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 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:

$\mathbf{P_2 = R \times P_1 + t}$

где:

- $\mathbf{P_1} = [x_1, y_1, z_1]$ - точка в исходной системе координат
- $\mathbf{P_2} = [x_2, y_2, z_2]$ - точка в целевой системе координат
- **R** - матрица поворота 3×3
- $\mathbf{t} = [t_x, t_\gamma, t_2]$ - 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:

$q = [x, y, z, w] = [q_x, q_y, q_z, q_w]$

Матрица поворота из кватерниона:

$$R = \begin{bmatrix} 1-2(q_y^2+q_z^2) & 2(q_xq_y-q_wq_z) & 2(q_xq_z+q_wq_y) \\ 2(q_xq_y+q_wq_z) & 1-2(q_x^2+q_z^2) & 2(q_yq_z-q_wq_x) \\ 2(q_xq_z-q_wq_y) & 2(q_yq_z+q_wq_x) & 1-2(q_x^2+q_y^2) \end{bmatrix}$$

Расчёт для примера:

$q_x = -0.216$, $q_y = 0.000$, $q_z = 0.000$, $q_w = 0.976$

$$R = \begin{bmatrix} 1.000 & 0.000 & 0.000 \\ 0.000 & 0.907 & 0.422 \\ 0.000 & -0.422 & 0.907 \end{bmatrix}$$

Применение поворота

$P\_rotated = R \times P_1$

$$\begin{bmatrix} 1.000 & 0.000 & 0.000 \\ 0.000 & 0.907 & 0.422 \\ 0.000 & -0.422 & 0.907 \end{bmatrix} \times \begin{bmatrix} 0.1 \\ 0.2 \\ 0.3 \end{bmatrix} = \begin{bmatrix} 1.000\times0.1 + 0.000\times0.2 + 0.000\times0.3 \\ 0.000\times0.1 + 0.907\times0.2 + 0.422\times0.3 \\ 0.000\times0.1 - 0.422\times0.2 + 0.907\times0.3 \end{bmatrix} = \begin{bmatrix} 0.100 \\ 0.181 + 0.127 \\ -0.084 + 0.272 \end{bmatrix} = \begin{bmatrix} 0.100 \\ 0.308 \\ 0.188 \end{bmatrix}$$

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

Применение вектора перемещения

$P_2 = P\_rotated + t$

$P_2 = [0.1, 0.308, 0.188] + [0.005, -0.155, 0.430]$

$P_2 = [0.105, 0.153, 0.618]$

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_s
houlder_pitch_link') # 'pelvis' - система координат таза робота, 'right_sho
ulder_pitch_link' - система координат плечавого сустава, замените на желаем
ые 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, rcl
py.time.Time(), timeout=rclpy.duration.Duration(seconds=1.0)):
                transform = self.tf_buffer.lookup_transform(target_frame, s
ource_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 = p
oint
                new_point = do_transform_point(point_msg, transform)
```

```
                return [new_point.point.x, new_point.point.y, new_point.poi
nt.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_shoul
der_pitch_link')
        if result:
            self.get_logger().info(f"Transform: {result}")
        else:
            self.get_logger().warning("TF not available yet, but still runn
ing...")
    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.t
ime.Time(), timeout=rclpy.duration.Duration(seconds=1.0)):
```

```
            transform = self.tf_buffer.lookup_transform(target_frame, sourc
e_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, rcl
py.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.

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

```
colcon 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_ha
nds 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 star
ted - will keep running even with errors
[INFO] [1761251684.390243757] [continuous_tf_node]: [1] Transform OK: [0.09
45, 0.3769926948296918, 0.032365578089181335]
[INFO] [1761251686.487127476] [continuous_tf_node]: [22] Transform OK: [0.0
945, 0.3769926948296918, 0.032365578089181335]
[INFO] [1761251688.488670306] [continuous_tf_node]: [42] Transform OK: [0.0
945, 0.3769926948296918, 0.032365578089181335]
[INFO] [1761251690.587000672] [continuous_tf_node]: [63] Transform OK: [0.0
945, 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.