Cole Fenner
6/8/2025

Odrive ROS2 CAN Package & HTTP Integration for A Projectile Motion System

Final Documentation

Description of Project:

This project will connect our two Odrive motors and ESP32 all together with ROS2 using ODrive ROS2 CAN Package and HTTP Communication. This will allow higher level control with service call control and simple velocity command inputs.

Motivation: Control my Master's Thesis Arm:

In order to throw an object with exact spin and velocity, you will need a robotic system with two axis of control and a release mechanism. This is the basis for my masters project which requires software to control the arm for future graduate students.

Results:

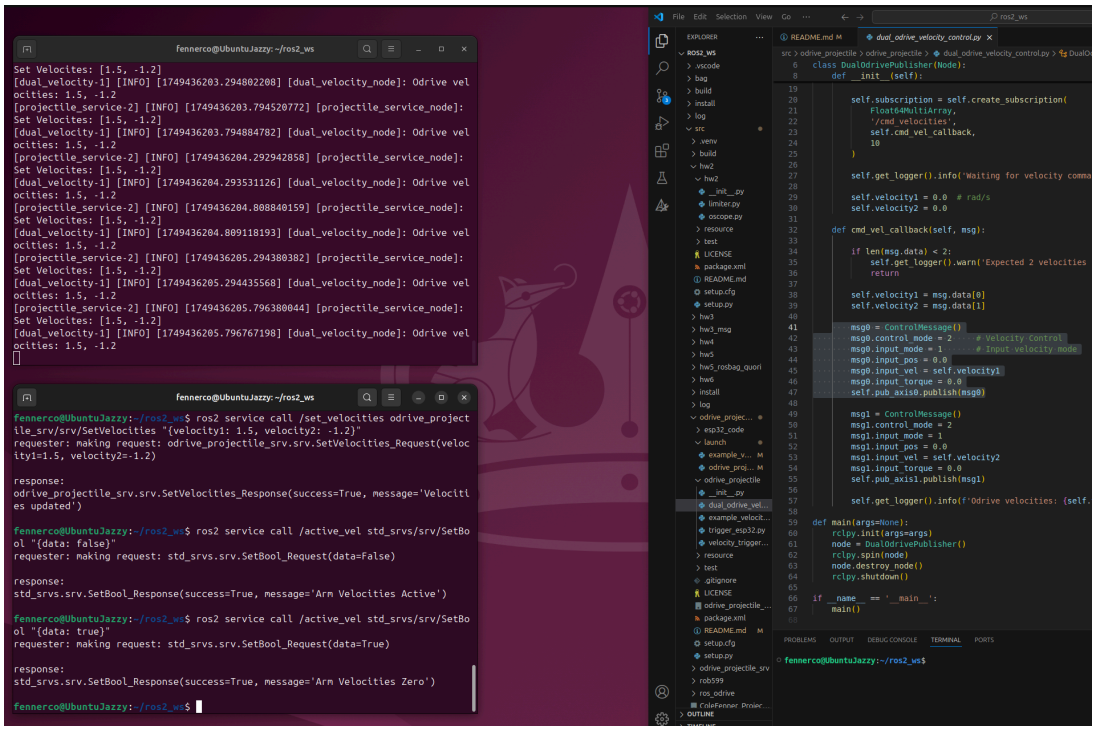

Figure 1: ROS2 Workspace showcasing velocity control through service calls of arm.

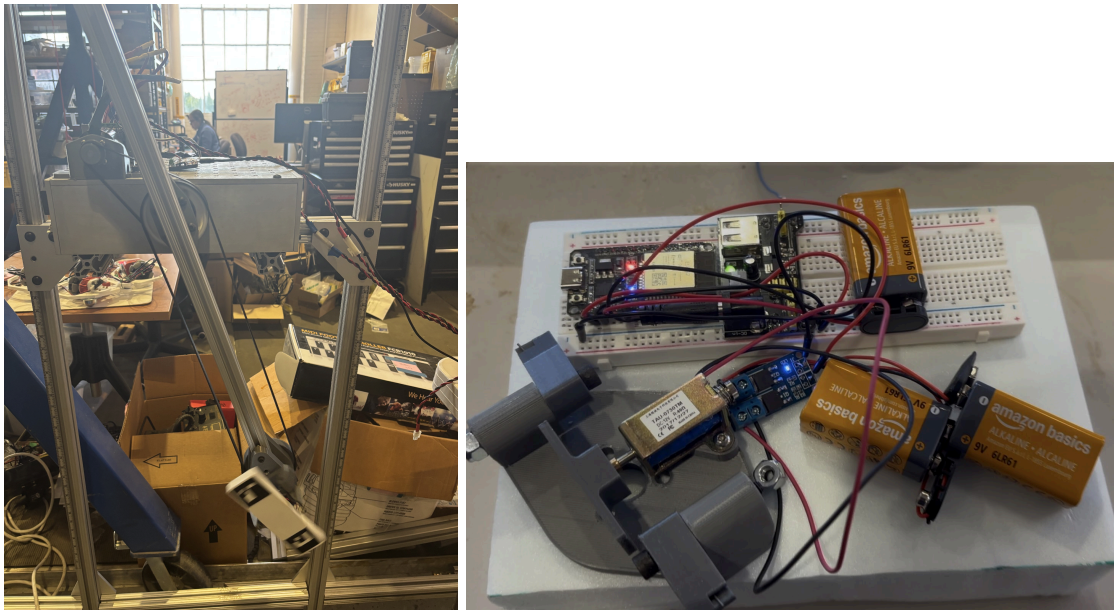

Figure 2: Working hardware of Odrive spinning & ESP32 HTTP control.

As Shown in figure 1, we have ROS2 control over our hardware system in figure 2. This can be controlled with a combined velocity message or service controlled for an abstraction for future robotic students.

Challenges and Successes:

Some challenges included in building up the Odrive ROS software as some packages were broken like the ros2_control. Though were able to interface with their simpler ROS2 CAN package. Attempted to integrate Micro_ROS instead of HTTP communication with the ESP32, yet the building overhead and delay it would add to a simple trigger is not worth it.

Lastly other system architecture can be more real time, yet ROS2 delay is fine for the given project and tests.

Learned on the Project:

Things learned on this project mostly included the software integration of the Odrive modules and getting them working. In addition this software let me learn about changing Namespaces in launch files to work with setting up specific controller nodes.

Lastly I learned alot about micro_ROS and HTTP communication with the ESP32 microcontroller.

Documentation:

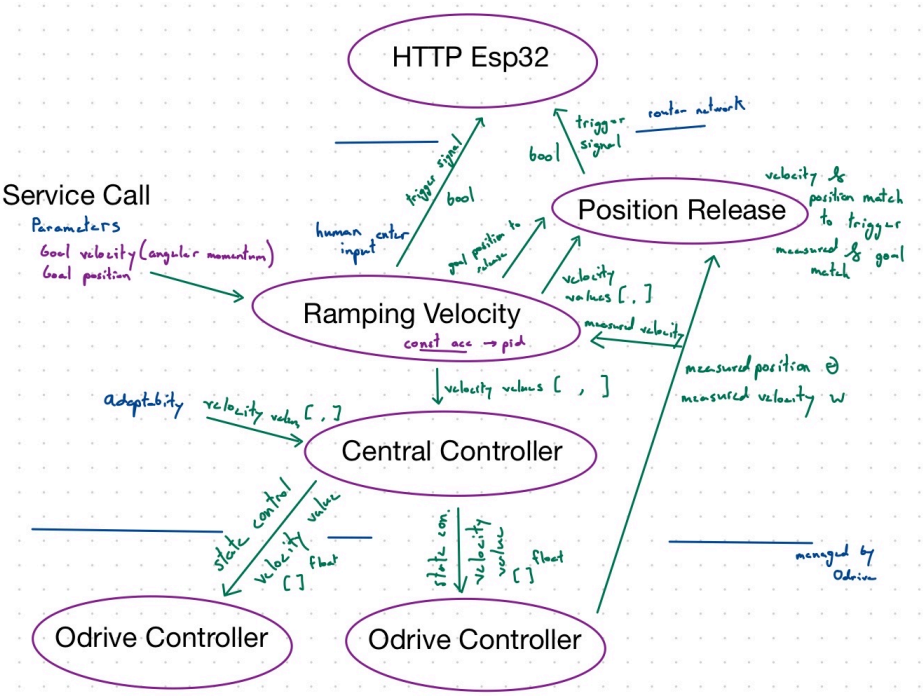

Figure 3: Node Graph of Odrive System

Description of the System:

As shown in figure 3, we see the node architecture of the system. With the ODrive nodes running for each controller, we have a central controller node that translates a grouped velocity message to ODrive specific state control messages. We also have our ramping velocity node which takes in service call actions to stop, start, and set velocities given to our arm. In addition we have a service call set up to send a trigger signal over HTTP to our ESP 32. Lastly we have implemented example velocity over time nodes to input to our system to enable less friction in future research.

Code Instructions

Get the Code from the github:

https://github.com/ColeFenner/ODrive-2DOFProjectile-Arm

Required Package:

ros_odrive: https://github.com/odriverobotics/ros_odrive

Software Setup:

Clone this repository and the ros_odrive package into your workspace

For more information in general Odrive ROS2 CAN control, refer to official documentation: https://docs.odriverobotics.com/v/latest/guides/ros-package.html

1. As the odrive_ros2_control package in ros_drive does not build properly with this ROS2 version, we build what is required:

   colcon build --packages-select odrive_can odrive_projectile odrive_projectile_srv
   source install/setup.bash

2. Lauch Up the Velocity Controller & Service Manager:

   ros2 launch odrive_projectile odrive_projectile_launch.py

3. Set up ramping velocity control on both axis:

ros2 service call /odrive_axis0/request_axis_state odrive_can/srv/AxisState "{axis_requested_state: 8}"

ros2 service call /odrive_axis1/request_axis_state odrive_can/srv/AxisState "{axis_requested_state: 8}"

4. Check Service Calls Interact With Velocity Control:

5. Set the target velocities the arm will swing too

ros2 service call /set_velocities odrive_projectile_srv/srv/SetVelocities "{velocity1: 1.5, velocity2: -1.2}"

6. Switch from [0,0] Velocity to Target Velocity:

   ros2 service call /active_vel std_srvs/srv/SetBool "{data: true}"
   ros2 service call /active_vel std_srvs/srv/SetBool "{data: false}"

7. Send a release toggle signal:
   ros2 service call /release std_srvs/srv/SetBool

8. Service Call throwing sequence of the combined actions above:

   ros2 service call /run_velocity_sequence std_srvs/srv/Trigger

9. To Run the example v(t) velocity input, launch up another file for sin velocity motion being called to both motors:

   ros2 launch odrive_projectile example_velocity_launch.py

   ros2 service call /odrive_axis0/request_axis_state odrive_can/srv/AxisState "{axis_requested_state: 8}"
   ros2 service call /odrive_axis1/request_axis_state odrive_can/srv/AxisState "{axis_requested_state: 8}"