AutoNet18+

Техническое описание унифицированной платформы (робота)

(версия документа 1.0)

Введение

В данном документе описывается мобильный робот, используемый на соревнованиях Autonet 18+ в рамках Всероссийского фестиваля «Робофест 2020 Онлайн», проводимого в дистанционном формате. В документе описываются решения, взятые за основу при создании мобильного робота, их модификации, способ тестирования программных компонентов для управления роботом участниками соревнований Autonet 18+, а также порядок запуска решения и способ загрузки исходных кодов в робот во время проведения соревнований.

1. Спецификация комплектующих

1.1. Механическая платформа (шасси)

В качестве механической основы робота взято шасси радиоуправляемого автомобиля Traxxas Ford F-150 1:10 2Wd. Электроника модели была подвергнута модификации для реализации возможности автономного управления роботом.

Рис. 1. Вид шасси сбоку

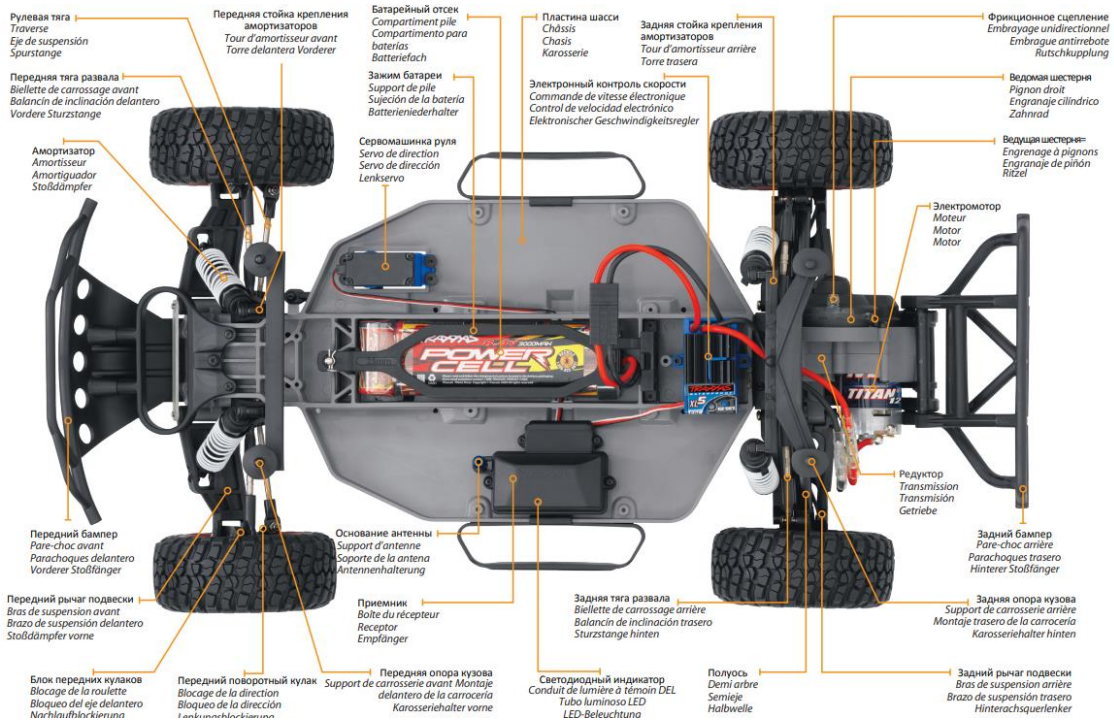

Рис. 2. Вид шасси снизу

Рис. 3. Вид шасси сверху

Стандартная платформа Traxxas подвергнута следующим модификациям:

- Вместо одного стандартного аккумулятора используются два аккумулятора (для питания компьютера NVIDIA JX2 и для питания других электронных компонент)
- Стандартный регулятор двигателя заменен на управляемый контроллер двигателя Pololu (см. Рис. 9).
- В текущей комплектации робота отсутствуют приемник и антенна, т.к. робот полностью автономный.

Рис. 4. Схема подключения бесколлекторного электродвигателя

Рис. 5. Актуальный вид робота

На рисунке 5 представлено шасси с двумя полками из оргстекла для размещения всех электронных компонент, двумя аккумуляторами снизу и местом под техническое оснащение сверху. Два аккумулятора используются из-за большого электропотребления одноплатного компьютера и двигателя постоянного тока. Шасси заднеприводное с рулевой тягой в виде сервопривода.

1.2. Двигатель DC

Рис. 6. Установленный электродвигатель Traxxas Titan 550 12T

Класс	550
Тип	Коллекторный
Напряжение	4.8-9.6
Мощность	200
Диаметр вала	3.2
Кол-во витков	12
Обороты на 1 Вольт	900
Длина двигателя	57.5
Диаметр двигателя	38
Рабочий ток	8.4

1.3. Поворотный двигатель

Рис. 7. Сервопривод для рулевого управления

Корпус защищен от попадания влаги и пыли, подшипник на главной шестерне

Габариты	40x20x39 мм	
Вес	45 г	
Подшипники	1	
Редуктор	Пластиковый	
Защита от воды	есть	
Напряжение	Усилие	Скорость
6.0 В	9.0 кг/см	0,17 сек/60грд

1.4. Контроллер для DC двигателя

Для программирования передвижения робота используется библиотека smc на Python.

Рис. 8. Pololu Simple High-Power Motor Controller 18v15

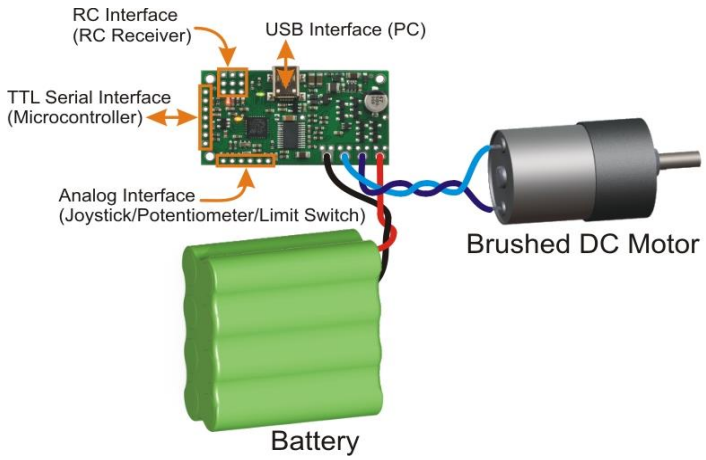

Рис. 9. Схема подключения двигателя к контроллеру

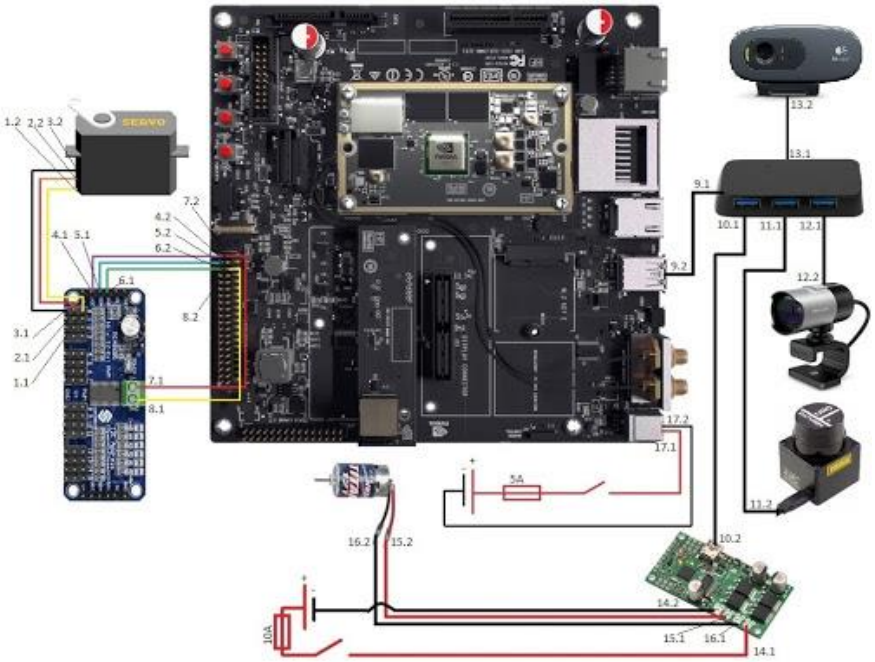

Рис. 10. Схема подключения технического оснащения к Jetson Nvidia TX2

Для управления двигателем постоянного тока (для прямолинейного движения робота) используется драйвер Pololu Simple Motor Controller, на вход 14.1 подается питание 12В, которое Pololu направляет на мотор 15.1. Заземление проходит через соединения (16.2-16.1 для мотора) и (14.2 для драйвера). Подключения (10.1-10.2), (11.1-11.2), (12.1-12.2), (13.1-13.2), соединяют USB разветвитель с драйвером Pololu, лазерным дальномером, веб-камерой Microsoft и веб-камерой Logitech соответственно. USB-hub общается с Jetson TX2 через (9.1-9.2). Основная плата питается от аккумулятора через (17.1-17.2)

Система взаимодействует с ШИМ контроллером через (4.1-4.2), (5.1-5.2) и (6.1-6.2) - I²C порт. Поступающие сигналы и питание (7.1-7.2), (8.1-8.2) направляются на сервомотор по (1.1-1.2), (2.1-2.2), (3.1-3.2).

На рис 10 представлен одноплатный компьютер Jetson Nvidia TX2, к которому через USB подсоединен контроллер Pololu.

1.5. Контроллер для сервопривода

Сам сервоконтроллер изображен на Рис. 11.

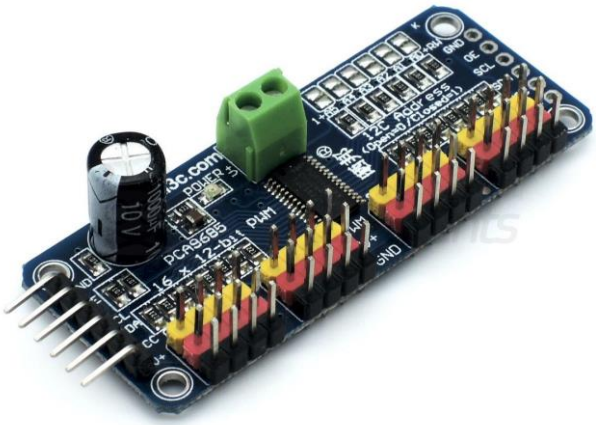

Рис. 11. Модуль управления сервоприводами 16-канальный

Управление и питание сервоконтроллера (см. Рис. 10) происходит за счет подключения к компьютеру Nvidia Jetson TX2 (см. рис 10) через микросхему PCA9685PW .

Для программирования управления используется библиотека Adafruit_Python_PCA9685.

1.6. Аккумулятор и зарядное устройство

Зарядное устройство аккумулятора

Рис. 12. Зарядное устройство универсальное Deep RC B6AC

1.7. Лидар

Рис. 13. Лазерный дальномер Lidar HOKUYO URG-04LX-UG01

Технические характеристики:

- Макс. диапазон измерения: 5,6м
- Диапазон измерения: 4.0м
- Сектор сканирования: 240°
- Разрешение: 0,36°
- Скорость сканирования: 100мс/скан.

- Габаритные размеры: 50x50x70мм
- Вес: 160г
- Потребляемый ток: 500мА
- Напряжение питания: 5В
- Интерфейс: USB 2.0

Программное управление лидаром осуществляется с использованием пакета ROS для лидата hokuy. Передача данных осуществляется через серийный порт.

1.8. Камеры с описанием крепления

В роботе используется две камеры: одна для считывания участка трассы со светофорами и дорожными знаками, другая для считывания линий дорожной разметки.

Рису. 14. Веб-камера Logitech HD Webcam C270

Рис. 15. Веб камера MICROSOFT LifeCam Studio

1.9. Управляющий компьютер NVIDIA Jetson TX2

Рис. 16. Одноплатный компьютер NVIDIA Jetson TX2

На рисунке 16 представлен компьютер NVIDIA Jetson TX2 на базе Ubuntu. В его состав входят два кластера с ядрами NVIDIA Denver 2 и ARM Cortex-A57. Чип поддерживает работу с 4K-видео с частотой до 60 кадров/с. Традиционно сильное место в устройствах NVIDIA - графика. За обработку визуальной информации отвечают 256 ядер CUDA в паре с 8 ГБ оперативной памяти типа LPDDR4. По умолчанию вся информация может сохраняться на встроенных 32 ГБ eMMC. Поддерживаются беспроводные стандарты связи Wi-Fi 802.11 ac и Bluetooth, а также Ethernet со скоростью до 1 Гбит/с.

Описание внешних интерфейсов:

PCIe	Gen 2 | 1x4 + 1x1 OR 2x1 + 1x2
USB	USB 3.0 + USB 2.0

Камера	До 6 камер (2-канальные) До 6 камер (2-канальные) CSI2 D-PHY 1.2 (2,5 Гб/с на канал)
Подключение	Подключение 801.11ac WLAN и Bluetooth
Флэш-память	32 Гб eMMC
Разъемы дисплея	HDMI
Прочие разъемы	GPIO, I2 C, I2 S, SPI, CAN, UART

2. Технические характеристики собранного робота

- **Габаритные размеры**
  Длина 568 мм
  Высота 220 мм

- **Скорость перемещения (min, max) и ее программные ограничения**
  Программные ограничения = 9 км/ч, рекомендованная скорость = 4км/ч. Максимальная скорость = 55 км/ч .

- **Радиус разворота** 123 см
- **Диаметр колес:** 105 мм
- **Дорожный просвет:** 89 мм
- **Ширина 296 мм**
- **Длина колесной базы: 927 мм**

3. Описание низкоуровневого программирования перемещения робота с примерами

Для регулировки перемещения используется ПИД-регулятор

листинг кода 1.1 Python - пример использования PID

```
from lib.PID import PID
from configs import config_adjustment as config
import rospy
from std_msgs.msg import String

class Adjustment:
    def __init__(self):
        #rospy.init_node('movement_adjustment', anonymous=False)
        rospy.Subscriber("walls_and_crossroad", String, self.callback)
        self.pub = rospy.Publisher('speed_and_angle', String,
queue_size=10)
        p = config.P
        i = config.I
        d = config.D
        self.pid = PID(p,i,d)
        self.pid.sample_time = config.SEC
        self.pid.setpoint = config.IDEAL_DISTANCE

    def calculate_angle(self,distance):
        angle = self.pid(distance)
        return angle
```

В листинге 1.2 представлен код для поворота за счет изменения положения рулевой тяги (угол поворота подбирается вручную), остановки. Для использования Pololu подходит библиотека SMC (быстрое переключение скоростей может привести к неисправности контроллера.) с помощью нее можно регулировать направления тока, скорость. Значение скорости подается в виде семибитной команды в диапазоне (-127,127), чему края интервалов эквиваленты максимально возможной скорости DC мотора(55 км/ч). А также рекомендуется использовать библиотеку Adafruit_Python_PCA9685 для управления серводвигателем (регулирования угла поворота) программное ограничение в 60 градусов максимального и минимального угла (-60,60).

листинг кода 1.2 Python - пример управления роботом

```
from __future__ import division
import time
from configs.config_controller_jetson import FORWARD_TIME
from smc import SMC
import time
# Import the PCA9685 module.
from configs.config_controller_jetson import
MOVE_TO_LEFT_TIME, MOVE_TO_RIGHT_TIME
from lib.Adafruit_Python_PCA9685 import PCA9685

class MotorController:

   def move_to_right(self, speed=10, sec=FORWARD_TIME,
angle=MOVE_TO_RIGHT_TIME):
       try:
           #self.mc.speed7b(speed)
           #time.sleep(2.5)
           self.servo.setServo(0, min(60, max(-60, angle)))
           time.sleep(2.8)
           self.servo.setServo(0,min(60,max(-60, 0)))
           #self.mc.stop()
       except KeyboardInterrupt:
            time.sleep(int(sec))
            self.servo.setServo(0, min(60, max(-60, 0)))
            #self.mc.stop()

   def move_to_left(self, speed=10, sec=FORWARD_TIME,
angle=MOVE_TO_LEFT_TIME):
       try:
          #self.mc.speed7b(speed)
          time.sleep(1)
          self.servo.setServo(0,min(60, max(-60,angle)))
          time.sleep(2.8)
          self.servo.setServo(0,min(60,0))
         # self.mc.stop()
       except KeyboardInterrupt:
            time.sleep(sec)
            self.servo.setServo(0, min(60,max(-60, 0)))
            self.mc.stop()
```

```
    def move_forward(self, speed=10,sec=FORWARD_TIME):
        try:
            self.servo.setServo(0,0)
            self.mc.speed7b(speed)
            time.sleep(2.8)
            time.sleep(2.4)
            self.mc.stop()
        except KeyboardInterrupt:
             time.sleep(sec)
             self.servo.setServo(0,0)
             self.mc.stop()

    def set_angle(self, angle):
        self.servo.setServo(0, min(60, max(-60, angle)))

    def move_angle(self, sec=MOVE_TO_RIGHT_TIME, angle=0):
        self.servo.setServo(0, min(60, max(-60, angle)))
        time.sleep(sec)
        self.servo.setServo(0, min(60, max(-60, 0)))

    def motor_init(self, port="/dev/ttyACM1"):
        while not self.is_dc_initialized:
            try:
                self.mc = SMC(port, 115200)
                # self.mc = serial.Serial(port)
                self.mc.init()
                self.is_dc_initialized = True
            except KeyboardInterrupt:
                exit()
        print('motor_initialized!')

    def __init__(self):
        self.is_dc_initialized = False
        self.motor_init()
        self.servo = PCA9685()
        self.servo.servos[0].set(signed=True, reverse=True, min=120,
max=120, trim=0, exp=0)

    def set_speed(self, speed=0):
        self.mc.speed7b(speed)

    def stop_motor(self):
```

```
        self.mc.stop()




if __name__ == "__main__":
    MotorController()
```

4. Программно доступные компоненты в структуре ROS проекта

4.1. Описание launch файла

Это конфигурационный xml файл, с расширением “.launch”, в котором указываются устанавливаемые параметры и узлы для запуска, а также машины на которых они должны работать.

4.2. Сведения о версиях используемого ПО

- ROS melodic
- python v 3.6
- pip 20.0.2
- numpy 1.17.3
- rviz 1.13.5
- Keras-Applications 1.0.8
- tensorflow 1.14.0
- catkin 0.7.19
- pyserial 3.4
- requests 2.22.0
- Adafruit-Blinka 4.1.0

5. Порядок запуска робота с решением участников

- Участники размещают свои решения в репозитории git (github, bitbucket, gitlab), добавляют в репозиторий служебного пользователя - бота, аккаунт которого предоставляется

организаторами соревнований. Название проекта должно соответствовать имени команды с префиксом autonet18p, например, autonet18p2020_team_robolife

- Участники заполняют форму, в которой указывают адрес репозитория и название запускаемого launch файла не позднее, чем за 7 дней до проведения соревнований. В репозитории должны располагаться исходные коды приложения участников, а также launch файлы для запуска решения. На первом этапе это может быть шаблонное решение, предоставляемое организаторами.
- Организаторы рассылают сообщения со ссылкой на полученный репозиторий командам по email, указанным в форме, не позднее чем за 5 дней до соревнований с целью проверки корректности полученных данных.
- Организаторы в назначенное время скачивают репозиторий, просматривают исходные коды программ (осуществляют небольшое code review), размещают решения в папке с решением ROS.
- Для запуска заезда организаторами сначала запускаются ноды датчиков и нода со слушателем на управление роботом (исходные коды предоставляются в открытом репозитории). Затем при запуске робота для каждого участника запускается его launch файл из соответствующей папки.
- Обновление исходных кодов из репозиториев участников осуществляется не позднее, чем за 30 минут до каждого заезда.
- Лог скачивания исходных кодов предоставляется участникам для отслеживания того, что на робот будет загружена актуальная версия управляющего ПО.