Министерство науки и высшего образования РФ Федеральное государственное бюджетное образовательное учреждение высшего образования «Удмуртский государственный университет»

Институт гражданской защиты

Кафедра цифровых инженерных технологий

# Лабораторная работа №1

по дисциплине

Прикладное программное обеспечение

Выполнил студент

группы ОМ-20.04.01.04-11

Вологжанин Е. А.

Проверил к.т.н. доцент

Клековкин А. В

Оглавление

[Лабораторная работа №1 1](#_Toc162960045)

[Цель 3](#_Toc162960046)

[Задачи 3](#_Toc162960047)

[Теоретическая часть 3](#_Toc162960048)

[Практическая часть 4](#_Toc162960049)

[Задача 1 4](#_Toc162960050)

[Задача 2 5](#_Toc162960051)

[Вывод 5](#_Toc162960052)

[Приложение 1 6](#_Toc162960053)

[Приложение 2 6](#_Toc162960054)

# Цель

Реализовать движение мобильного робота, используя линейную и угловую скорости робота.

# Задачи

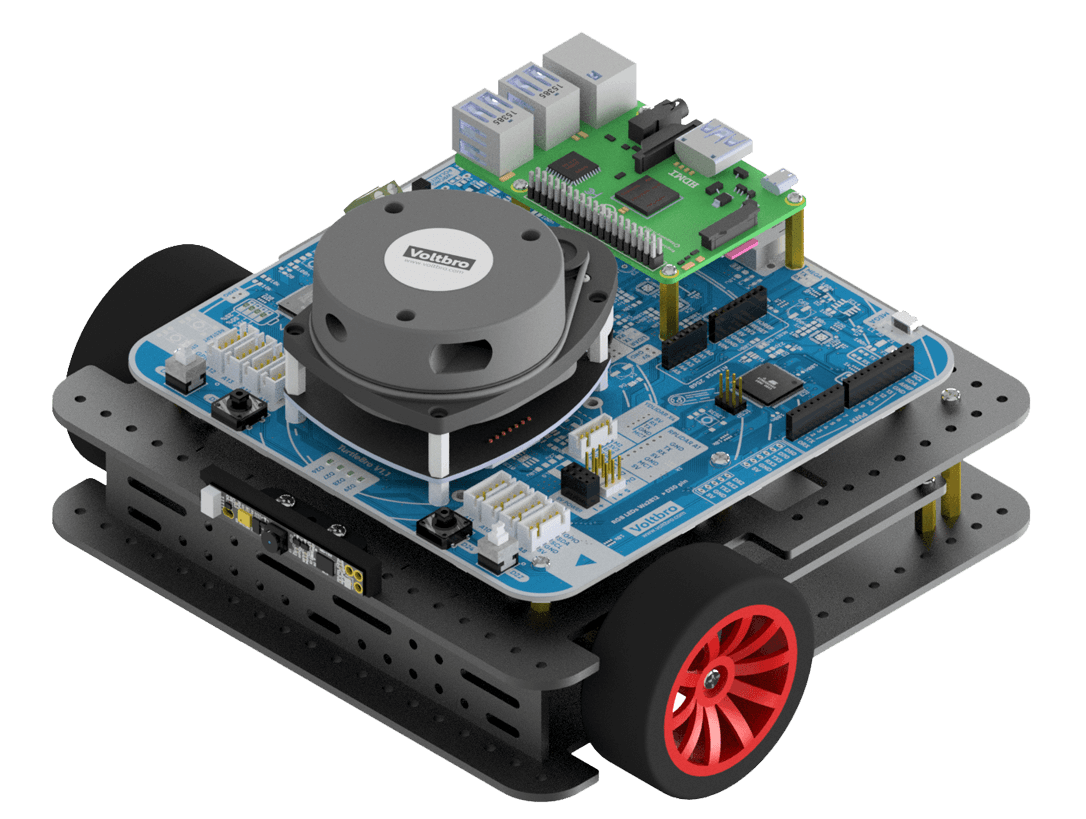
1. Передача данных о линейной и угловой скорости в топик /cmd\_vel;
2. Реализовать движение робота, используя клавиши на клавиатуре.

# Теоретическая часть

ROS (robot operating system) – это пакет промежуточного программного обеспечения для робототехники с открытым исходным кодом.

Основными языками программирования, предназначенными для работы с ROS, являются Python и C++.

Используемый робот turtlebro.



Для реализации движения робота необходимо воспользоваться определённым топиком, т.к. информация в ROS передаётся преимущественно с помощью топиков.

Используемым топиком выступит топик «/cmd\_vel». В него необходимо отправлять данные о линейной и угловой скоростях.

Возможности движения, используемого робота ограничены, например, нельзя дать роботу команду двигаться боком. Всё потому, что модель движения, используемая данным роботом, имеет возможность двигаться лишь вдоль оси X и поворачиваться вокруг оси Z.

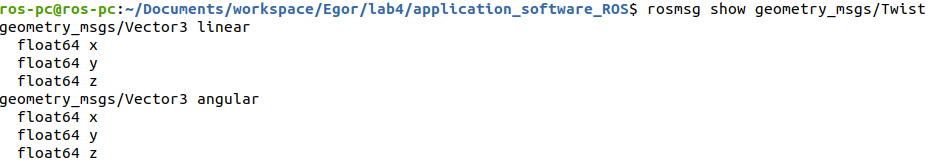
Таким образом для реализации движения робота необходимо изменять две координаты (oX – отвечает за линейную скорость oZ – отвечает за угловую скорость).

# Практическая часть

## Задача 1

При выполнении задачи было решено запустить работа двигаться по окружности. Для этого было отправлено сообщение типа «geometry\_msgs/Twist», содержащее информацию о необходимых скоростях – linear.x = 0.1 и angular.z = 0.1.

Структура сообщения:



Листинг программы прикреплён в Приложении 1.

## Задача 2

При выполнении задачи было решено использовать английскую раскладку клавиатуры и символы «w» – для движения вперёд, «s» – для движения назад, «a» – для поворота влево, «d» – для поворота вправо.

Также было решено использовать библиотеку getch. Модуль getch в Python предоставляет единую функцию getch.getch(), которая читает один символ с клавиатуры, не требуя нажатия Enter. Листинг программы прикреплён в Приложении 2.

# Вывод

В результате выполнения работы были получены основные навыки использования ROS и разработки программного обеспечения с использованием ROS. Также были успешно реализованы, поставленные, задачи, включающие ознакомление с работой топиков и движение робота посредством ввода символов в консоль.

# Приложение 1

|  |
| --- |
| *import* rospy  *from* geometry\_msgs.msg *import* Twist  class SimpleMove():      def \_\_init\_\_(*self*) -> None:  *self*.pub = rospy.Publisher("/cmd\_vel", Twist, *queue\_size*=10)        def move(*self*) -> None:          vel = Twist()          vel.linear.x = 0.1          vel.angular.z = 0.1  *self*.pub.publish(vel)  *if* \_\_name\_\_ == "\_\_main\_\_":      rospy.init\_node("simple\_move\_node")      sm = SimpleMove()  *while* not rospy.is\_shutdown():          sm.move()      rospy.spin() |

# Приложение 2

|  |
| --- |
| *import* rospy  *from* geometry\_msgs.msg *import* Twist  *import* getch  class ButtonMove():      def \_\_init\_\_(*self*) -> None:  *self*.vel = Twist()  *self*.base\_lin\_value = 0.01  *self*.base\_ang\_value = 0.05  *self*.pub = rospy.Publisher("/cmd\_vel", Twist, *queue\_size*=10)      def up\_lin\_vel(*self*, *value*): *#Forward*  *self*.vel.linear.x += *value*      def down\_lin\_vel(*self*, *value*): *#back*  *self*.vel.linear.x -= *value*      def up\_ang\_vel(*self*, *value*): *#left*  *self*.vel.angular.z += *value*      def down\_ang\_vel(*self*, *value*): *#right*  *self*.vel.angular.z -= *value*      def stop(*self*):  *self*.vel.linear.x = 0  *self*.vel.angular.z = 0      def pub\_vel(*self*):  *self*.pub.publish(*self*.vel)  *if* \_\_name\_\_ == "\_\_main\_\_":      rospy.init\_node("button\_move\_node")      bm = ButtonMove()  *while* not rospy.is\_shutdown():          symbol = getch.getch()  *if*(symbol == 'w'):              bm.up\_lin\_vel(bm.base\_lin\_value)  *elif*(symbol == 's'):              bm.down\_lin\_vel(bm.base\_lin\_value)  *elif*(symbol == 'a'):              bm.up\_ang\_vel(bm.base\_ang\_value)  *elif*(symbol == 'd'):              bm.down\_ang\_vel(bm.base\_ang\_value)  *else*:              bm.stop()          bm.pub\_vel()      rospy.spin() |