# ПРЕОБРАЗОВАНИЕ КООРДИНАТ ТОЧКИ МЕЖДУ СИСТЕМАМИ КООРДИНАТ СУСТАВОВ РОБОТА

## Введение

Для работы с преобразованиями координат в ROS 2 используется утилита **tf2\_echo**. Она подписывается на систему трансформаций (TF) и выводит в консоль преобразование между двумя указанными системами отсчёта (фреймами).

**Важное условие:** Для работы утилиты робот должен публиковать описания своих фреймов и преобразований между ними в соответствующем топике, как это реализовать описано ниже.

## 1. Запуск системы преобразований

В репозитории [unitree\_h1\_visualization\_ws](https://github.com/cyberbanana777/unitree_h1_visualization_ws) при запуске ***show.launch.py*** реализована система преобразований (трансформ) для робота Unitree H1. Установить репозиторий можно по инструкции в его описании.

### Варианты запуска публикации преобразований

* **Полный запуск с визуализацией в Rviz** (используется для отладки и визуального контроля):

ros2 launch completed\_scripts\_visualization show.launch.py mode:=without\_hands launch\_rviz:=True robot:=simulation

* **Запуск без визуализации:**

ros2 launch completed\_scripts\_visualization show.launch.py mode:=without\_hands robot:=simulation

### Просмотр доступных аргументов launch-файла

Чтобы увидеть все доступные аргументы launch-файла и их значения по умолчанию (что позволяет гибко настроить запуск для разных ситуаций: работа с реальным роботом, симуляция и т.д.), используйте команду:

ros2 launch completed\_scripts\_visualization show.launch.py -s

### Принцип работы системы преобразований

Для публикации преобразований в указанном launch-файле запускается нода:

***robot\_state\_publisher***.

***robot\_state\_publisher*** использует URDF, указанный в параметре ***robot\_description***, и положения суставов из топика ***/joint\_states*** для расчёта прямой кинематики робота (преобразований между системами координат) и публикации результатов через tf.

### Проверка работоспособности преобразований

После запуска любого из вариантов убедитесь, что система преобразований работает корректно:

ros2 node list

В списке активных узлов должен присутствовать узел с названием ***/robot\_state\_publisher***.

ros2 topic list

В списке топиков должны быть топики ***/tf\_static*** и ***/tf***.

## 2. Использование утилиты tf2\_echo

### Формат команды:

ros2 run tf2\_ros tf2\_echo <исходный\_фрейм> <целевой\_фрейм>

### Пример использования:

Чтобы получить преобразование из системы координат таза (***pelvis***) в систему координат правого плеча (***right\_shoulder\_pitch\_link***), выполните:

ros2 run tf2\_ros tf2\_echo pelvis right\_shoulder\_pitch\_link

### Где найти названия фреймов?

* В окне визуализации Rviz (в разделе TF)
* В URDF-описании модели робота

### Интерпретация результата

После запуска команды в консоль начнёт выводиться информация примерно следующего вида:

At time 1761151063.357288585  
- Translation: [0.005, -0.155, 0.430]  
- Rotation: in Quaternion [-0.216, 0.000, 0.000, 0.976]

**Расшифровка данных:**

* **Translation [x, y, z]** — вектор перемещения (в метрах) от исходного фрейма к целевому:
  + **X = 0.005 м**: Правое плечо находится почти прямо по оси X от таза (слегка впереди)
  + **Y = -0.155 м**: Отрицательное значение по оси Y означает, что плечо смещено влево от таза (в стандартной системе координат робота ось Y обычно направлена влево)
  + **Z = 0.430 м**: Положительное значение указывает, что плечо находится значительно выше таза
* **Rotation [x, y, z, w]** — ориентация, представленная в виде кватерниона:
  + В данном примере ненулевые компоненты x и w указывают на поворот в основном вокруг оси X

### Важное замечание

Преобразования между фреймами публикуются постоянно и меняются в реальном времени по мере движения звеньев робота.

**Для получения статичных и корректных данных:** Установите робота (или его симуляционную модель) в нужное положение, дождитесь прекращения движений, и только затем снимите показания с помощью tf2\_echo.

## 3. Матричный метод преобразования координат

### Общая формула преобразования

Формула преобразования точки из системы 1 в систему 2:

**P₂ = R × P₁ + t**

где:

* **P₁** = [x₁, y₁, z₁] - точка в исходной системе координат
* **P₂** = [x₂, y₂, z₂] - точка в целевой системе координат
* **R** - матрица поворота 3×3
* **t** = [tₓ, tᵧ, t₂] - вектор перемещения

### Пример данных

Translation: t = [0.005, -0.155, 0.430]  
Rotation: q = [-0.216, 0.000, 0.000, 0.976] # [x, y, z, w]  
Point: P₁ = [0.1, 0.2, 0.3]

### Преобразование кватерниона в матрицу поворота

#### Формула кватерниона:

q = [x, y, z, w] = [qₓ, qy, qz, qw]

#### Матрица поворота из кватерниона:

[1-2(qy²+qz²) 2(qₓqy-qwqz) 2(qₓqz+qwqy) ]  
R = [2(qₓqy+qwqz) 1-2(qₓ²+qz²) 2(qyqz-qwqₓ) ]  
 [2(qₓqz-qwqy) 2(qyqz+qwqₓ) 1-2(qₓ²+qy²) ]

#### Расчёт для примера:

qₓ = -0.216, qy = 0.000, qz = 0.000, qw = 0.976  
  
 [1.000 0.000 0.000]  
 R = [0.000 0.907 0.422]  
 [0.000 -0.422 0.907]

### Применение поворота

**P\_rotated = R × P₁**

[1.000 0.000 0.000] [0.1] [1.000×0.1 + 0.000×0.2 + 0.000×0.3] [0.100] [0.100]

[0.000 0.907 0.422] × [0.2] = [0.000×0.1 + 0.907×0.2 + 0.422×0.3] = [0.181 + 0.127] = [0.308]

[0.000 -0.422 0.907] [0.3] [0.000×0.1 - 0.422×0.2 + 0.907×0.3] [-0.084 + 0.272] [0.188]

**P\_rotated = [0.1, 0.308, 0.188]**

### Применение вектора перемещения

**P₂ = P\_rotated + t**

P₂ = [0.1, 0.308, 0.188] + [0.005, -0.155, 0.430]  
 P₂ = [0.105, 0.153, 0.618]

## 4. Преобразование координат с помощью ROS2 ноды

### Полный код ноды

#!/usr/bin/env python3  
import rclpy  
from rclpy.node import Node  
from geometry\_msgs.msg import PointStamped  
from tf2\_ros import Buffer, TransformListener  
from tf2\_geometry\_msgs import do\_transform\_point  
  
class RobustTFNode(Node):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_('robust\_tf\_node')  
 self.tf\_buffer = Buffer()  
 self.tf\_listener = TransformListener(self.tf\_buffer, self)  
 self.timer = self.create\_timer(0.5, self.timer\_callback) # 2 Hz  
 self.get\_logger().info("Robust TF node started")  
  
 def timer\_callback(self):  
 try:  
 result = self.convert\_point([0.1, 0.2, 0.3], 'pelvis', 'right\_shoulder\_pitch\_link') # 'pelvis' - система координат таза робота, 'right\_shoulder\_pitch\_link' - система координат плечавого сустава, замените на желаемые после теста  
 if result:  
 self.get\_logger().info(f"Transform: {result}")  
 else:  
 self.get\_logger().warning("TF not available yet, but still running...")  
 except Exception as e:  
 self.get\_logger().error(f"Callback error: {e} - BUT CONTINUING!")  
  
 def convert\_point(self, point, source\_frame, target\_frame):  
 try:  
 if self.tf\_buffer.can\_transform(target\_frame, source\_frame, rclpy.time.Time(), timeout=rclpy.duration.Duration(seconds=1.0)):  
 transform = self.tf\_buffer.lookup\_transform(target\_frame, source\_frame, rclpy.time.Time())  
 point\_msg = PointStamped()  
 point\_msg.header.frame\_id = source\_frame  
 point\_msg.point.x, point\_msg.point.y, point\_msg.point.z = point  
 new\_point = do\_transform\_point(point\_msg, transform)  
 return [new\_point.point.x, new\_point.point.y, new\_point.point.z]  
 return None  
 except:  
 return None  
  
def main():  
 rclpy.init()  
 node = RobustTFNode()  
 rclpy.spin(node)  
 node.destroy\_node()  
 rclpy.shutdown()  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

## Построчное объяснение работы ноды

### **1. Импорты и зависимости**

#!/usr/bin/env python3  
import rclpy  
from rclpy.node import Node  
from geometry\_msgs.msg import PointStamped  
from tf2\_ros import Buffer, TransformListener  
from tf2\_geometry\_msgs import do\_transform\_point

**Что это значит:**

* **rclpy** - основной модуль ROS 2 для Python
* **Node** - базовый класс для создания ROS 2 нод
* **PointStamped** - сообщение для точки с указанием системы координат
* **Buffer** и **TransformListener** - для работы с TF (трансформациями)
* **do\_transform\_point** - функция для применения трансформации к точке

### **2. Создание класса ноды**

class RobustTFNode(Node):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_('robust\_tf\_node')

**Объяснение:**

* **RobustTFNode** - наш собственный класс ноды, наследуется от Node
* **super().\_\_init\_\_('robust\_tf\_node')** - вызывает конструктор родительского класса и задаёт имя ноды
* Имя ноды **'robust\_tf\_node'** будет видно в ROS 2 системе

### **3. Инициализация TF системы**

self.tf\_buffer = Buffer()  
self.tf\_listener = TransformListener(self.tf\_buffer, self)

**Как это работает:**

**Buffer()** - создаёт буфер для хранения трансформаций

**TransformListener()** - слушает топики TF и заполняет буфер

**Аналогия**: Представьте что Buffer - это блокнот, а TransformListener - секретарь, который записывает в него все изменения положений объектов

### **4. Таймер и периодичность**

self.timer = self.create\_timer(0.5, self.timer\_callback) # 2 Hz

**Что происходит:**

**create\_timer(0.5, callback)** - создаёт таймер, который вызывает функцию каждые 0.5 секунд

**2 Hz** - 2 раза в секунду

**timer\_callback** - функция, которая будет выполняться по таймеру

### **5. Основной цикл - timer\_callback**

def timer\_callback(self):  
 try:  
 result = self.convert\_point([0.1, 0.2, 0.3], 'pelvis', 'right\_shoulder\_pitch\_link')  
 if result:  
 self.get\_logger().info(f"Transform: {result}")  
 else:  
 self.get\_logger().warning("TF not available yet, but still running...")  
 except Exception as e:  
 self.get\_logger().error(f"Callback error: {e} - BUT CONTINUING!")

**Пошагово:**

1. **Каждые 0.5 секунд** вызывается эта функция
2. **Пытается преобразовать** точку [0.1, 0.2, 0.3] из системы ***pelvis*** в ***right\_shoulder\_pitch\_link***
3. **Если успешно** - выводит результат
4. **Если нет** - предупреждает, но продолжает работу
5. **При любой ошибке** - пишет в лог, но НЕ останавливается

### **6. Функция преобразования координат**

def convert\_point(self, point, source\_frame, target\_frame):  
 try:  
 if self.tf\_buffer.can\_transform(target\_frame, source\_frame, rclpy.time.Time(), timeout=rclpy.duration.Duration(seconds=1.0)):  
 transform = self.tf\_buffer.lookup\_transform(target\_frame, source\_frame, rclpy.time.Time())  
 point\_msg = PointStamped()  
 point\_msg.header.frame\_id = source\_frame  
 point\_msg.point.x, point\_msg.point.y, point\_msg.point.z = point  
 new\_point = do\_transform\_point(point\_msg, transform)  
 return [new\_point.point.x, new\_point.point.y, new\_point.point.z]  
 return None  
 except:  
 return None

**Детальный разбор:**

#### **Шаг 1: Проверка доступности трансформации**

self.tf\_buffer.can\_transform(target\_frame, source\_frame, rclpy.time.Time(), timeout=1.0)

* **Проверяет**: Можно ли преобразовать из source\_frame в target\_frame?
* **timeout=1.0**: Ждёт до 1 секунды
* **Возвращает**: True если трансформация доступна, False если нет

#### **Шаг 2: Получение трансформации**

transform = self.tf\_buffer.lookup\_transform(target\_frame, source\_frame, rclpy.time.Time())

* **Получает** математическое преобразование между системами координат
* **Важно порядок**: target\_frame, source\_frame - “куда”, “откуда”

#### **Шаг 3: Создание точки для преобразования**

point\_msg = PointStamped()  
point\_msg.header.frame\_id = source\_frame  
point\_msg.point.x, point\_msg.point.y, point\_msg.point.z = point

* ***PointStamped()*** - создаёт сообщение “точка с меткой времени и системой координат”
* ***header.frame\_id*** - указывает в какой системе координат находится точка

#### **Шаг 4: Применение трансформации**

new\_point = do\_transform\_point(point\_msg, transform)

* **Математически применяет** трансформацию к точке
* **Возвращает** ту же точку, но в новой системе координат

## Инструкция по запуску ноды

### 1. Создание ROS2 пакета

Создайте ROS2 пакет ***simple\_tf\_demo*** в папке ***/src***вашей ***\_ws***или откройте папку уже существующего пакета ***<ваш\_пакет>/<ваш\_пакет>***, создайте файл ***simple\_tf\_node.py***.

Подробнее о создании ROS2 пакета и его структуре смотрите в методическом указании [6\_3\_Публикатор\_и\_подписчик](https://github.com/cyberbanana777/unitree_h1_docs/blob/main/instructions_and_manuals/6_3_Публикатор_и_подписчик.md).

### 2. Настройка файла package.xml

После тега <license></license> добавьте:

<depend>rclpy</depend>  
<depend>geometry\_msgs</depend>  
<depend>tf2\_ros</depend>  
<depend>tf2\_geometry\_msgs</depend>

### 3. Настройка файла setup.py

Укажите точку входа (предполагается, что пакет называется ***simple\_tf\_demo***, а файл - ***simple\_tf\_node.py***):

entry\_points={  
 'console\_scripts': [  
 'simple\_tf\_node = simple\_tf\_demo.simple\_tf\_node:main',  
 ],  
},

### 4. Сборка и запуск

сolcon build  
source install/local\_setup.bash

### 5. Запуск в реальном сценарии

# Terminal 1 - Запускаем систему трансформаций (с визуализацией в rviz)  
ros2 launch completed\_scripts\_visualization show.launch.py mode:=without\_hands launch\_rviz:=True robot:=simulation

# Terminal 2 - Запускаем нашу ноду  
ros2 run simple\_tf\_demo simple\_tf\_node

## **Пример работы ноды:**

[INFO] [1761251684.294985257] [continuous\_tf\_node]: Continuous TF node started - will keep running even with errors  
[INFO] [1761251684.390243757] [continuous\_tf\_node]: [1] Transform OK: [0.0945, 0.3769926948296918, 0.032365578089181335]  
[INFO] [1761251686.487127476] [continuous\_tf\_node]: [22] Transform OK: [0.0945, 0.3769926948296918, 0.032365578089181335]  
[INFO] [1761251688.488670306] [continuous\_tf\_node]: [42] Transform OK: [0.0945, 0.3769926948296918, 0.032365578089181335]  
[INFO] [1761251690.587000672] [continuous\_tf\_node]: [63] Transform OK: [0.0945, 0.3769926948296918, 0.032365578089181335]  
^C[INFO] [1761251691.009605015] [continuous\_tf\_node]: Node stopped by user

**Нода будет работать всегда**, пока вы не остановите ее вручную.