ENG 8905 - Quadrature Encoders Part 1 The Knob

Kyle Chisholm

2022-05-01

Decoding a quadrature incremental encoder to measure position of a knob. A custom implementation of the decoding logic is compared against a published Python package.

Submission Deadline: May 01, 2022

Online version of this document: https://chishok.github.io/ENG8905/ENG8905-Lab7

Background

Incremental encoders are everywhere, used for high-precision motion control applications such as robotics or radar systems and also for simple interfaces like volume knobs. A rotary encoder repeats a signal sequentially "high" then "low" as it rotates. The signals are generated by various physical means, including magnetic, optical, and mechanical interfaces.

An **optical encoder** involves placing a light emitter and a received on either sides of an opaque disc, with open slots cut out at regular angle intervals. When the disc rotates, the receiver will register the light in a sequential on and off pattern as shown in Figure 1.

The rising and/or falling edges of the one/off square wave pattern are counted by a **decoder**, which can be implemented as a simple integrated circuit or part of microcontroller code. In this lab we will be using the Raspberry Pi GPIO and Python to count encoder signal edges.

A **mechanical encoder** uses physical conductive contacts that brush up one another to generate the on and off (high and low) signals as it rotates. This lab will utilize a mechanical encoder KY-040 shown in Figure 2 .

Compared to optical encoders, mechanical encoders are much cheaper but also more prone to noise associated with the mechanical interface. We will see the effect of mechanical noise on the ability to capture the knob position as it's rotated.

Magnetometers or hall effect sensors and rotating magnetic fields are also used for rotational sensor encoding, especially in 3-phase DC motor commutation. Encoded magnetic strips are also used in linear sensing applications. This lab will be focused on a low-cost mechanical rotary encoder and the next part will use a hall effect rotary encoder on a DC motor.

Quadrature and Absolute Encoders

The disadvantage of using an incremental encoder is that the pulsing provided by the sensor in one direction is indistinguishable from motion in the opposite direction. Single channel incremental encoders are more useful for applications with one direction of motion.

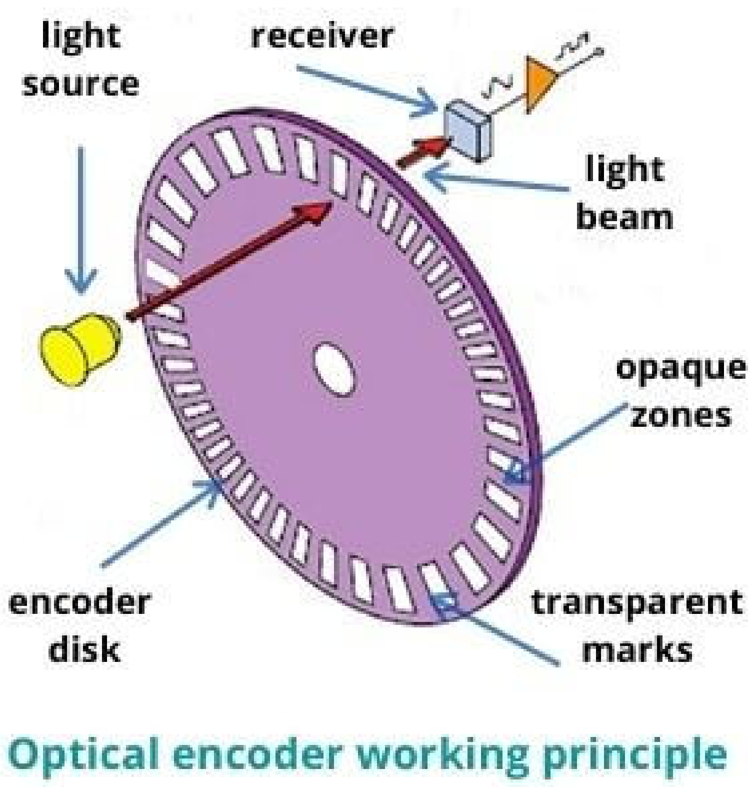


Figure 1: Optical Encoder

For position feedback and determining the direction of rotation, a **quadrature encoder** solves the problem by adding a row of on/off signals in the physical interface with an offset such that the direction can be determined. Refer to Figure 3 which shows the operation of a quadrature encoder and the resulting two signals A and B that follow from a clockwise rotation. Figure 4 shows the resulting signal from rotation in clockwise and counter-clockwise directions.

Gray-Code Decoding

When the A and B signals arrive from a quadrature encoder, we need to interpret the sequence of high and low signals from both lines to generate a change in value. This is the decoding process.

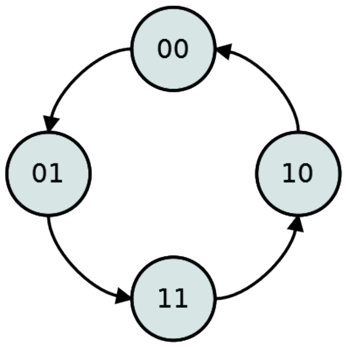
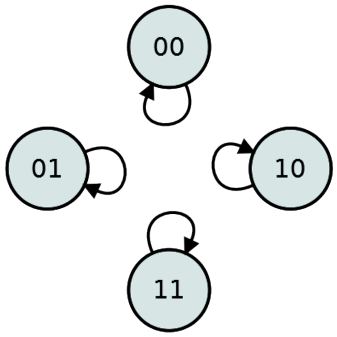
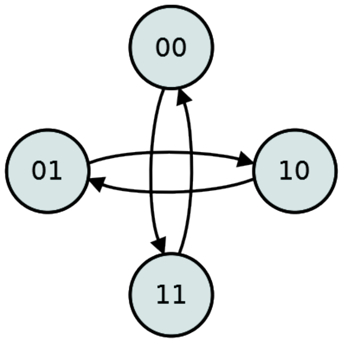
It is convenient to interpret the two signals in an incremental quadrature encoder as a 2-bit code. The first least significant bit (LSB: the right-most bit in most programming languages) signifies the B value as either ON (1) or OFF (0). The next bit contains the value of the A signal. We can therefore assign a change in value according to the sequence of bits that are received from the encoder.

The progression of the 2-bit code is as follows:

Direction	Sequence
CW	$00 \to 01 \to 11 \to 10$
CCW	$00 \to 10 \to 11 \to 01$

From these state transitions, given the current state of the A and B inputs, and the next state when either signal changes, we can compute the increment of pulse count in positive or negative direction. With only one bit changing at a time, this sequence is referred to as a 2-bit **Gray Code**.

However, this sequence is only valid if one bit changes. If two bits change at the same time, it is

Transitions	Diagram
Reverse Movement (-1)	
No Movement (0)	
Invalid Motion (±2)	

The number of edges counted (state transitions) per each revolution of the rotary encoder provides the resolution of the encoder in **Counts Per Revolution (CPR)**. A higher resolution may provide better precision of an angle measurement, but if the encoder is spinning too fast, the frequency of the pulses that need to be counted per second increases and may exceed the capabilities of the decoder (counter) and counts can be missed.

In the lab with the motion control application (part 2), we will investigate the relationship between the motor RPM and pulse frequency of the encoder. They are important factors in designing a feedback control system using rotary encoders. For this lab, we not as concerned with keeping an accurate absolute position count and missed steps are tolerable.

Materials

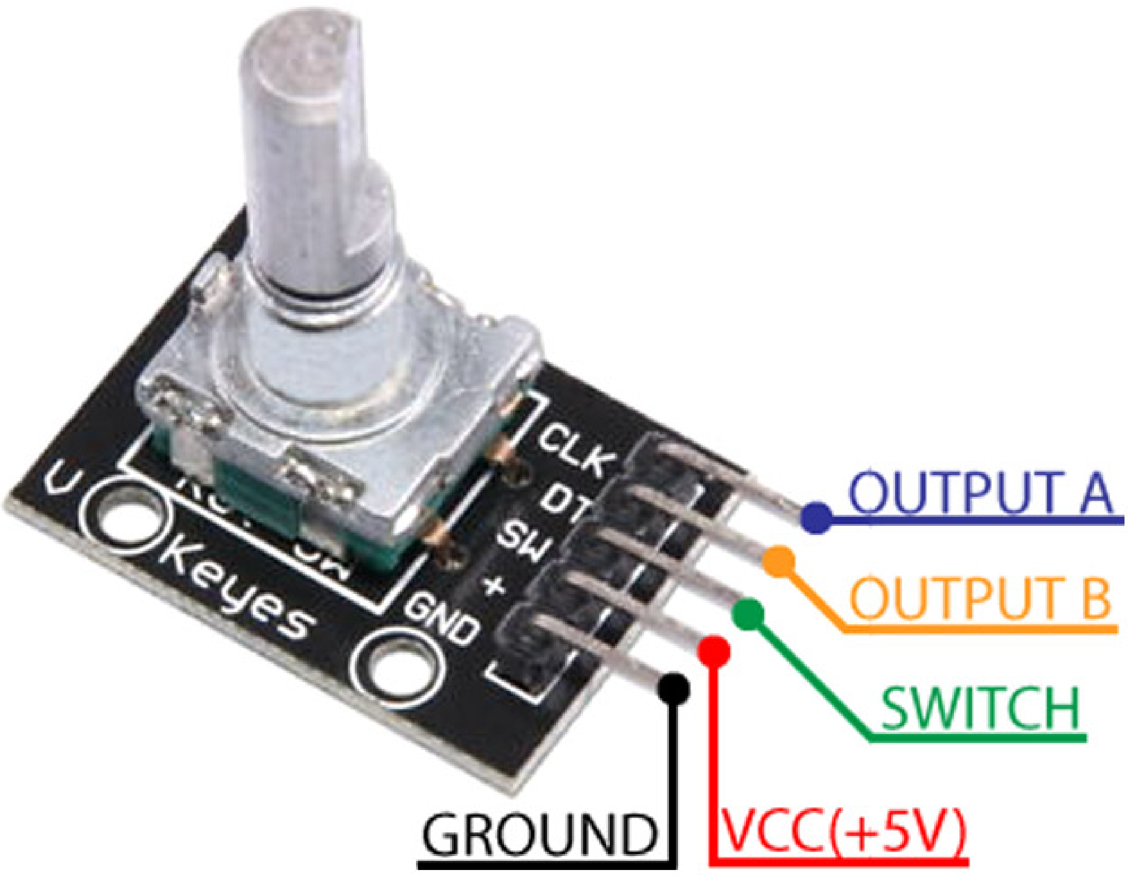


Figure 2: Mechanical Encoder KY-040

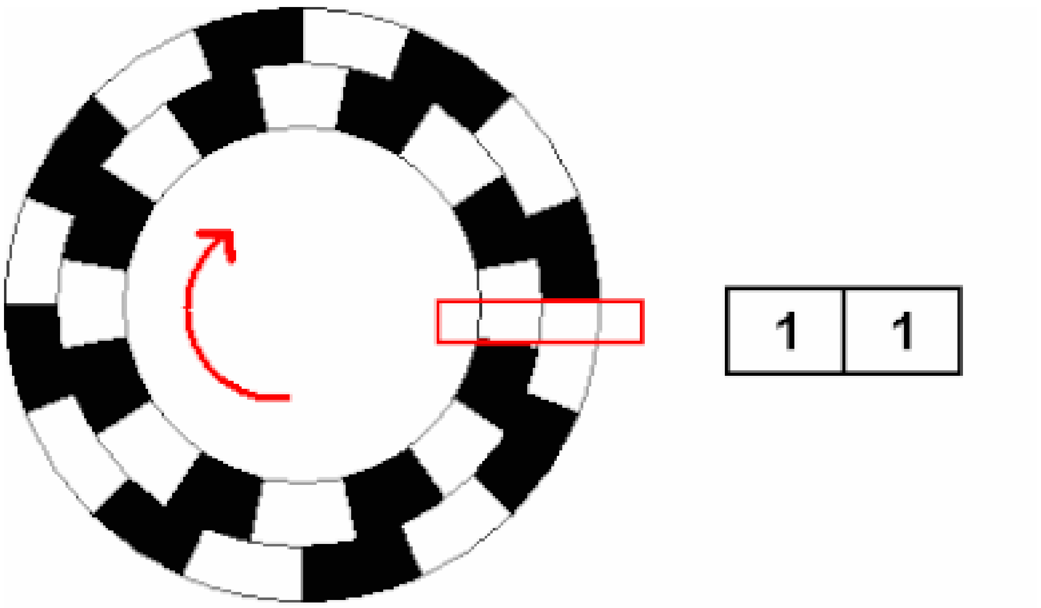


Figure 3: Quadrature code change in clockwise direction

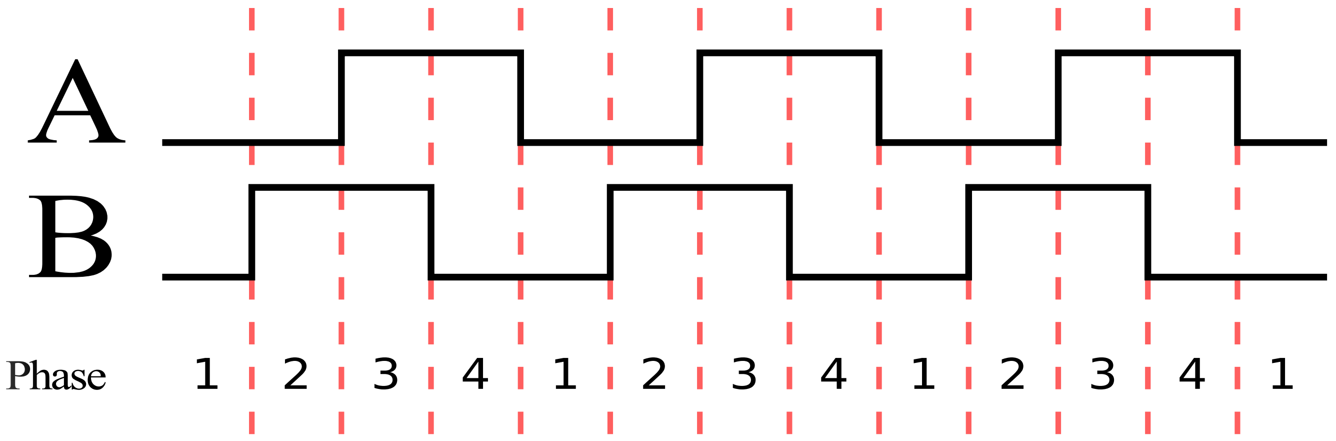
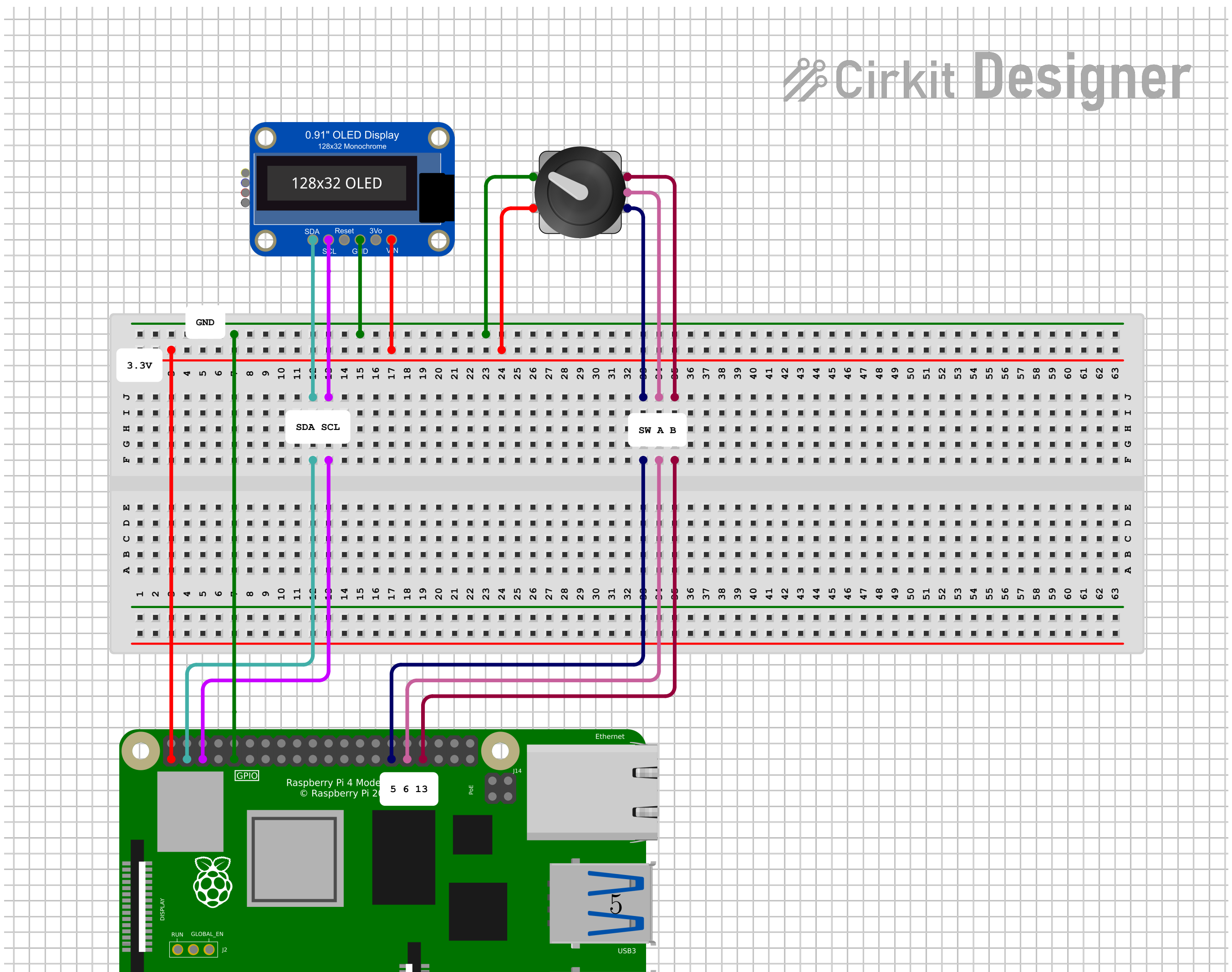


Figure 4: Quadrature A/B signal pulses

Item	Description
Raspberry Pi	
SSD1306 I2C OLED 128x32 Display	
KY-040 Quadrature Encoder	
Breadboard	
Jumper Wire	
4A USB Power Supply	



Methods

Raspberry Pi System Software Prerequisites

1. Install python

   ```
   sudo apt install -y \
       python3 \
       python3-pip \
       python3-venv \
       python3-setuptools \
       python3-flake8 \
       python3-flake8-docstrings \
       python3-autopep8 \
       python3-rpi.gpio
   ```

2. Enable i2C in `raspi-config`

   ```
   sudo raspi-config
   ```

   Set `3 Interface Options -> I5 I2C` to `Yes`.

   Check that i2c is available

   ```
   ls -l /dev/i2c*
   ```

   Show connected devices

   ```
   i2cdetect -y 1
   ```

3. Update i2C bitrate

   ```
   echo "dtparam=i2c_baudrate=1000000" | sudo tee -a /boot/config.txt
   ```

   Reboot the module.

4. Update bash config with local directory `~/.local` added to `PATH` environment variable

   ```
   mkdir -p "${HOME}/.local/bin"
   echo "source ${HOME}/.bashrc" >> ${HOME}/.bash_profile
   echo "export PYTHONPATH=\$PYTHONPATH:/usr/local/lib/python3.9/site-packages" >> ${HOME}/.bash_profile
   echo "export PATH=\$PATH:${HOME}/.local/bin" >> ${HOME}/.bash_profile
   ```

Lab Software Requirements

Install matplotlib for plotting, pillow library for LED screen imaging, and scipy with numpy for numerical processing.

```
sudo apt install -y \
    python3-scipy \
    python3-numpy \
    python3-matplotlib \
    python3-pil
```

Install Python packages Encoder and adafruit-circuitpython-ssd1306 for the OLED display hardware interface.

```
python3 -m pip install --user \
    Encoder \
    adafruit-circuitpython-ssd1306
```

Code Overview

Data Collection

Run the data collection script

```
./scripts/Lab7/scripts/run_data_collection.sh
```

Analysis

Example output

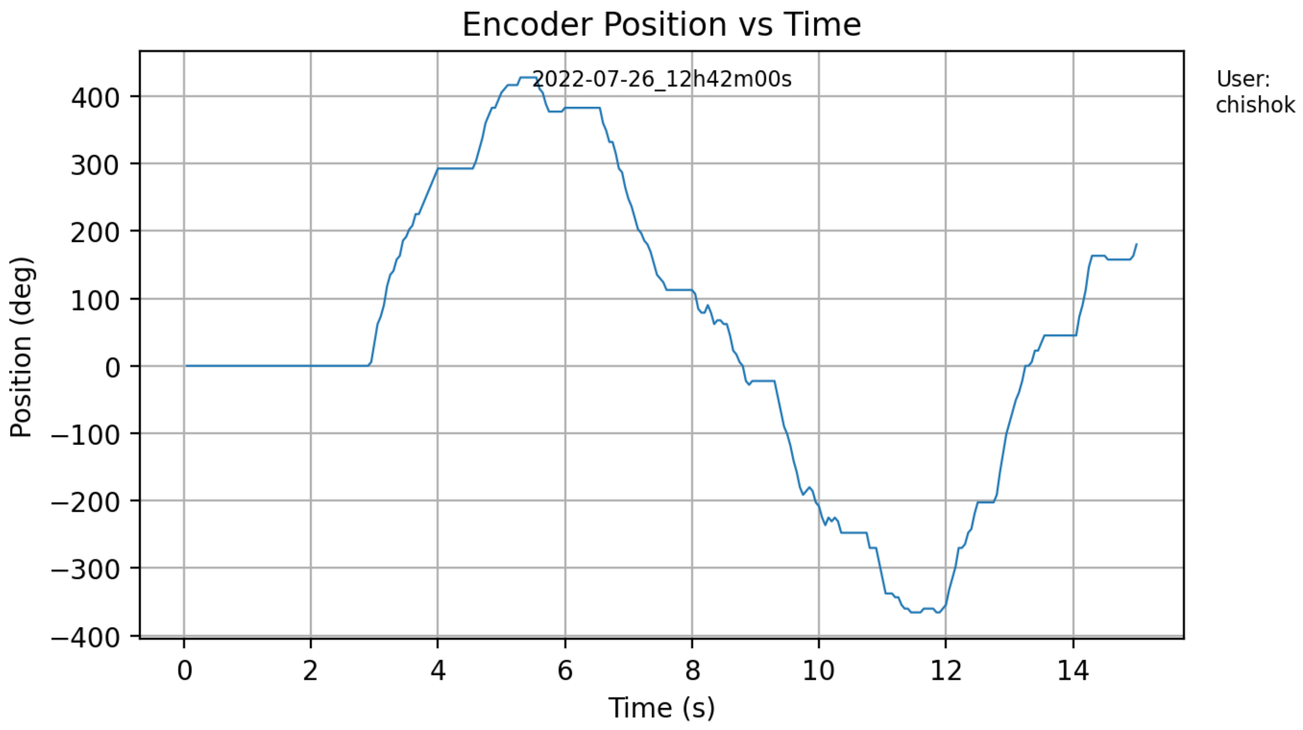


Figure 6: Example of position data output at 50Hz sampling rate.

Submission Guidelines

References

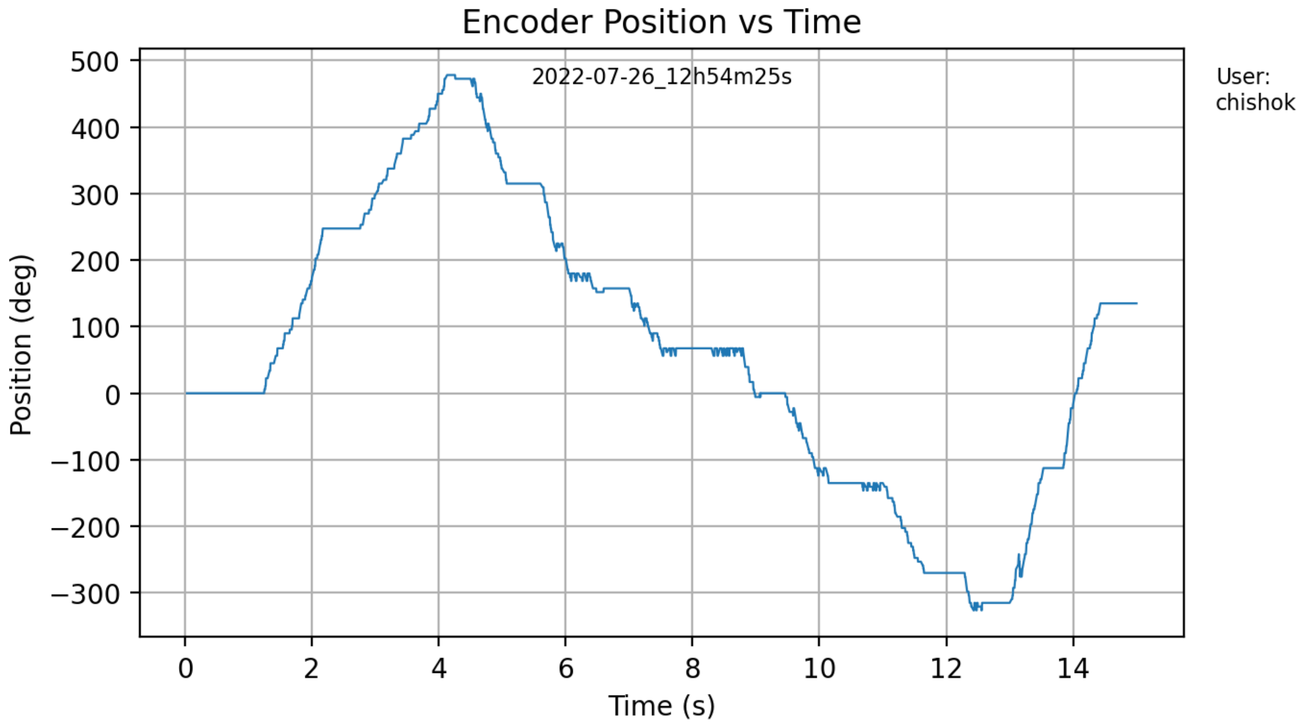


Figure 7: Example of position data output at 200Hz sampling rate.