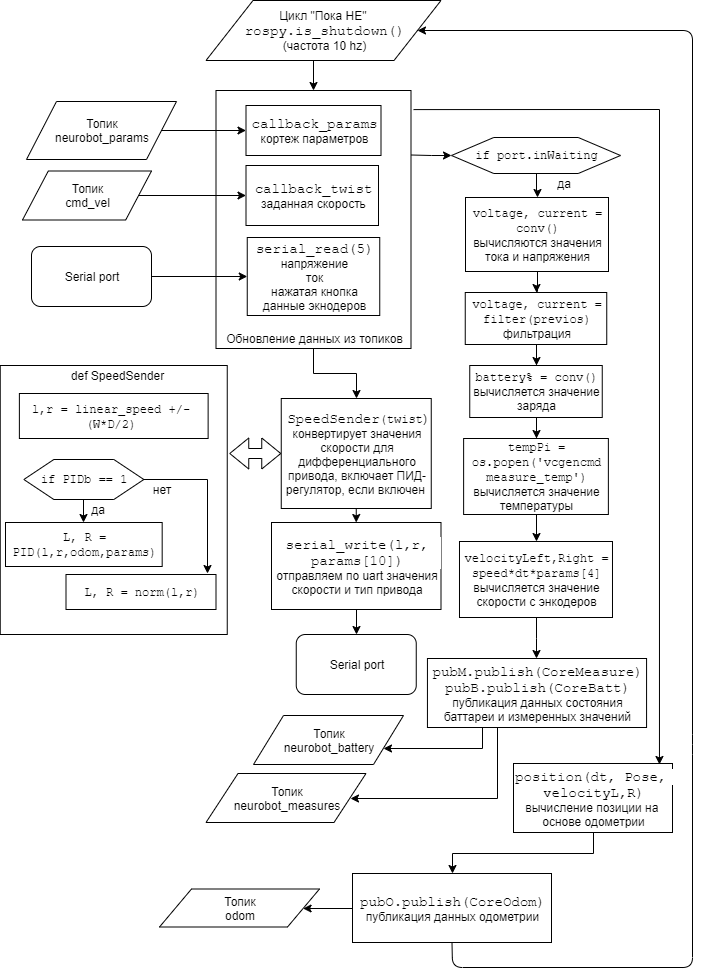


Рисунок 7 – Блок схема работы узла nb\_core

В узел nb\_assist поступают данные из топиков neurobot\_sonars, neurobot\_measures, neurobot\_vel, neurobot\_params. Топик neurobot\_vel имеет тип сообщения TwistStamped, является командой движения, переменные TwistStamped.twist.linear.x и TwistStamped.twist.angular.z регламентируют линейную скорость робота в м/с и угловую в рад/с, диапазон их значений может быть любым, а TwistStamped.header.stamp – время отправки сообщения. Также узел выдает данные в топик cmd\_vel.

Узел nb\_assist принимает команды скорости с внешнего компьютера из топика neurobot\_vel и если метка времени в сообщении превышает разницу с текущим временем больше, чем задает параметр из топика neurobot\_params, то такое сообщение не учитывается. В ином случае это сообщение сразу отправляется в топик cmd\_vel, если не включена функции избегания столкновения. Эта функция активируется при положительном значении соответствующего параметра и проверяет расстояние до объекта по направлению движения робота. Если расстояние до объекта с учетом выбранного запаса меньше, чем преодолеваемая дистанция за такт работы узла на основе данных текущей скорости или поступающей команды, то функция изменяет входящее значение скорости на то, что удовлетворит условию. Расстояние до объекта обновляется на основе данных из топика neurobot\_sonars. Также узел может послать сигнал выключения на одноплатный компьютер на основе нажатой кнопки выключения с модуля движителя или низком напряжении, состояние которых обновляется из топика neurobot\_measures или команды с внешнего компьютера, которая поступает в кортеже параметров. Частоту работы узла можно изменять. Блок-схема работы узла показана на рисунке 8. Код узла приведен в приложении В.

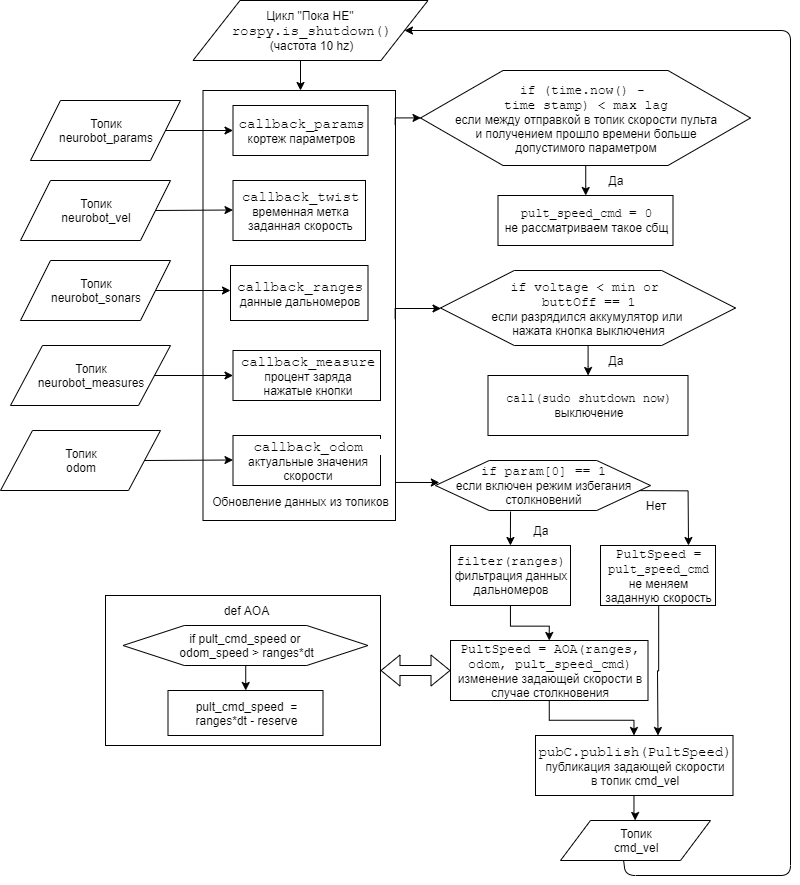


Рисунок 8 – Блок схема работы узла nb\_assist

Узел nb\_imu осуществляет выдачу данных, поступающих на шину I2C от 9-осевого сенсора движения MPU9255. Чтение регистров осуществляется с помощью функции imu.getIMUData() библиотеки «RTIMU». Установка поправок калибровки, осуществленной предварительно при помощи программы RTIMULibCal, происходит с помощью функции RTIMU.Settings(файл). Данные публикуются в топики neurobot\_imu и neurobotcompas. Настраиваемым параметром узла является частота его работы. Топик neurobot\_imu имеет тип Imu() - необработанные данные 9-осевого инерциального датчика: вектор трехмерного пространства Vector3 угловой скорости CoreImu.angular\_velocity в рад/с и Vector3 линейного ускорения CoreImu.linear\_aceleration в м/с2 c элементами типа float64. Частота отправки равна 10 Гц. Топик neurobot\_compass имеет тип MagneticField() - необработанные данные магнитометра в составе 9-осевого инерциального датчика: Vector3 компоненты CoreM.magnetic\_field в Тесла c элементами типа float64 и частотой отправки также 10 Гц. Код узла приведен в приложении Г.

В узел neurobot\_pult5 поступают данные из топиков neurobot\_measures, neurobot\_sonars, odom, neurobot\_battery, neurobot\_compass, neurobot\_barometer и neurobot\_imu. Также узел выдает данные в топики neurobot\_vel и neurobot\_params.

Узел nb\_pult5 отправляет команды скорости в топик neurobot\_vel на основе нажатых клавиш на клавиатуре внешнего компьютера с меткой текущего времени системы. Также узел отправляет набор параметров в топик neurobot\_params. Также узел отображает в отдельном окне 640 х 480 пикселей свой интерфейс со всеми поступающими и исходящими данными в удобном виде, пример которого показан на рисунке 9.

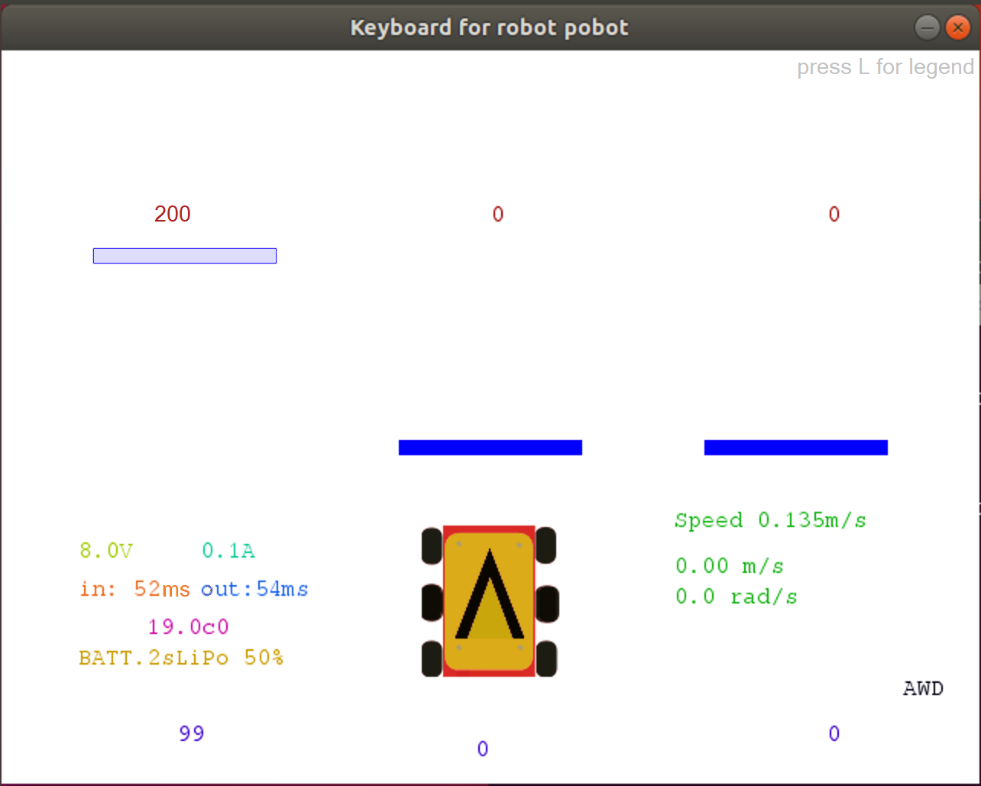


Рисунок 9 - Скрин интерфейса узла nb\_pult5

Среди отображаемые данных есть: данные с ультразвуковых датчиков, обновляемые из топика neurobot\_sonars, данные состояния модуля движителя из топика neurobot\_measures, данные аккумулятора из топика neurobot\_battery, данные температуры и давления из топика neurobot\_barometer, а также данные одометрии: положения в пространстве и скорости, поступающие из топика odom. При переключении основного вида миниатюра робота заменяется данными с инерциального датчика из топиков neurobot\_imu и neurobot\_compass. В интерфейсе узла можно включать и сбрасывать одометр для подсчета пройденного пути, а также включить фильтр низких частот, чтобы фильтровать данные от шума.

Выбор направления движения робота осуществляется нажатием клавиш со стрелками, а изменение отправляемой скорости осуществляется прибавлением по нажатию клавиши L\_shift или убавлением по нажатию R\_shift. По нажатию клавиши TAB выставляется максимальная скорость. Выбор типа привода осуществляется нажатием клавиши Space, активация функции избегания препятствий с помощью клавиши I. Клавиши P – выключает ПИД регулятор, F – включает фильтрацию поступающих данных, O – включает одометр, G – переключает вид робота на данные инерциального датчика, ESC – посылает сигнал выключения ядру, Backspace – выключает узел, L – отображает легенду клавиш. Частоту работы узла можно изменять. Код узла приведен в приложении Д.

Как было сказано ранее, в узел nb\_i2c\_data поступают данные из модуля сенсоров по шине I2C. Работа модуля сенсоров заключается в цикличном опросе датчиков ультразвуковых дальномеров – измерении времени прохождения сигнала. Опрос датчиков производится поочередно противоположных для уменьшения необходимой задержки для затухания сигналов.

Модуль привода работает следующим образом: совершается опрос всех датчиков и кнопки, после чего, производится отправка сообщения в порт UART их состояния и счетчиков энкодера - количество тиков, проделанных за такт. Счетчик обновляется при вызове внешнего прерывания при работе энкодера и обнуляется после совершения отправки сообщения. Затем принимается входящая команда, если поступила, и проверяется на соответствие. В случае истины обновляются параметры ШИМ сигнала драйверов. Если новая команда не поступает в течение трех тактов работы модуля – сигнал драйверам сменяется на остановку.

ПРИЛОЖЕНИЕ А

**СКРИПТ УЗЛА NB\_I2C\_DATA**

import sys

import time

import rospy

from std\_msgs.msg import Float32MultiArray # some meuserements?

from std\_msgs.msg import String # error msg

from sensor\_msgs.msg import LaserScan # Ultrasonic Scan 6x

from ctypes import c\_short

from ctypes import c\_byte

from ctypes import c\_ubyte

from smbus import SMBus

addri2cSonars = 0x8

addri2cbarometer = 0x77

bus = SMBus(1)

# bme280 not bmp180

def filter(sensor, prev, filtConst):

return ((sensor-prev)\*filtConst + (1-filtConst)\*prev)

def getShort(data, index):

# return two bytes from data as a signed 16-bit value

return c\_short((data[index+1] << 8) + data[index]).value

def getUShort(data, index):

# return two bytes from data as a signed 16-bit value

return c\_short((data[index+1] << 8) + data[index]).value

def getUShort(data, index):

# return two bytes from data as an unsigned 16-bit value

return (data[index+1] << 8) + data[index]

def getChar(data,index):

# return one byte from data as a signed char

result = data[index]

if result > 127:

result -= 256

return result

def getUChar(data,index):

# return one byte from data as an unsigned char

result = data[index] & 0xFF

return result

def readBME280ID(addr=addri2cbarometer):

# Chip ID Register Address

REG\_ID = 0xD0

(chip\_id, chip\_version) = bus.read\_i2c\_block\_data(addr, REG\_ID, 2)

return (chip\_id, chip\_version)

def readBME280All(addr=addri2cbarometer):

# Register Addresses

REG\_DATA = 0xF7

REG\_CONTROL = 0xF4

REG\_CONFIG = 0xF5

REG\_CONTROL\_HUM = 0xF2

REG\_HUM\_MSB = 0xFD

REG\_HUM\_LSB = 0xFE

# Oversample setting - page 27

OVERSAMPLE\_TEMP = 2

OVERSAMPLE\_PRES = 2

MODE = 1

# Oversample setting for humidity register - page 26

OVERSAMPLE\_HUM = 2

bus.write\_byte\_data(addr, REG\_CONTROL\_HUM, OVERSAMPLE\_HUM)

control = OVERSAMPLE\_TEMP<<5 | OVERSAMPLE\_PRES<<2 | MODE

bus.write\_byte\_data(addr, REG\_CONTROL, control)

# Read blocks of calibration data from EEPROM

# See Page 22 data sheet

cal1 = bus.read\_i2c\_block\_data(addr, 0x88, 24)

cal2 = bus.read\_i2c\_block\_data(addr, 0xA1, 1)

cal3 = bus.read\_i2c\_block\_data(addr, 0xE1, 7)

# Convert byte data to word values

dig\_T1 = getUShort(cal1, 0)

dig\_T2 = getShort(cal1, 2)

dig\_T3 = getShort(cal1, 4)

dig\_P1 = getUShort(cal1, 6)

dig\_P2 = getShort(cal1, 8)

dig\_P3 = getShort(cal1, 10)

dig\_P4 = getShort(cal1, 12)

dig\_P5 = getShort(cal1, 14)

dig\_P6 = getShort(cal1, 16)

dig\_P7 = getShort(cal1, 18)

dig\_P8 = getShort(cal1, 20)

dig\_P9 = getShort(cal1, 22)

dig\_H1 = getUChar(cal2, 0)

dig\_H2 = getShort(cal3, 0)

dig\_H3 = getUChar(cal3, 2)

dig\_H4 = getChar(cal3, 3)

dig\_H4 = (dig\_H4 << 24) >> 20

dig\_H4 = dig\_H4 | (getChar(cal3, 4) & 0x0F)

dig\_H5 = getChar(cal3, 5)

dig\_H5 = (dig\_H5 << 24) >> 20

dig\_H5 = dig\_H5 | (getUChar(cal3, 4) >> 4 & 0x0F)

dig\_H6 = getChar(cal3, 6)

# Wait in ms (Datasheet Appendix B: Measurement time and current calculation)

wait\_time = 1.25 + (2.3 \* OVERSAMPLE\_TEMP) + ((2.3 \* OVERSAMPLE\_PRES) + 0.575) + ((2.3 \* OVERSAMPLE\_HUM)+0.575)

time.sleep(wait\_time/1000) # Wait the required time

# Read temperature/pressure/humidity

data = bus.read\_i2c\_block\_data(addr, REG\_DATA, 8)

pres\_raw = (data[0] << 12) | (data[1] << 4) | (data[2] >> 4)

temp\_raw = (data[3] << 12) | (data[4] << 4) | (data[5] >> 4)

hum\_raw = (data[6] << 8) | data[7]

#Refine temperature

var1 = ((((temp\_raw>>3)-(dig\_T1<<1)))\*(dig\_T2)) >> 11

var2 = (((((temp\_raw>>4) - (dig\_T1)) \* ((temp\_raw>>4) - (dig\_T1))) >> 12) \* (dig\_T3)) >> 14

t\_fine = var1+var2

temperature = float(((t\_fine \* 5) + 128) >> 8);

# Refine pressure and adjust for temperature

var1 = t\_fine / 2.0 - 64000.0

var2 = var1 \* var1 \* dig\_P6 / 32768.0

var2 = var2 + var1 \* dig\_P5 \* 2.0

var2 = var2 / 4.0 + dig\_P4 \* 65536.0

var1 = (dig\_P3 \* var1 \* var1 / 524288.0 + dig\_P2 \* var1) / 524288.0

var1 = (1.0 + var1 / 32768.0) \* dig\_P1

if var1 == 0:

pressure=0

else:

pressure = 1048576.0 - pres\_raw

pressure = ((pressure - var2 / 4096.0) \* 6250.0) / var1

var1 = dig\_P9 \* pressure \* pressure / 2147483648.0

var2 = pressure \* dig\_P8 / 32768.0

pressure = pressure + (var1 + var2 + dig\_P7) / 16.0

return temperature/100.0, pressure/100.0

def mainfunc():

global bus

print('step1')

rospy.init\_node('nbcoredata', anonymous=True)

pubBar = rospy.Publisher('neurobotbarometer', Float32MultiArray, queue\_size=1)

pubUS = rospy.Publisher('neurobotsonarscan', LaserScan, queue\_size=1)

# pubError = rospy.Publisher('neuroboterror', String, queue\_size=1)

rate = rospy.Rate(4) # 4 - 10hz

CoreBar = Float32MultiArray()

CoreScan = LaserScan()

CoreScan.range\_min = 0.02

CoreScan.range\_max = 2.56

CoreScan.scan\_time = 0.3

CoreScan.angle\_increment = 1.04719

CoreScan.angle\_min = -2.09439

CoreScan.angle\_max = 3.1415

# Error = String()

print('step2')

while not rospy.is\_shutdown():

i2cstr = []

for i in range(7):

i2cstr.append(bus.read\_byte(addri2cSonars)) # six values in cm

CoreScan.ranges = [float(i2cstr[0])/100, float(i2cstr[1])/100,

float(i2cstr[4])/100, float(i2cstr[5])/100,

float(i2cstr[3])/100, float(i2cstr[2])/100]

#rospy.loginfo(CoreScan)

temperature, pressure = readBME280All()

CoreBar.data = [temperature, pressure]

pubB.publish(CoreBar)

pubUS.publish(CoreScan)

#Error.data = "none"

#pubError.publish(Error)

rate.sleep()

if \_\_name\_\_ == '\_\_main\_\_':

try:

mainfunc()

except rospy.ROSInterruptException:

pass

ПРИЛОЖЕНИЕ Б

**СКРИПТ УЗЛА NB\_CORE**

import sys

import time

import rospy

import datetime

import serial

import os

#import tf

import math

from math import sin, cos, pi

from subprocess import call

from std\_msgs.msg import Int32MultiArray # meuserements

from std\_msgs.msg import Float32MultiArray # params

from std\_msgs.msg import String # error msg

from nav\_msgs.msg import Odometry # odometry

from geometry\_msgs.msg import TwistStamped # cmd\_vel

from geometry\_msgs.msg import Point, Pose, Quaternion, Twist, Vector3

from sensor\_msgs.msg import LaserScan # Ultrasonic Scan 6x

Timing = 10

dt = float(1)/Timing

PoseXYth = [0.00001, 0, 0]

halftimespeed = 0 # speed filter

koefBreak = 2.5 # koef of breaking system to AOA

margin = 0.05 # in m for AOA

PIDpreverrM1 = 0 # for pid

PIDintegralM1 = 0

PIDpreverrM2 = 0

PIDintegralM2 = 0

port = serial.Serial("/dev/ttyAMA0", baudrate=57600, timeout=3.0)

recv\_cmd\_vel = TwistStamped()

recv\_params = [0, 0, 0.12, 0.268, 0.00406, 0.3, 1, 0.1, 0.1, 0.1, 1, 2]

# AOA, PWR, wheelDia, wheelBase, EncToSpeed, MaxSpeed, PorV, PID.P, PID.I, PID.D, FullDrive, MaxLagInS

two\_encoders = [0, 0, 0, 0, 0, 0, 0, 0]

# Nah, Nah, Nah, Nah, Nah, Nah, LeftTicks, RightTicks

def callback\_params(data):

global recv\_params

#rospy.loginfo(rospy.get\_caller\_id() + 'I heard %s', data.data)

recv\_params = data.data

def callback\_encoders(data):

global two\_encoders

two\_encoders = data.data

def position(dt, vx, vy, vth, x, y, th):

delta\_x = (vx \* cos(th) - vy \* sin(th)) \* dt

delta\_y = (vx \* sin(th) + vy \* cos(th)) \* dt

delta\_th = vth \* dt

x += delta\_x

y += delta\_y

th += delta\_th

# since all odometry is 6DOF we'll need a quaternion created from yaw

#odom\_quat = tf.transformations.quaternion\_from\_euler(0, 0, th)

odom\_quat = [0, 0, sin(th/2), cos(th/2)]

return Pose(Point(x, y, 0.), Quaternion(\*odom\_quat))

def filter(sensor, prev, filtConst):

return ((sensor-prev)\*filtConst + prev)

def AOA(speedy, cmdSpeed, cmdAngle, dist1, dist2):

global halftimespeed, koefBreak, margin, dt

halftimespeed = filter(speedy, halftimespeed, 0.5)

if (speedy > 0):

if (speedy\*dt) > (dist1-margin)/koefBreak:

cmdSpeed = 0

elif (speedy < 0):

if (-speedy\*dt) > (dist2-margin)/koefBreak:

cmdSpeed = 0

return [cmdSpeed, cmdAngle]

def PID(error, integral, previous\_error, limits, Kp, Ki, Kd):

global dt

integral = integral + error \* dt

derivative = (error - previous\_error) / dt

output = float(Kp \* error + Ki \* integral + Kd \* derivative)\*limits / 100

previous\_error = error

if (output > limits):

output = limits

elif (output < - limits):

output = -limits

return output, integral, previous\_error

def SpeedSender(speed, angle, realSpeed, realAngle):

global recv\_params # drive, PorV, MaxSpeed, WheelBase, PIDs

global PIDpreverrM1, PIDpreverrM2, PIDintegralM1, PIDintegralM2

l = ((2\*speed) - (angle \* recv\_params[3]))/2

r = ((2\*speed) + (angle \* recv\_params[3]))/2

if (l > recv\_params[5]):

l = recv\_params[5]

elif (l < -recv\_params[5]):

l = -recv\_params[5]

if (r > recv\_params[5]):

r = recv\_params[5]

elif (r < -recv\_params[5]):

r = -recv\_params[5]

leftS = ((2\*realSpeed) - (realAngle \* recv\_params[3]))/2

rightS = ((2\*realSpeed) + (realAngle \* recv\_params[3]))/2

if (recv\_params[6] == 1):

lout, PIDpreverrM1,

PIDintegralM1 = PID(l-leftS, PIDintegralM1, PIDpreverrM1, 100,

recv\_params[7], recv\_params[8], recv\_params[9])

rout, PIDpreverrM2,

PIDintegralM2 = PID(r-rightS, PIDintegralM2, PIDpreverrM2, 100,

recv\_params[7], recv\_params[8], recv\_params[9])

else:

lout = int(recv\_params[5]/l)\*100

rout = int(recv\_params[5]/r)\*100

if (lout > 100):

lout = 100

elif (lout < -100):

lout = -100

if (rout > 100):

rout = 100

elif (rout < -100):

rout = -100

if (lout < 0):

L = 60 - int(lout\*0.51)

else:

L = int(lout\*0.51)

if (rout < 0):

R = 60 - int(rout\*0.51)

else:

R = int(rout\*0.51)

return str(chr(125+recv\_params[10])+chr(L)+chr(R))

def mainfunc():

global port, recv\_cmd\_vel, Timing, dt, recv\_params, two\_encoders,\

PoseXYth, UartCmd

print('step1')

rospy.init\_node('nbcorecontrol', anonymous=True)

pubO = rospy.Publisher('odom', Odometry, queue\_size=1)

#pubError = rospy.Publisher('neuroboterror', String, queue\_size=1)

rate = rospy.Rate(Timing) # 10hz

CoreOdom = Odometry()

CoreOdom.header.frame\_id = "odom"

CoreOdom.child\_frame\_id = "base\_link"

SonarScan = LaserScan()

PultSpeed = TwistStamped()

while not rospy.is\_shutdown():

rospy.Subscriber('neurobotmeasure',

Int32MultiArray, callback\_encoders)

rospy.Subscriber('neurobotparams', Float32MultiArray,

callback\_params)

velocityLeft = two\_encoders[6]\*recv\_params[4]\*Timing

velocityRight = two\_encoders[7]\*recv\_params[4]\*Timing

speedBase = (velocityLeft + velocityRight)/2

angleSpeed = (velocityRight - velocityLeft)/recv\_params[3]

CoreOdom.header.stamp = rospy.Time.now()

CoreOdom.pose.pose = position(dt, speedBase, 0, angleSpeed,

PoseXYth[0], PoseXYth[1], PoseXYth[2])

CoreOdom.twist.twist = Twist(Vector3(speedBase, 0, 0),

Vector3(0, 0, angleSpeed))

pubO.publish(CoreOdom)

rospy.Subscriber('neurobotsonarscan', LaserScan, SonarScan)

sonars = SonarScan.ranges

#sonars[2] = filter

rospy.Subscriber('cmd\_vel', TwistStamped, PultSpeed)

pult\_speed\_cmd = [PultSpeed.twist.linear.x,

PultSpeed.twist.angular.z] # speed, angle

if ((rospy.Time.now() - PultSpeed.header.stamp).to\_sec() >

recv\_params[11]):

print("big lag")

pult\_speed\_cmd[0] = 0

pult\_speed\_cmd[1] = 0

if (recv\_params[0] == 1): # AOA on

pult\_speed\_cmd = AOA(speedBase, pult\_speed\_cmd[0],

pult\_speed\_cmd[1], sonars[2], sonars[5])

port.write(SpeedSender(pult\_speed\_cmd[0], pult\_speed\_cmd[1],

speedBase, angleSpeed))

if (recv\_params[1] == 1): # shutdown now

print("shutdown now")

#call("sudo shutdown now", shell=True)

rate.sleep()

if \_\_name\_\_ == '\_\_main\_\_':

try:

mainfunc()

except rospy.ROSInterruptException:

pass

ПРИЛОЖЕНИЕ В

**СКРИПТ УЗЛА NB\_ASSIST**

import rospy

#from sys import stdout

from math import sin, cos, pi

from subprocess import call

from std\_msgs.msg import Float32MultiArray # params

from std\_msgs.msg import String # error msg

from nav\_msgs.msg import Odometry # odometry

from geometry\_msgs.msg import TwistStamped, Twist # speed

from sensor\_msgs.msg import LaserScan # Ultrasonic Scan 6x

Timing = 10

dt = float(1)/Timing

halftimespeed = 0 # speed filter

koefBreak = 2.5 # koef of breaking system to AOA

margin = 0.05 # in m for AOA

pult\_speed\_cmd = [.0, .0]

odom\_speed = [.0, .0]

recv\_params = [0, 0, 0.12, 0.268, 0.00406, 0.3, 1, 0.1, 0.1, 0.1, 1, 2]

# AOA, PWR, wheelDia, wheelBase, EncToSpeed, MaxSpeed, PorV, PID.P, PID.I, PID.D, FullDrive, MaxLagInS

rangesScan = [.0, 0, 0, 0, 0, 0]

delayPult = 0

def callback\_params(data):

global recv\_params

#rospy.loginfo(rospy.get\_caller\_id() + 'I heard %s', data.data)

recv\_params = data.data

def callback\_odom(data):

global odom\_speed

odom\_speed = [data.twist.twist.linear.x, data.twist.twist.angular.z]

def callback\_twist(data):

global pult\_speed\_cmd, delayPult

delayPult = data.header.stamp

pult\_speed\_cmd = [data.twist.linear.x, data.twist.angular.z]

def callback\_ranges(data):

global rangesScan

rangesScan = data.ranges

# low pass filter

def filter(sensor, prev, filtConst):

return ((sensor-prev)\*filtConst + prev)

def AOA(speedReal, cmdSpeed, cmdAngle, dist\_top, dist\_bottom):

global halftimespeed, koefBreak, margin, dt

halftimespeed = filter(speedy, halftimespeed, 0.5)

if (abs(speedReal) > abs(cmdSpeed)):

speedy = speedReal

else:

speedy = cmdSpeed

if (speedy > 0):

if (speedy\*dt) > (dist\_top-margin)/koefBreak:

cmdSpeed = 0

elif (speedy < 0):

if (-speedy\*dt) > (dist\_bottom-margin)/koefBreak:

cmdSpeed = 0

return [cmdSpeed, cmdAngle]

def mainfunc():

global pult\_speed\_cmd, Timing, dt, recv\_params, odom\_speed,\

rangesScan, delayPult

print('step1')

rospy.init\_node('nbcoreassist', anonymous=True)

rate = rospy.Rate(Timing) # 10hz

pubC = rospy.Publisher('cmd\_vel', Twist, queue\_size=1)

#pubError = rospy.Publisher('neuroboterror', String, queue\_size=1)

PultSpeed = Twist()

sonars = [0, 0, 0, 0, 0, 0]

delayPult = rospy.Time.now()

while not rospy.is\_shutdown():

rospy.Subscriber('neurobotparams', Float32MultiArray,

callback\_params)

rospy.Subscriber('neurobot\_vel', TwistStamped, callback\_twist)

rospy.Subscriber('odom', Odometry, callback\_odom)

rospy.Subscriber('neurobotsonarscan', LaserScan, callback\_ranges)

sonars[2] = filter(rangesScan[2], sonars[2], 0.5)

sonars[5] = filter(rangesScan[5], sonars[5], 0.5)

delay = (rospy.Time.now() - delayPult).to\_sec()

if (delay > recv\_params[11]):

print("\r\033[F BIG lag %.2f s!!!!!!!!" % delay)

pult\_speed\_cmd[0] = 0

pult\_speed\_cmd[1] = 0

else:

print("\r\033[F working with %.2fs lag" % delay)

if (recv\_params[0] == 1): # AOA on

pult\_speed\_cmd = AOA(odom\_speed[0], pult\_speed\_cmd[0],

pult\_speed\_cmd[1], sonars[2], sonars[5])

PultSpeed.linear.x = pult\_speed\_cmd[0]

PultSpeed.angular.z = pult\_speed\_cmd[1]

pubC.publish(PultSpeed)

if (recv\_params[1] == 1): # shutdown now

print("shutdown now")

#call("sudo shutdown now", shell=True)

rate.sleep()

if \_\_name\_\_ == '\_\_main\_\_':

try:

mainfunc()

except rospy.ROSInterruptException:

pass

ПРИЛОЖЕНИЕ Г

**СКРИПТ УЗЛА NB\_IMU**

import RTIMU

import os.path

import math

import rospy

from std\_msgs.msg import String # error msg

#from std\_msgs.msg import Int32 # angle

from sensor\_msgs.msg import Imu # Gyro and acel

from sensor\_msgs.msg import MagneticField # Magnetic field

from smbus import SMBus

addri2cimu = 0x68

bus = SMBus(1)

SETTINGS\_FILE = "RTIMULib.ini"

s = RTIMU.Settings(SETTINGS\_FILE)

imu = RTIMU.RTIMU(s)

if (not imu.IMUInit()):

sys.exit(1)

imu.setSlerpPower(0.02)

imu.setGyroEnable(True)

imu.setAccelEnable(True)

imu.setCompassEnable(True)

def filter(sensor, prev, filtConst):

return ((sensor-prev)\*filtConst + (1-filtConst)\*prev)

def mainfunc():

print('step1')

data = dict()

if imu.IMURead():

data = imu.getIMUData()

rospy.init\_node('nbcoreimu', anonymous=True)

pubI = rospy.Publisher('neurobotimu', Imu, queue\_size=1)

pubM = rospy.Publisher('neurobotcompas', MagneticField, queue\_size=1)

# pubError = rospy.Publisher('neuroboterror', String, queue\_size=1)

rate = rospy.Rate(10) # 4 - 10hz

CoreM = MagneticField()

CoreI = Imu()

# Error = String()

print('step2')

gyro = [0, 0, 0]

acel = [0, 0, 0]

while not rospy.is\_shutdown():

gyro = [0, 0, 0]

acel = [0, 0, 0]

if imu.IMURead():

data = imu.getIMUData()

gyro = data['gyro']

acel = data['accel']

magnetic = data['compass']

#angle = (math.atan2(a[0],a[1])\*180/math.pi)

#print(angle)

#print(gyro)

#print(accel)

CoreI.angular\_velocity\_covariance = gyro

CoreI.linear\_aceleration\_covariance = acel

CoreM.magnetic\_field\_covariance = magnetic

pubM.publish(CoreM)

pubI.publish(CoreI)

#Error.data = "none"

#pubError.publish(Error)

rate.sleep()

if \_\_name\_\_ == '\_\_main\_\_':

try:

mainfunc()

except rospy.ROSInterruptException:

pass

ПРИЛОЖЕНИЕ Д

**СКРИПТ УЗЛА NB\_PULT5**

import pygame

import socket

import datetime

import sys

import rospy

import tf

import math

from math import sin, cos, pi, acos

from std\_msgs.msg import Int32MultiArray # meuserements

from std\_msgs.msg import Float32MultiArray # params

from std\_msgs.msg import String # error msg

from nav\_msgs.msg import Odometry # odometry

from geometry\_msgs.msg import TwistStamped # cmd\_vel

from geometry\_msgs.msg import TwistStamped # cmd\_vel

from geometry\_msgs.msg import Point, Pose, Quaternion, Twist, Vector3

from sensor\_msgs.msg import LaserScan # Ultrasonic Scan 6x

from sensor\_msgs.msg import BatteryState # battery

pygame.init()

# preprocedure

# pygame settings

pygame.display.set\_caption('Keyboard for robot pobot')

size = [640, 480]

screen = pygame.display.set\_mode(size)

clock = pygame.time.Clock()

# by default the key repeat is disabled call set\_repeat() to enable it

pygame.key.set\_repeat(50, 50)

robot\_skin = pygame.image.load('catkin\_ws/src/neurobot5/src' +

'/robotskin.png').convert()

robot\_skin.set\_colorkey((255, 255, 255))

robot\_skin = pygame.transform.scale(robot\_skin,

(robot\_skin.get\_width()//2,

robot\_skin.get\_height()//2))

# graphics

# colors

white = (255, 255, 255)

red = (255, 0, 0)

grey = (100, 100, 100)

green = (0, 255, 0)

blue = (10, 10, 250)

black = (5, 0, 20)

violet = (200, 10, 160)

darkred = (155, 0, 0)

contrastgreen = (20, 175, 20)

popblue = (55, 0, 200)

lightred = (255, 200, 200)

lightgreen = (200, 255, 200)

orangedark = (200, 150, 0)

bluishred = (235, 100, 30)

bluishblue = (30, 100, 230)

# rendered text and pic

myfont = pygame.font.SysFont("monospace", 15)

GOfont = pygame.font.SysFont("monospace", 90)

GOlabel = GOfont.render(str("GAME OVER"), 1, black)

text\_filtered = myfont.render("Filtered", 1, violet)

text = ["key LEFT, RIGHT, UP, DOWN - control movement of robot",

"L\_Shift - decrease speed limit", "R\_Shift increase speed limit",

"TAB - increase speed limit to 90% - 'turbo mode'",

"F - turn on/off low-pass filter for robots params",

"O - turn on/off odometry in meters", "P - PID turn on/off",

"I - AOA turn on/off", "SPACE - Full/middle drive",

"Backspace - exit of this application",

"ESC - Raspberry Pi shutdown signal", " ", "\v version 0.5"]

legendary = []

for line in text:

legendary.append(myfont.render(line, True, white))

warningtext = myfont.render("WARNING! BATT disCHG", 1, red)

textAWD = myfont.render("AWD", 2, black)

text2WD = myfont.render("2-WheelD", 2, black)

textAOA = myfont.render("AOA is on", 2, red)

# params defined

params\_common = [0, 0, 0.12, 0.268, 0.00406, 0.3, 1, 0.1, 0.1, 0.1, 1, 5]

# AOA, PWR, wheelDia, wheelBase, EncToSpeed, MaxSpeed, PorV, PID.P, PID.I,

# PID.D, FullDrive, MaxLagInS

# var list

trigger\_animation = True

fps = 10 # FPS!!!!

timePi = 0

timeComp = 0

M1 = 0 # speed for left board

M2 = 0

k = 0.5 # from max speed

prevvoltage = 0 # filter vars

batteryS = 2

prevcurr = 0

odomSpeed = 0

odomAngle = 0

odomDistance = 0

odomX = 0

odomY = 0

odomTheta = 0

deltatoPi = 0

deltafromPi = 0

btn = False # btns on board

btn2 = False

odometerBtn = False # btns on comps keyboard

btnFilterOn = False

legend = False

PIDbtn = 1

OnOff = 0

AOAbtn = 0

FullDrivebtn = 1

filtrSCANdata = [0, 0, 0, 0, 0, 0] # ranges filtering

unfiltrSCANdata = filtrSCANdata

AnimationCounter = 0 # temporary

dt = float(1)/fps # integral part

# callback vars

rangesScan = [0.99, 0, 0, 0, 0, 0]

measuresCore = [0, 0, 8000, 50, 100, 19, 0, 0]

# [ But1, But2, Voltage, Charge%, Current, Temp, EncoderTicks1, Encoder2 ]

# legend of keyboard

def legend\_show():

screen.fill(black)

for line in range(len(legendary)):

screen.blit(legendary[line], (100, 100+line\*15))

# filter common

def filter(sensor, prev, filtConst):

return ((sensor-prev)\*filtConst + prev)

# callback functions

def callback\_ranges(data):

global rangesScan

rangesScan = data.ranges

def callback\_measures(data):

global measuresCore

measuresCore = data.data

# odomList = [0, 0, 0, 0, 0] #speedx, anglespeed, posX, posY, w

def callback\_odometry(data):

global odomX, odomY, odomTheta, odomSpeed, odomAngle, timePi

timePi = data.header.stamp

odomSpeed = data.twist.twist.linear.x

odomAngle = data.twist.twist.angular.z

odomX = data.pose.pose.position.x

odomY = data.pose.pose.position.y

odomTheta = acos(data.pose.pose.orientation.w)\*2

# function that changes vars by butns are pressed

def keyboardCheck():

global k, M1, M2, btnFilterOn, odometerBtn, legend, PIDbtn, \

OnOff, AOAbtn, FullDrivebtn, AnimationCounter

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

sys.exit()

if event.type == pygame.KEYDOWN:

if event.key == pygame.K\_LEFT:

M1 = k

M2 = -k

if event.key == pygame.K\_RIGHT:

M1 = -k

M2 = k

if event.key == pygame.K\_UP:

M1 = k

M2 = k

if event.key == pygame.K\_DOWN:

M1 = -k

M2 = -k

if pygame.key.get\_mods() & pygame.KMOD\_LSHIFT:

k = k - 0.05

if event.key == pygame.K\_TAB:

k = 1

if pygame.key.get\_mods() & pygame.KMOD\_RSHIFT:

k = k + 0.05

if event.key == pygame.K\_ESCAPE:

OnOff = 1

if event.key == pygame.K\_f:

if (btnFilterOn is False):

btnFilterOn = True

else:

btnFilterOn = False

if event.key == pygame.K\_o:

if (odometerBtn is False):

odometerBtn = True

else:

odometerBtn = False

if event.key == pygame.K\_p:

if (PIDbtn == 0):

PIDbtn = 1

else:

PIDbtn = 0

if event.key == pygame.K\_i:

if (AOAbtn == 0):

AOAbtn = 1

else:

AOAbtn = 0

if event.key == pygame.K\_SPACE:

if (FullDrivebtn == 0):

FullDrivebtn = 1

else:

FullDrivebtn = 0

if (event.key == pygame.K\_l):

legend = True

if event.key == pygame.K\_BACKSPACE:

pygame.quit()

sys.exit()

AnimationCounter += 1

else:

M1 = 0

M2 = 0

AnimationCounter = 0

if k > 1:

k = 1

elif k < 0.1:

k = 0.1

# convert diff drive to steering ackerman

def representSpeed(Motor1, Motor2, BaseWidth):

X = (Motor1 + Motor2)/2

W = (Motor1 - Motor2)/BaseWidth

return X, W

# func shows media

def blitting():

# buttons f and space

if (btnFilterOn is True):

screen.blit(text\_filtered, (440, 410))

if (FullDrivebtn == 0):

screen.blit(text2WD, (540, 410))

else:

screen.blit(textAWD, (590, 410))

if (AOAbtn == 1):

screen.blit(textAOA, (15, 15))

# obstacles visual:

pygame.draw.rect(screen, (filtrSCANdata[1], filtrSCANdata[1], 255),

((60, 255-filtrSCANdata[1]), (120, 10)), 0)

pygame.draw.rect(screen, (filtrSCANdata[2], filtrSCANdata[2], 255),

((260, 255-filtrSCANdata[2]), (120, 10)), 0)

pygame.draw.rect(screen, (filtrSCANdata[3], filtrSCANdata[3], 255),

((460, 255-filtrSCANdata[3]), (120, 10)), 0)

# obstacles text:

label2 = myfont.render(str(filtrSCANdata[1]), 1, darkred)

label3 = myfont.render(str(filtrSCANdata[2]), 1, darkred)

label4 = myfont.render(str(filtrSCANdata[3]), 1, darkred)

label1 = myfont.render(str(filtrSCANdata[0]), 1, popblue)

label6 = myfont.render(str(filtrSCANdata[5]), 1, popblue)

label5 = myfont.render(str(filtrSCANdata[4]), 1, popblue)

screen.blit(label2, (115, 100))

screen.blit(label3, (320, 100))

screen.blit(label4, (540, 100))

screen.blit(label1, (115, 440))

screen.blit(label6, (310, 450))

screen.blit(label5, (540, 440))

# params:

text\_volt = myfont.render(str(float('{:.2f}'.format(prevvoltage/1000))) +

"V", 1, (155, 200, 0))

text\_curr = myfont.render(str(float('{:.2f}'.format(prevcurr/1000))) + "A",

1, (15, 200, 150))

text\_timePi = myfont.render("in:" + str(deltatoPi)+"ms", 1, bluishred)

text\_timeComp = myfont.render("out:"+str(deltafromPi)+"ms", 1, bluishblue)

text\_temp = myfont.render(str(float(measuresCore[5]))+"c0", 1, violet)

if (measuresCore[3] > 10):

text\_bat0l0 = myfont.render("BATT." + str(batteryS) + "sLiPo " +

str(measuresCore[3])+"%", 1, orangedark)

screen.blit(text\_bat0l0, (50, 390))

else:

screen.blit(warningtext, (50, 390))

screen.blit(text\_volt, (50, 320))

screen.blit(text\_curr, (130, 320))

screen.blit(text\_timePi, (50, 345))

screen.blit(text\_temp, (95, 370))

screen.blit(text\_timeComp, (130, 345))

# odometry:

if (PIDbtn == 0):

text\_limitSpeed = myfont.render("Speed " + str(k\*100)+"% PWM",

1, contrastgreen)

else:

text\_limitSpeed = myfont.render("Speed " + str(k\*(params\_common[5])) +

"m/s", 1, contrastgreen)

screen.blit(text\_limitSpeed, (440, 300))

text\_speeed = myfont.render(str('{:.2f}'.format(odomSpeed))+" m/s",

1, contrastgreen)

text\_speeedw = myfont.render(str('{:.1f}'.format(odomAngle))+" rad/s",

1, contrastgreen)

screen.blit(text\_speeed, (440, 330))

screen.blit(text\_speeedw, (440, 350))

if (odometerBtn is True):

screen.blit(myfont.render("Distance: " +

str('{:.1f}'.format(odomDistance)) +

" m", 1, darkred), (440, 370))

screen.blit(myfont.render("X:" + str('{:.1f}'.format(odomX)) +

" | Y:" + str('{:.1f}'.format(odomY)),

1, orangedark), (440, 390))

# draw a robots background

if (M1 < 0):

pygame.draw.rect(screen, lightred, ((320, 290), (60, 135)), 0)

elif (M1 > 0):

pygame.draw.rect(screen, lightgreen, ((320, 290), (60, 135)), 0)

if (M2 < 0):

pygame.draw.rect(screen, lightred, ((260, 290), (60, 135)), 0)

elif (M2 > 0):

pygame.draw.rect(screen, lightgreen, ((260, 290), (60, 135)), 0)

# draw a robot rotation

if (trigger\_animation is True):

if (M1 == 0 and M2 == 0):

screen.blit(robot\_skin, (275, 310))

else:

rotSkinRobot = pygame.transform.rotate(robot\_skin,

AnimationCounter\*(M1-M2))

screen.blit(rotSkinRobot, (320-(rotSkinRobot.get\_width()//2),

360-(rotSkinRobot.get\_width()//2) -

((M1+M2)/2\*AnimationCounter) % 10))

else:

rotSkinRobot = pygame.transform.rotate(robot\_skin, odomTheta)

screen.blit(rotSkinRobot,

(320 - (rotSkinRobot.get\_width()//2) -

(sin(odomTheta)\*odomSpeed\*AnimationCounter % 10)),

(360 - (rotSkinRobot.get\_width()//2) -

(cos(odomTheta)\*odomSpeed\*AnimationCounter % 10)))

# game over

if (btn is True):

screen.blit(GOlabel, (50, 180))

if (legend is True):

legend\_show()

# func for evaluations

def calculatingParams():

global prevcurr, prevvoltage, odometerBtn, odomDistance, fps, rangesScan,\

filtrSCANdata, timeComp, deltafromPi, deltatoPi

#print(rangesScan)

# odometry

if (odometerBtn is False):

odomDistance = 0

else:

odomDistance += abs(odomSpeed/fps)

# low pass filter

if (btnFilterOn is False):

filtrSCANdata[0] = int(rangesScan[0]\*100)

filtrSCANdata[5] = int(rangesScan[5]\*100)

filtrSCANdata[4] = int(rangesScan[4]\*100)

filtrSCANdata[1] = int(filter(rangesScan[1]\*100, filtrSCANdata[1], 0.5))

filtrSCANdata[2] = int(filter(rangesScan[2]\*100, filtrSCANdata[2], 0.5))

filtrSCANdata[3] = int(filter(rangesScan[3]\*100, filtrSCANdata[3], 0.5))

prevcurr = filter(measuresCore[4], prevcurr, 0.1)

prevvoltage = filter(measuresCore[2], prevvoltage, 0.5)

else:

filtrSCANdata[1] = int(filter(rangesScan[1]\*100, filtrSCANdata[1], 0.2))

filtrSCANdata[2] = int(filter(rangesScan[2]\*100, filtrSCANdata[2], 0.2))

filtrSCANdata[3] = int(filter(rangesScan[3]\*100, filtrSCANdata[3], 0.2))

filtrSCANdata[0] = int(filter(rangesScan[0]\*100, filtrSCANdata[0], 0.2))

filtrSCANdata[5] = int(filter(rangesScan[5]\*100, filtrSCANdata[5], 0.2))

filtrSCANdata[4] = int(filter(rangesScan[4]\*100, filtrSCANdata[4], 0.2))

prevcurr = filter(measuresCore[4], prevcurr, 0.05)

prevvoltage = filter(measuresCore[2], prevvoltage, 0.1)

# delay from pc&pi

timeComp = rospy.Time.now()

deltafromPi = ((timeComp - timePi).to\_nsec())//1000000 # in ms

deltatoPi = 50 - deltafromPi

# measuresCore[last] # corecontrol.timenow - twiststamped

# ############################################## main part here #####

def mainfunc():

global params\_common, timeComp, M1, M2, AOAbtn, OnOff, PIDbtn,\

FullDrivebtn, timePi

print('step0')

rospy.init\_node('pult5', anonymous=True)

timePi = rospy.Time.now()

#pubC = rospy.Publisher('turtle1/cmd\_vel', Twist, queue\_size=1)

pubC = rospy.Publisher('cmd\_vel', TwistStamped, queue\_size=1)

pubP = rospy.Publisher('neurobotparams', Float32MultiArray, queue\_size=1)

rate = rospy.Rate(fps) # 20hz

#PultSpeed = Twist()

PultSpeed = TwistStamped()

PultParams = Float32MultiArray()

while not rospy.is\_shutdown():

# batteryS = int(prevvoltage//3200)

rospy.Subscriber('neurobotmeasure', Int32MultiArray, callback\_measures)

rospy.Subscriber('neurobotsonarscan', LaserScan, callback\_ranges)

rospy.Subscriber('odom', Odometry, callback\_odometry)

keyboardCheck()

calculatingParams()

screen.fill(white)

blitting()

pygame.display.update()

params\_common[0] = AOAbtn

params\_common[1] = OnOff

params\_common[6] = PIDbtn

params\_common[10] = FullDrivebtn

PultParams.data = params\_common

pubP.publish(PultParams)

PultSpeed.twist.linear.x, PultSpeed.twist.angular.z = \

representSpeed(M1, M2, params\_common[3])

PultSpeed.header.stamp = timeComp

pubC.publish(PultSpeed)

rate.sleep()

#print(rospy.Time.now())

if \_\_name\_\_ == '\_\_main\_\_':

try:

mainfunc()

except rospy.ROSInterruptException:

pass