LEG ACTUATIONS USING ROS

Single Leg Actuation Using ROS + Arduino + PCA9685

Objective

To control hip, knee, and ankle servos of a single quadruped leg using:

- ROS Noetic on Raspberry Pi (publisher)
- Arduino Nano with PCA9685 (subscriber & actuator)
- ROS topics for sending joint angles

Hardware Used

- Raspberry Pi: Runs ROS nodes and sends joint angles
- Arduino Nano: Receives angles from ROS and drives servos
- PCA9685: 16-channel PWM driver for controlling servos
- Servo Motors: Physical actuators for hip, knee, ankle
- USB cable: Serial communication between Pi and Arduino
- I2C wires: Communication between Arduino and PCA9685

System Overview and Code Descriptions

Interaction Flow

[ROS Publisher Node – Python]

↓ (topics: /fr_hip_angle, /fr_knee_angle, /fr_ankle_angle)

[rosserial_python → /dev/ttyUSB0]

↓

[Arduino Nano]

↓ (I2C)

[PCA9685 Driver]

↓

[3 Servo Motors: Hip, Knee, Ankle]

1. move_leg_publisher.py (ROS Python Publisher Node)

- **Purpose**: Publishes servo angles as individual Float64 values to 3 topics: /fr_hip_angle, /fr_knee_angle, /fr_ankle_angle.
- **Functionality**: Alternates between two sets of servo angles:
    - **Step Position**: Hip = 110°, Knee = 60°, Ankle = 100°
    - **Stand Position**: Hip = 90°, Knee = 120°, Ankle = 90°
- **Role in System**: Simulates basic gait motion by sending servo angles every 2 seconds. Acts as the control logic for the leg movement.

2. rosserial_python (ROS ↔ Arduino Communication Bridge)

- **Purpose**: Transfers ROS topic data to the Arduino Nano over serial.
- **Functionality**: Uses:**rosrun rosserial_python serial_node.py _port:=/dev/ttyUSB0 _baud:=57600** to open a serial connection between ROS and Arduino.
- **Why Needed**: Since the Arduino cannot run ROS natively, this bridge converts ROS messages into serial data understood by Arduino.
- **Role in System**: Acts as a translator/relay, enabling the Python ROS node to control hardware.

3. Arduino Nano Sketch

- **Purpose**: Receives float values for hip, knee, and ankle angles via serial (using rosserial) and drives corresponding servos.
- **Functionality**:
    - Subscribes to /fr_hip_angle, /fr_knee_angle, and /fr_ankle_angle

    - Converts each angle into a PWM signal using angleToPulse()
    - Sends the PWM signal via I2C to PCA9685 for servo control
- **Why Needed**: The Arduino:
    - Handles real-time servo actuation
    - Communicates with the PCA9685 over I2C
    - Interprets the exact float angles coming from ROS topics
- **Why Not Directly from Pi?**
    - Raspberry Pi can use I2C, but lacks real-time timing precision.
    - Arduino ensures smoother PWM output and hardware-level control.
    - Separates ROS logic from low-level actuation for modularity and safety.

1.Creating Workspace

bash

mkdir -p ~/catkin_ws/src

Create the catkin workspace directory with a src folder

cd ~/catkin_ws

Navigate into the workspace root directory

catkin_make

Build the workspace using catkin (compiles all packages in src)

source devel/setup.bash

Source the workspace so ROS can find your packages

2.Create ROS Package

bash

cd ~/catkin_ws/src

Go to the 'src' folder inside your catkin workspace

catkin_create_pkg servo_publisher std_msgs rospy

Create a new ROS package named 'servo_publisher'.It depends on 'std_msgs' and 'rospy' libraries

cd ~/catkin_ws

#Go back to the root of your workspace

catkin_make

Build your workspace again to include the new package

source devel/setup.bash

Source the workspace to make the new package available to ROS

3.Create Python script

bash

cd ~/catkin_ws/src/servo_publisher

Navigate into your new ROS package folder

mkdir scripts

Create a 'scripts' directory to store Python nodes

cd scripts

Enter the 'scripts' folder

touch move_leg_publisher.py

Create a new empty Python file for your publisher node

ROS Python Publisher – `move_leg_publisher.py`

Paste the below code in the move_leg_publisher.py in VSCode

```
#!/usr/bin/env python3
# Specifies this is a Python 3 ROS executable script

import rospy                                    # Import rospy to use ROS functionalities
from std_msgs.msg import Float64          # Import message type for sending float values

# Main function to control the leg
def move_leg():
    # Create publishers to publish servo angles on specific topics
    pub_hip = rospy.Publisher('/fr_hip_angle', Float64, queue_size=10)
    pub_knee = rospy.Publisher('/fr_knee_angle', Float64, queue_size=10)
    pub_ankle = rospy.Publisher('/fr_ankle_angle', Float64, queue_size=10)

    rospy.init_node('move_leg_publisher', anonymous=True)  # Initialize ROS node
    rate = rospy.Rate(0.5)                 # Set publish rate to 0.5 Hz (once every 2 seconds)

    # Define servo angles for stand position
    HIP_STAND = 90
    KNEE_STAND = 120
    ANKLE_STAND = 90

    # Define servo angles for step/lift position
    HIP_FORWARD = 110
    KNEE_LIFT = 60
    ANKLE_ADJUST = 100

    state = 0                                       # Used to toggle between step and stand

    while not rospy.is_shutdown():                          # Loop until ROS is shutdown
        if state == 0:
            rospy.loginfo("Step 1: Move leg forward")          # Print info on console
            pub_hip.publish(Float64(HIP_FORWARD))          # Publish hip angle
            pub_knee.publish(Float64(KNEE_LIFT))           # Publish knee angle
            pub_ankle.publish(Float64(ANKLE_ADJUST))       # Publish ankle angle
        else:
            rospy.loginfo("Step 2: Return to stand")           # Print info on console
            pub_hip.publish(Float64(HIP_STAND))            # Publish hip angle
```

```
            pub_knee.publish(Float64(KNEE_STAND))          # Publish knee angle
            pub_ankle.publish(Float64(ANKLE_STAND))        # Publish ankle angle

        state = (state + 1) % 2                            # Toggle state between 0 and 1
        rate.sleep()                                       # Wait for next cycle

# Main program entry point
if __name__ == '__main__':
    try:
        move_leg()                                         # Run the function
    except rospy.ROSInterruptException:
        pass                                               # Ignore if program is interrupted
```

bash:
chmod +x move_leg_publisher.py
Make the script executable so ROS can run it

Arduino Code

As Arduino acts as the subscriber,upload this code via Arduino IDE.

```
#include <Wire.h>
// I2C communication for PCA9685
#include <Adafruit_PWMServoDriver.h>
// Adafruit library to control PCA9685
#include <ros.h>
// ROS client library for Arduino
#include <std_msgs/Float64.h>
// Message type for receiving float angles

Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver();
// Create servo driver object
ros::NodeHandle nh;
// Create ROS node handle

#define SERVO_MIN 150                  // Minimum PWM pulse length
#define SERVO_MAX 600                  // Maximum PWM pulse length
```

```
// Function to convert angle (0–180) to PWM value (150–600)
int angleToPulse(int angle) {
  return map(angle, 0, 180, SERVO_MIN, SERVO_MAX);
}

// Function to set servo angle for a given PCA9685 channel
void setServoAngle(uint8_t channel, int angle) {
  pwm.setPWM(channel, 0, angleToPulse(angle));  // Set PWM pulse on that channel
}

// Callback for hip angle messages from ROS
void hipCallback(const std_msgs::Float64 &msg) {
  setServoAngle(0, (int)msg.data);  // Channel 0 - Hip servo
}

// Callback for knee angle messages from ROS
void kneeCallback(const std_msgs::Float64 &msg) {
  setServoAngle(1, (int)msg.data);  // Channel 1 - Knee servo
}

// Callback for ankle angle messages from ROS
void ankleCallback(const std_msgs::Float64 &msg) {
  setServoAngle(2, (int)msg.data);  // Channel 2 - Ankle servo
}

// Subscribers for each joint's topic
ros::Subscriber<std_msgs::Float64> sub_hip("/fr_hip_angle", hipCallback);
ros::Subscriber<std_msgs::Float64> sub_knee("/fr_knee_angle", kneeCallback);
ros::Subscriber<std_msgs::Float64> sub_ankle("/fr_ankle_angle", ankleCallback);

void setup() {
  pwm.begin();                                    // Start PCA9685 driver
  pwm.setPWMFreq(50);                             // Set frequency to 50 Hz for servos
  nh.initNode();                                  // Initialize ROS node on Arduino
  nh.subscribe(sub_hip);                          // Subscribe to hip topic
  nh.subscribe(sub_knee);                         // Subscribe to knee topic
  nh.subscribe(sub_ankle);                        // Subscribe to ankle topic
}
```

```
void loop() {
  nh.spinOnce();                                  // Check for incoming ROS messages
  delay(10);                                      // Short delay to avoid overloading
}
```

Running the System

Build and source workspace

```
cd ~/catkin_ws
catkin_make                    # Compile workspace
source devel/setup.bash        # Load ROS environment
```

Open 3 terminals:

Terminal 1: Start ROS Core

bash
roscore

#Starts the master node, required for all other ROS communication.

Output in terminal:

```
... logging to ~/.ros/log/xxxx.log
Checking log directory for disk usage. This may take a while.
Press Ctrl+C to interrupt
Started roslaunch server http://raspberrypi:xxxxx/

SUMMARY
========

PARAMETERS
 * /rosdistro: noetic
 * /rosversion: 1.15.x

NODES

auto-starting new master
process[master]: started with pid [xxxx]
ROS Master is running at http://localhost:11311/
```

Terminal 2: Start Arduino Serial Node

bash
source ~/catkin_ws/devel/setup.bash
rosrun rosserial_python serial_node.py _port:=/dev/ttyUSB0 _baud:=57600

#This connects Arduino to ROS and allows it to receive topic messages.

Output in terminal:

[INFO] [1688099600.123456]: ROS Serial Python Node

[INFO] [1688099600.123789]: Connecting to /dev/ttyUSB0 at 57600 baud

[INFO] [1688099602.456789]: Note: subscribe buffer size = 512

[INFO] [1688099602.456999]: Setup complete.

3.Terminal 3: Run the Python Publisher Node

bash
source ~/catkin_ws/devel/setup.bash
rosrun servo_publisher move_leg_publisher.py

#Publishes angles to hip, knee, and ankle servos

Output in terminal:.

[INFO] [1688099610.112233]: Step 1: Move leg forward

[INFO] [1688099612.114455]: Step 2: Return to stand

[INFO] [1688099614.118899]: Step 1: Move leg forward

[INFO] [1688099616.123001]: Step 2: Return to stand

[INFO] [1688099618.126777]: Step 1: Move leg forward

[INFO] [1688099620.129000]: Step 2: Return to stand

…

Final Output of This Setup

Servo Movement Behavior

The 3 servos (connected to PCA9685 on channels 0, 1, and 2) will move like this every 2 seconds in a repeating loop:

Step 1: LIFT + FORWARD
• Hip Servo (Ch 0): moves from 90° to 110° → pushes leg forward
• Knee Servo (Ch 1): moves from 120° to 60° → lifts the leg
• Ankle Servo (Ch 2): moves from 90° to 100° → adjusts foot tilt

After 2 seconds:

Step 2: STAND POSITION
• Hip Servo (Ch 0): moves back to 90°
• Knee Servo (Ch 1): moves back to 120°
• Ankle Servo (Ch 2): moves back to 90°

This cycle repeats every 2 seconds, simulating a simple lift-forward-stand stepping motion for one leg.

Video: dingo leg actuation.mp4

Two Leg Actuation-ROS to Arduino Servo Control

Objective:

Control four servo motors via Arduino Nano using commands published from a ROS node running on Raspberry Pi.

System Overview and Code Descriptions

Interaction Flow

[ROS Publisher Node] → (topic: /servo_angles) → [ROS Subscriber Node (Python)] → (Serial /dev/ttyUSB0) → [Arduino Nano] → [PCA9685] → Servo Motors

1. publish_walking_steps.py (ROS Publisher Node)

- **Purpose:** Publishes servo angle strings like "60,100,120,90" to the /servo_angles topic.
- **Functionality:** Cycles through a list of predefined servo movements simulating leg motion.
- **Role in system:** Generates movement commands that would typically come from joystick, keyboard, or autonomous control.

2. servo_controller_node.py (ROS Subscriber Node on Raspberry Pi)

- **Purpose:** Subscribes to the /servo_angles topic and forwards received data to the Arduino over serial.
- **Functionality:** Converts ROS message into plain text format and writes it via USB.
- **Why Needed:** Arduino can't run ROS directly. This node acts as a bridge, allowing ROS to communicate with the Arduino.

3. Arduino Nano Code

- **Purpose:** Parses angle strings from serial input and commands the PCA9685 to move servos.
- **Functionality:** Waits for complete string (e.g., "60,100,120,90\n"), splits it into integers, and maps each angle to servo pulse widths.
- **Why Needed:** The Arduino communicates with PCA9685 via I2C and translates ROS data into PWM signals.
- **Why Not Direct from Pi?:** While it's possible to use I2C on Raspberry Pi, Arduino offers precise, real-time control and ensures hardware isolation.

Step-by-Step Setup

1. Create Catkin Workspace

```
mkdir -p ~/catkin_ws/src
 # Create workspace directory
cd ~/catkin_ws/src
#  Navigate into the workspace root directory
catkin_init_workspace
# Initialize catkin workspace
cd ..
catkin_make
# Build the workspace
source devel/setup.bash
 # Source the environment
```

2. Create ROS Package

```
cd ~/catkin_ws/src
```

Navigate into the workspace root directory

catkin_create_pkg servo_comm std_msgs rospy

Create ROS package with dependencies

cd ~/catkin_ws

#Go back to the root of your workspace

catkin_make

Build your workspace again to include the new package

source devel/setup.bash

Source the workspace to make the new package available to ROS

3.Create Python script

bash

cd ~/catkin_ws/src/servo_comm

Navigate into your new ROS package folder

mkdir scripts

Create a 'scripts' directory to store Python nodes

cd scripts

Enter the 'scripts' folder

touch servo_controller_node.py

Create a new empty Python file for Subscriber node

Servo_controller_node.py - Subscriber (on Raspberry Pi)

Paste the below code in the Servo_controller_node.py in VSCode

code:

```
#!/usr/bin/env python3
# Specifies this is a Python 3 ROS executable script
import rospy                                    # ROS Python module
import serial                                   # For serial communication with Arduino
from std_msgs.msg import String                 # ROS message type
import time

class ServoController:
    def __init__(self):
        # Open serial port to communicate with Arduino
        self.port = serial.Serial('/dev/ttyUSB0', 9600, timeout=1)
        time.sleep(2)                           # Wait for Arduino reset after serial connection

        rospy.init_node('servo_controller_node')        # Initialize ROS node
        # Subscribe to topic where angles will be published
        rospy.Subscriber("/servo_angles", String, self.callback)
        rospy.loginfo(" Ready to receive servo angles from ROS.")
        rospy.spin()                                    # Keep the node running

    def callback(self, msg):
        angles = msg.data.strip() + '\n'                # Ensure newline for Arduino parsing
        self.port.write(angles.encode())                # Send angle string to Arduino
        rospy.loginfo(f"Sent to Arduino: {angles.strip()}")

if __name__ == "__main__":                              #Main function
    ServoController()
```

bash:

chmod +x Servo Controller_node.py

Make the script executable so ROS can run it

bash
touch publish_walking_steps.py

Create a new empty Python file for Publisher node

publish_walking_steps.py - Publisher (on Raspberry Pi)

Paste the below code in the publish_walking_steps.py in VSCode,

code:

```
#!/usr/bin/env python3
# Specifies this is a Python 3 ROS executable script
import rospy                                                    # ROS Python module
from std_msgs.msg import String                                 # ROS message type

def main():
    rospy.init_node('walking_step_publisher')             # Initialize ROS node
    pub = rospy.Publisher('/servo_angles', String, queue_size=10)
# Create publisher
    rate = rospy.Rate(0.5)
# Delay between messages (2 seconds)

 # List of walking step commands
    steps = [
        "60,100,120,90",   # Lift right leg
        "120,90,120,90",   # Stand
        "120,90,60,100",   # Lift left leg
        "120,90,120,90"    # Stand
    ]

    i = 0
    while not rospy.is_shutdown():
        angle_str = steps[i % len(steps)]                        # Cycle through steps
        pub.publish(angle_str)                                  # Publish to ROS topic
        rospy.loginfo(f" Published: {angle_str}")
        i += 1
        rate.sleep()                                             # Wait before next step
```

```
if __name__ == '__main__':                                  #Main function
    main()
```

bash:
chmod +x publish_walkin_steps.py
Make the script executable so ROS can run it

Arduino Code (Uploaded to Nano) Via Arduino IDE

```
#include <Wire.h>                                  // I2C communication
#include <Adafruit_PWMServoDriver.h>          // PCA9685 Servo driver library

Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver(); // Servo driver object

#define SERVO_MIN 150                              // Min pulse
#define SERVO_MAX 600                              // Max pulse

// Convert angle (0-180) to PWM pulse
int angleToPulse(int angle) {
  return map(angle, 0, 180, SERVO_MIN, SERVO_MAX);
}

// Set PWM pulse to specific channel
void setServo(uint8_t ch, int angle) {
  angle = constrain(angle, 0, 180);                  // Keep angle in range
  pwm.setPWM(ch, 0, angleToPulse(angle));    // Send pulse to servo
}

void setup() {
  Serial.begin(9600);                                 // Start serial for ROS communication
  pwm.begin();                                       // Initialize PCA9685
  pwm.setPWMFreq(50);                               // Set frequency to 50Hz for servos

  // Initial standing position
  setServo(0, 120);
  setServo(1, 90);
  setServo(2, 120);
  setServo(3, 90);
```

```
  Serial.println("Arduino ready");
}

void loop() {
  static String input = "";

  while (Serial.available()) {
    char c = Serial.read();                          // Read one char
    if (c == '\n') {                                 // Full command received
      Serial.print("Received: ");
      Serial.println(input);

      int angles[4] = {0};
      int index = 0;

      // Replace commas with spaces to tokenize
      for (int i = 0; i < input.length(); i++) {
        if (input[i] == ',') input[i] = ' ';
      }

      char buffer[32];
      input.toCharArray(buffer, 32);                 // Convert to char array
      char* part = strtok(buffer, " ");              // Tokenize
      while (part != NULL && index < 4) {
        angles[index++] = atoi(part);
        part = strtok(NULL, " ");
      }

      if (index == 4) {
        // Send angles to each servo
        setServo(0, angles[0]);
        setServo(1, angles[1]);
        setServo(2, angles[2]);
        setServo(3, angles[3]);
        Serial.println("Servos moved.");
      } else {
        Serial.println("Error: Not enough angles received.");
      }
```

```
      input = "";
    } else {
      input += c;                                  // Keep building input string
    }
  }
}
```

Running the ROS System

Build and source workspace

cd ~/catkin_ws
catkin_make # Compile workspace
source devel/setup.bash # Load ROS environment

Open 3 terminals:

Terminal 1:

roscore # Start ROS core

Output in terminal:
... logging to /home/your_user/.ros/log/...
started core service [/rosout]

Terminal 2:
rosrun servo_comm servo_controller_node.py # Start subscriber node

Output in terminal:
Ready to receive servo angles from ROS.
Sent to Arduino: 60,100,120,90
Sent to Arduino: 120,90,120,90
Sent to Arduino: 120,90,60,100
Sent to Arduino: 120,90,120,90
...

Terminal 3:
rosrun servo_comm publish_walking_steps.py # Start publisher node

Output in terminal:

Published: 60,100,120,90
Published: 120,90,120,90
Published: 120,90,60,100
Published: 120,90,120,90
…….

Expected Output on Robot

Your **four servos** will perform this repeating motion every 2 seconds:

1. **Step 1**: Right leg lifts
    - RIGHT_KNEE → 60
    - RIGHT_ANKLE → 100
    - Left leg stands
2. **Step 2**: Stand
    - All servos to standing angle: KNEE = 120, ANKLE = 90
3. **Step 3**: Left leg lifts
    - LEFT_KNEE → 60
    - LEFT_ANKLE → 100
    - Right leg stands
4. **Step 4**: Stand
    - All servos back to standing

Although the setup is complete, the expected servo motion output is not observed. Therefore, further testing and debugging will focus on ensuring correct angle transmission and successful servo actuation based on received commands.