Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и кибербезопасности
Высшая школа программной инженерии

К У Р С О В О Й П Р О Е К Т

“Симуляция движений робота-собаки с использованием имитационного моделирования”

по дисциплине «Основы теории управления»

Выполнили
студенты гр. 5130904/20102 Симоненко И.С.
Попов Е.
Митрошин Д.В.

Руководитель Круглов С.К.

Санкт-Петербург

2025

СОДЕРЖАНИЕ

Введение

Современные технологии робототехники активно развиваются, и одним из перспективных направлений является создание роботов с бионическими функциями, способных имитировать поведение живых существ. Роботы-собаки, обладающие подвижными конечностями и сложными механизмами, применяются в научных исследованиях, обучении и промышленности. Для эффективной разработки алгоритмов управления движениями таких роботов важным этапом является создание их имитационных моделей — цифровых симуляций, которые воспроизводят кинематику и динамику движения устройства в виртуальной среде.

Имитационное моделирование позволяет тестировать и оптимизировать алгоритмы управления в условиях, близких к реальным, без необходимости непосредственного использования физического робота. Это существенно снижает затраты времени и ресурсов, а также повышает безопасность экспериментов. В рамках данного проекта рассматривается разработка симуляции движений робота-собаки, которая позволит обеспечить удобное средство для разработки, тестирования и совершенствования алгоритмов локомоции и управления.

Таким образом, реализация имитационной модели робота-собаки открывает возможности для глубокого изучения поведения робота в различных условиях и является важным компонентом в развитии робототехнических систем нового поколения.

Задание

Для выполнения курсовой работы необходимо реализовать алгоритмы движения робота-собаки на базе существующей имитационной модели. Основная задача заключается в программировании контроллера, который обеспечит управление движением конечностей и тела робота с учетом кинематических ограничений и физических параметров модели.

Задачи включают разработку и внедрение алгоритмов для различных видов движений (ходьба, поворот, смена положения корпуса), а также реализацию обратной кинематики для расчета управляющих воздействий на приводы робота. Необходимо провести тестирование точности и плавности движения, отладку программного обеспечения и оценку корректности работы в симуляционной среде. Результатом работы станет работающая система управления движениями модели робота-собаки, готовая к дальнейшему использованию и развитию.

Описание решения

Архитектура

В работе использовалась среда и аппаратная платформа робота МОРС — Малого Образовательного Робота-Собаки, созданного для образовательных и научно-исследовательских целей. Этот робот базируется на архитектуре x86 и работает под управлением Ubuntu Linux и Robot Operating System (ROS). Благодаря открытой архитектуре и открытому исходному коду, доступному на GitHub, была возможность реализовать и адаптировать алгоритмы локомоции и управления движением под конкретные задачи. Использование платформы МОРС существенно упростило процесс разработки за счет готового API и набора инструментов для взаимодействия с аппаратной частью и симуляцией, предоставив надежную и гибкую базу для создания сложных сценариев, таких как танцевальные движения робота-собаки.

В работе использовалась операционная система Ubuntu 20.04, запущенная внутри WSL2, что обеспечило гибкость и удобство работы с Linux-средой в Windows. ROS Noetic, как версия Robot Operating System, выбран для реализации системы управления роботом-собакой, так как именно эта версия ROS адаптирована под Ubuntu 20.04 и поддерживает современные инструменты и библиотеки.

ROS Noetic обеспечивает модульную архитектуру, где различные компоненты управления (например, подписка на команды движения, отправка управляющих сигналов сервоприводам, обработка данных сенсоров) взаимодействуют через топики и сервисы ROS. Это позволяет удобно структурировать код и упрощает интеграцию различных функций управления движением.

Симуляция движений робота-собаки реализована с использованием движка Bullet Physics — популярного физического движка, который обеспечивает реалистичную симуляцию динамики и столкновений. Bullet Physics позволяет точно моделировать физические свойства робота,

такие как масса, трение, упругость суставов, а также динамику движения конечностей под управлением сервоприводов.

Алгоритм

Алгоритм управления роботом-собакой в данном ROS-пакете организован как система, обеспечивающая взаимодействие между управляющими командами и аппаратным обеспечением робота через ROS-сообщения и сервисы. Основная задача алгоритма — преобразование команд движения (например, шаги, повороты, танцевальные элементы) в последовательности управляющих сигналов на актуаторы конечностей робота.

Основные этапы алгоритма включают:

1. Инициализация и настройка ROS-узлов, подписка на необходимые топики и подготовка к передаче команд на сервоприводы.

2. Прием управляющих команд движения из внешних источников — например, темповые команды, команды для исполнения танца или ходьбы.

3. Обработка команд с использованием кинематических моделей робота, где рассчитываются углы поворота суставов для каждой конечности с учетом текущего положения.

4. Генерация траекторий движения суставов для плавного и синхронизированного исполнения команд, включая циклы движения лап (шагающие циклы), повороты и сложные последовательности, например, танцевальные движения.

5. Отправка рассчитанных управляющих значений через ROS-сообщения к исполнительным механизмам (серводвигателям).

6. Мониторинг обратной связи, если имеется, для коррекции движений и стабилизации позы робота.

В совокупности данный алгоритм обеспечивает плавное и реалистичное движение робота-собаки в симуляции или на реальном устройстве, позволяя реализовать комплексные поведенческие сценарии, включая танцы, ходьбу и другие динамические движения.

Запуск симуляции

В рамках работы запуск симуляционной среды с физическим движком Bullet Physics и робота MOPC, а также разработанной управляющей ноды для реализации движений, выполнялся с помощью следующих последовательных команд в отдельных терминалах:

Для запуска симуляционной среды с моделью робота MOPC использовалась команда:

```
roslaunch mors bringup_sim.launch
```

Для запуска управляющей ноды с реализованными алгоритмами движений (например, танцевальных) использовалась команда:

```
rosrun dogzilla_moves custom_moves.py
```

Эти команды обеспечивали инициализацию всей симуляционной среды, загрузку URDF-модели с физическим движком Bullet и активацию собственного узла ROS, отвечающего за выполнение заданных последовательностей движений робота-собаки.

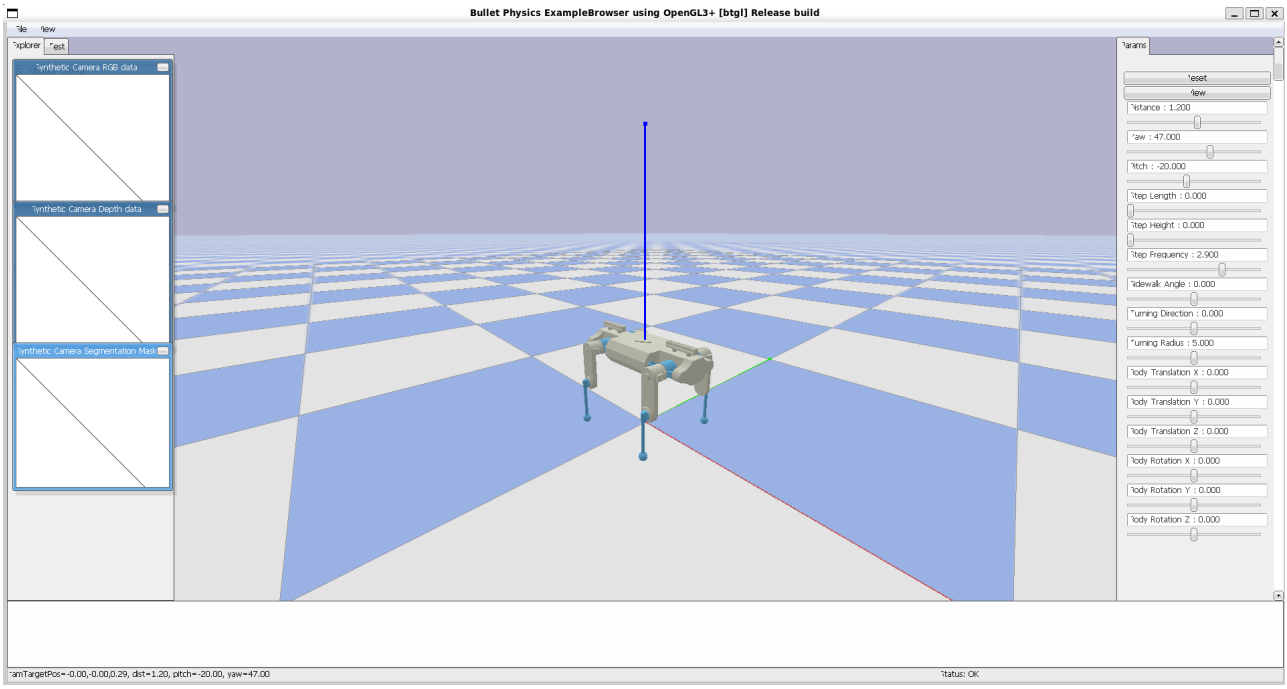
Демонстрация

Запуск симуляции:

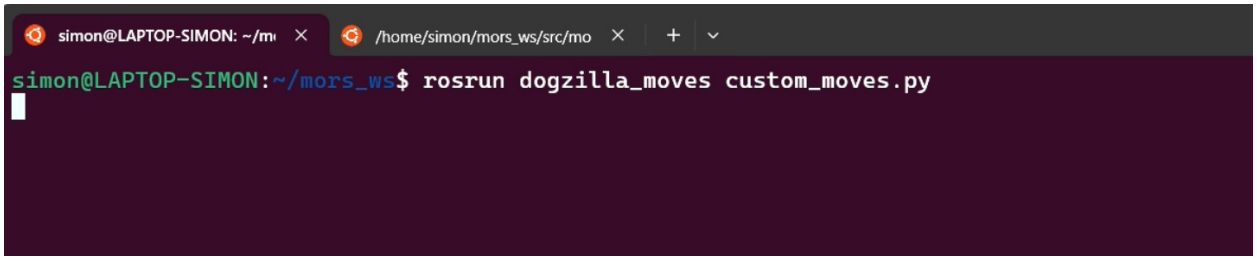
```
simon@LAPTOP-SIMON:~/mors_ws$ roslaunch mors bringup_sim.launch
... logging to /home/simon/.ros/log/c7fe68de-b3ce-11f0-af50-cda0b1147701/roslaunch-LAPTOP-SIMON-1414289.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://LAPTOP-SIMON:44131/
```

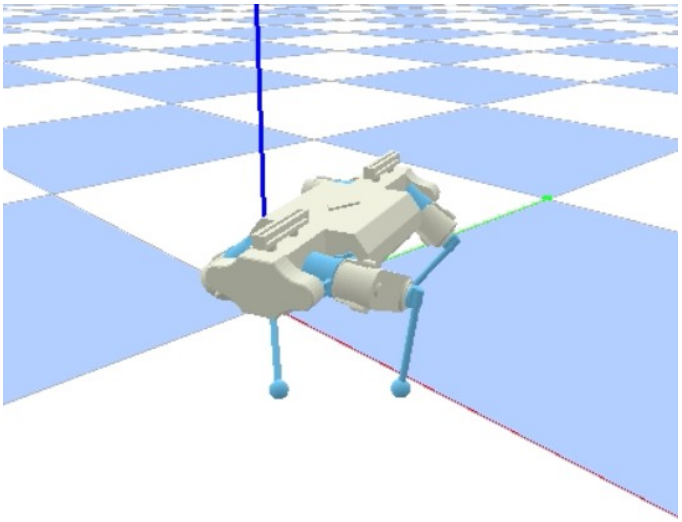
Окно симуляции:

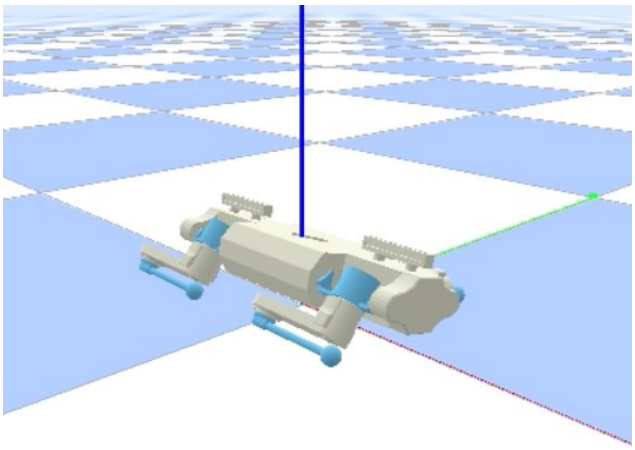


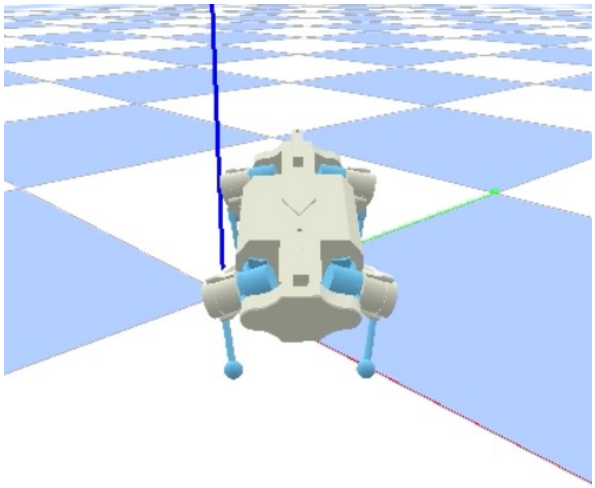
Запуск ноды:

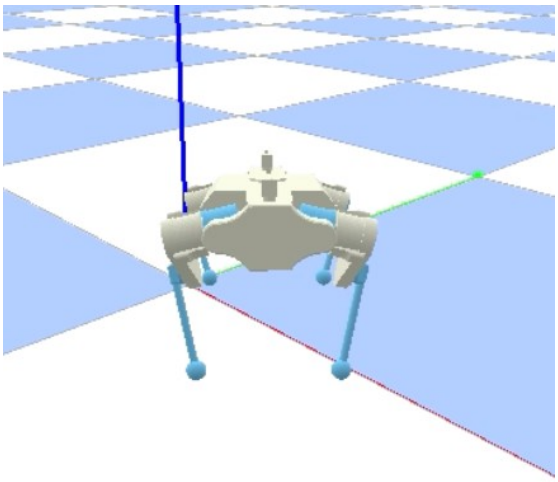


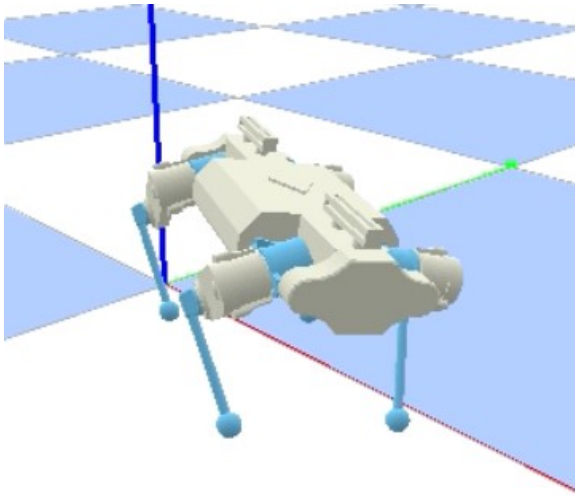
Движения робота:

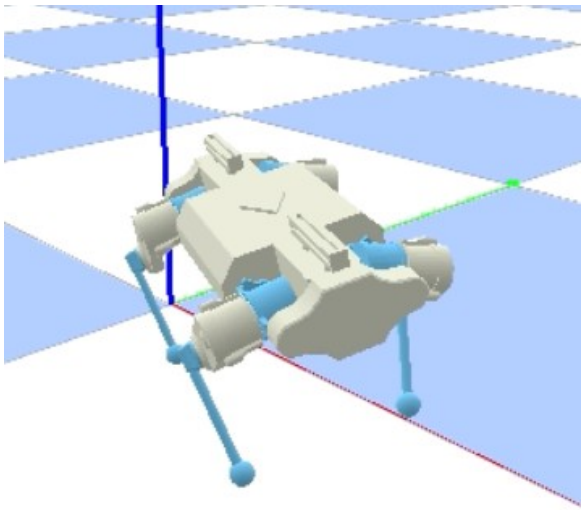




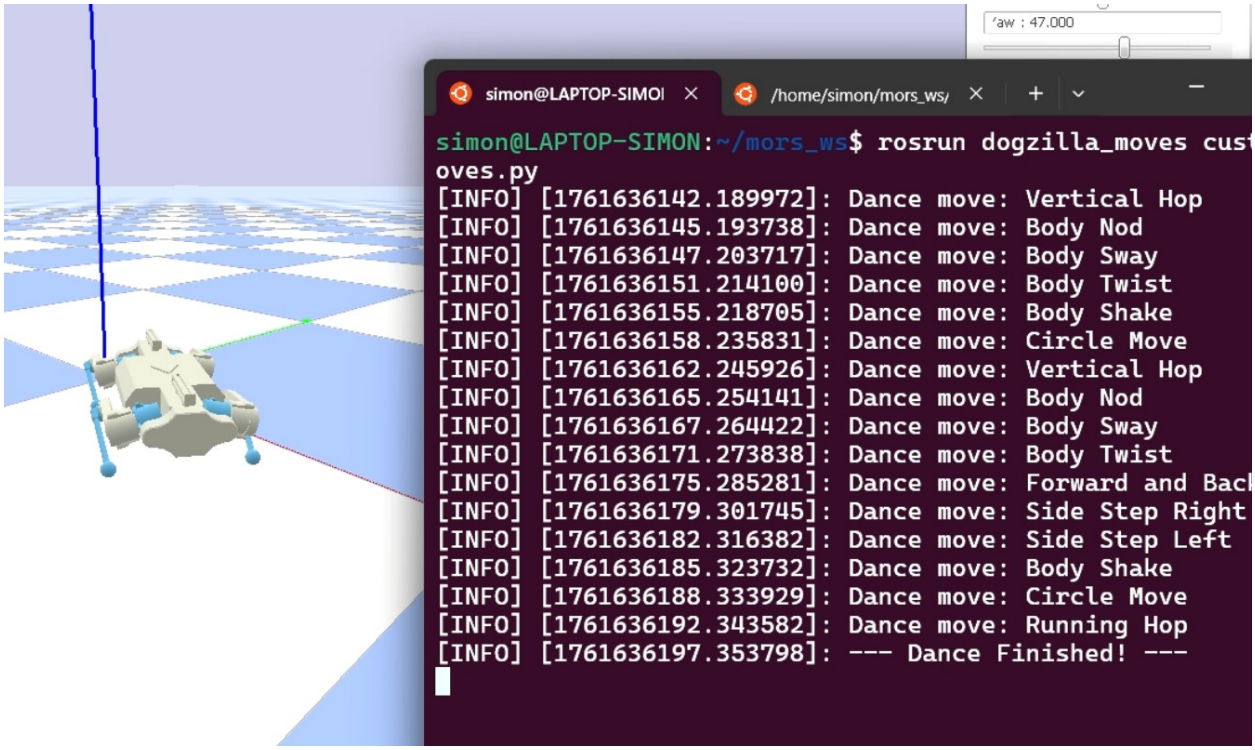








Вывод программы:



Выводы

В ходе работы была реализована симуляция движений робота-собаки на основе существующей имитационной модели, с применением среды ROS Noetic и физического движка Bullet Physics в Gazebo Classic под управлением Ubuntu 20.04 (WSL2). Выполнена разработка и запуск управляющего ROS-узла, отвечающего за воспроизведение заданных последовательностей движений, включая танцевальные элементы. Благодаря интеграции с платформой MOPC и использованию открытых исходных кодов и документации, удалось достичь высокой степени реализма движений и структурировать программное окружение для дальнейших экспериментов и творчества.

В результате работы цели и задачи исследования были достигнуты: разработанный алгоритм управления обеспечивает корректное и синхронизированное движение конечностей модели, а программное окружение позволяет эффективно тестировать и совершенствовать сценарии поведения робота. Полученный результат представляет интерес для образовательных и исследовательских проектов в области робототехники, а созданная база может быть расширена новыми типами движений и алгоритмами управления.

Ссылка на исходный код и реализованный функционал представлена в публичном репозитории: https://github.com/simonoffcc/dogzilla-moves

Список литературы

1. ROS Noetic Ninjemys: установка на Ubuntu 20.04 / ROS Wiki [Электронный ресурс]. – Режим доступа: http://wiki.ros.org/noetic/Installation/Ubuntu, свободный. – Дата обращения: 28.10.2025.
2. Установка ROS Noetic на Ubuntu 20.04 / Хабр [Электронный ресурс]. – Режим доступа: https://habr.com/ru/articles/551944/, свободный. – Дата обращения: 28.10.2025.
3. Voltbros. Официальная документация платформы MOPC [Электронный ресурс]. – Режим доступа: https://voltbro.gitbook.io/robot-sobaka-mors, свободный. – Дата обращения: 28.10.2025.
4. Bullet Physics SDK Manual [Электронный ресурс]. – Режим доступа: https://pybullet.org/Bullet/BulletFull/, свободный. – Дата обращения: 28.10.2025.
5. Gazebo Sim: Physics engines [Электронный ресурс]. – Режим доступа: https://gazebosim.org/docs/ros/physics, свободный. – Дата обращения: 28.10.2025.

Код программы

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import rospy
import numpy as np
from geometry_msgs.msg import Twist
from std_msgs.msg import Bool
from mors.srv import QuadrupedCmd

# --- Сервисные клиенты (остаются без изменений) ---
def set_mode_client(mode):
    rospy.wait_for_service('robot_mode')
    try:
        set_mode = rospy.ServiceProxy('robot_mode', QuadrupedCmd)
        resp = set_mode(mode)
        return resp.result
    except rospy.ServiceException as e:
        print("Service call failed: %s" % e)

def set_action_client(action):
    rospy.wait_for_service('robot_action')
    try:
        set_action = rospy.ServiceProxy('robot_action', QuadrupedCmd)
        resp = set_action(action)
        return resp.result
    except rospy.ServiceException as e:
        print("Service call failed: %s" % e)
```

```
# --- Функции для танцевальных движений ---

def body_shake(pub, status_pub, duration, rate):
    """Движение: тряска корпусом из стороны в сторону (крен)."""
    rospy.loginfo("Dance move: Body Shake")
    start_time = rospy.get_time()
    cmd_pose_msg = Twist()

    while rospy.get_time() - start_time < duration and not rospy.is_shutdown():
        t = rospy.get_time() - start_time
       cmd_pose_msg.angular.x = 0.4 * np.sin(4 * 2 * np.pi * t) # Быстрые колебания по крену
        pub.publish(cmd_pose_msg)
        status_pub.publish(True)
        rate.sleep()

def vertical_hop(pub, status_pub, duration, rate):
    """Движение: имитация прыжков на месте."""
    rospy.loginfo("Dance move: Vertical Hop")
    start_time = rospy.get_time()
    cmd_pose_msg = Twist()

    while rospy.get_time() - start_time < duration and not rospy.is_shutdown():
        t = rospy.get_time() - start_time
       cmd_pose_msg.linear.z = 0.06 * abs(np.sin(2 * 2 * np.pi * t)) # Используем abs для "подпрыгивания"
        pub.publish(cmd_pose_msg)
        status_pub.publish(True)
        rate.sleep()
```

```
def body_twist(pub, status_pub, duration, rate):
    """Движение: повороты корпуса (рысканье)."""
    rospy.loginfo("Dance move: Body Twist")
    start_time = rospy.get_time()
    cmd_pose_msg = Twist()

    while rospy.get_time() - start_time < duration and not rospy.is_shutdown():
        t = rospy.get_time() - start_time
      cmd_pose_msg.angular.z = 0.5 * np.sin(0.5 * 2 * np.pi * t) # Плавные повороты
        pub.publish(cmd_pose_msg)
        status_pub.publish(True)
        rate.sleep()

# --- Обычные движения ---

def forward_back(pub, status_pub, duration, rate):
    """Движение: вперед-назад по X."""
    rospy.loginfo("Dance move: Forward and Backward")
    start_time = rospy.get_time()
    msg = Twist()
    while rospy.get_time() - start_time < duration and not rospy.is_shutdown():
        t = rospy.get_time() - start_time
        msg.linear.x = 0.05 * np.sin(2 * np.pi * t)   # 1 Гц
        pub.publish(msg)
        status_pub.publish(True)
        rate.sleep()

def side_step(pub, status_pub, duration, rate, direction=1):
```

```
    """Движение: боковые шаги по Y. direction=1 вправо, -1 влево."""
      rospy.loginfo("Dance move: Side Step %s" % ("Right" if direction > 0 else "Left"))
    start_time = rospy.get_time()
    msg = Twist()
    while rospy.get_time() - start_time < duration and not rospy.is_shutdown():
        t = rospy.get_time() - start_time
        msg.linear.y = direction * 0.04 * np.sin(2 * np.pi * t)  # 1 Гц
        pub.publish(msg)
        status_pub.publish(True)
        rate.sleep()


def running_hop(pub, status_pub, duration, rate):
    """Бег вперед в припрыжку: поступательно по X + быстрые подпрыгивания по Z."""
    rospy.loginfo("Dance move: Running Hop")
    start_time = rospy.get_time()
    msg = Twist()
    base_speed = 0.08          # м/с эквивалент для визуального “бега”
    hop_amp = 0.06             # амплитуда прыжка по Z
    hop_freq = 6.0             # Гц подпрыгивания
    while rospy.get_time() - start_time < duration and not rospy.is_shutdown():
        t = rospy.get_time() - start_time
        msg.linear.x = base_speed  # постоянное поступательное движение
        msg.linear.z = hop_amp * abs(np.sin(2 * np.pi * hop_freq * t))
        pub.publish(msg)
        status_pub.publish(True)
        rate.sleep()
```

```
def body_sway(pub, status_pub, duration, rate):
    """НОВОЕ ДВИЖЕНИЕ: Плавное покачивание корпусом в плоскости XY."""
    rospy.loginfo("Dance move: Body Sway")
    start_time = rospy.get_time()
    cmd_pose_msg = Twist()

    while rospy.get_time() - start_time < duration and not rospy.is_shutdown():
        t = rospy.get_time() - start_time
        # Движение по эллипсу
        cmd_pose_msg.linear.x = 0.04 * np.sin(1 * 2 * np.pi * t)
        cmd_pose_msg.linear.y = 0.03 * np.cos(1 * 2 * np.pi * t)
        pub.publish(cmd_pose_msg)
        status_pub.publish(True)
        rate.sleep()

def body_nod(pub, status_pub, duration, rate):
    """НОВОЕ ДВИЖЕНИЕ: Кивки корпусом (тангаж)."""
    rospy.loginfo("Dance move: Body Nod")
    start_time = rospy.get_time()
    cmd_pose_msg = Twist()

    while rospy.get_time() - start_time < duration and not rospy.is_shutdown():
        t = rospy.get_time() - start_time
        # Колебания по тангажу (ось Y)
        cmd_pose_msg.angular.y = 0.35 * np.sin(2 * 2 * np.pi * t)
        pub.publish(cmd_pose_msg)
        status_pub.publish(True)
        rate.sleep()
```

```
def circle_move(pub, status_pub, duration, rate):
    """НОВОЕ ДВИЖЕНИЕ: Кружение на месте."""
    rospy.loginfo("Dance move: Circle Move")
    start_time = rospy.get_time()
    cmd_pose_msg = Twist()

    while rospy.get_time() - start_time < duration and not rospy.is_shutdown():
        # Постоянный поворот и небольшое смещение для имитации шага в сторону
        cmd_pose_msg.angular.z = 0.6  # Постоянная скорость поворота
        cmd_pose_msg.linear.y = 0.02 # Небольшое боковое смещение
        pub.publish(cmd_pose_msg)
        status_pub.publish(True)
        rate.sleep()

def main():
    try:
        rospy.init_node("mors_dance_script")
        rate = rospy.Rate(100)

        cmd_pose_pub = rospy.Publisher("/head/cmd_pose", Twist, queue_size=10)
        status_pub = rospy.Publisher("/head/status", Bool, queue_size=10)

        # Подготовка: встаем и переходим в нужный режим
        set_action_client(1)
        set_mode_client(2)
        rospy.sleep(2.0) # Пауза для стабилизации

        # Начало: легкие подпрыгивания и кивки
```

```
vertical_hop(cmd_pose_pub, status_pub, duration=3.0, rate=rate)
body_nod(cmd_pose_pub, status_pub, duration=2.0, rate=rate)

# Развитие: покачивания и повороты
body_sway(cmd_pose_pub, status_pub, duration=4.0, rate=rate)
body_twist(cmd_pose_pub, status_pub, duration=4.0, rate=rate)

# Кульминация: быстрая тряска и кружение
body_shake(cmd_pose_pub, status_pub, duration=3.0, rate=rate)
circle_move(cmd_pose_pub, status_pub, duration=4.0, rate=rate)

# Начало: легкие подпрыгивания и кивки
vertical_hop(cmd_pose_pub, status_pub, duration=3.0, rate=rate)
body_nod(cmd_pose_pub, status_pub, duration=2.0, rate=rate)

# Развитие: покачивания и повороты
body_sway(cmd_pose_pub, status_pub, duration=4.0, rate=rate)
body_twist(cmd_pose_pub, status_pub, duration=4.0, rate=rate)

# Вперед-назад несколько секунд
forward_back(cmd_pose_pub, status_pub, duration=4.0, rate=rate)

# Боком вправо, затем влево
side_step(cmd_pose_pub, status_pub, duration=3.0, rate=rate, direction=1)
side_step(cmd_pose_pub, status_pub, duration=3.0, rate=rate, direction=-1)

# Кульминация: быстрая тряска и кружение
body_shake(cmd_pose_pub, status_pub, duration=3.0, rate=rate)
circle_move(cmd_pose_pub, status_pub, duration=4.0, rate=rate)
```

```
        # Бег вперед в припрыжку
        running_hop(cmd_pose_pub, status_pub, duration=5.0, rate=rate)

        # Завершение
        rospy.loginfo("--- Dance Finished! ---")
    except rospy.ROSInterruptException:
        print("Program interrupted before completion.")
    finally:
        # 3. Возвращаемся в нейтральное положение и ложимся
        # Убедимся, что паблишеры были созданы, прежде чем их использовать
        if 'status_pub' in locals():
            status_pub.publish(False)
        if 'cmd_pose_pub' in locals():
            cmd_pose_pub.publish(Twist()) # Сброс позы

        rospy.sleep(1.0)
        set_action_client(2)
        rospy.loginfo("Script finished.")

if __name__ == '__main__':
    main()
```