(1)

(a)

Encoders are vital for motion control systems, which use a "closed-loop feedback" system, where a controller sends a command to a motor, and the encoder reports the motor's actual position, speed, and direction. This allows the controller to make real-time adjustments to ensure the motor performs the task precisely as commanded.

So Encoders are like mediums that receive signals from the sender and control motors or other obstacle-detecting components. So before getting into the solution let me give you some description of the encoders.

Encoders usually have 3 pins where the middle pin is to be connected to the “GND” /ground and other 2 pins act as “VCC”. So from the code I can see that the pins are indicating only the “input” values not the efficient one. Because you need resistors here.

Why?

Because the airhas infinite electrons and electrons are the reasons that current flow. So if current flow occurs in order to control it we need resistors to make it HIGH or LOW flow. So as the Encoders give electrical signals to the components, it’s vital to control signals (current flow) and to do this it’s important to use specific resistors.

And if you don’t use resistors or don’t manually code it to control HIGH/LOW signals the rover will automatically pick signals from the existing electrons from the air. And that’s why the Encoders showed Random values.

From the code it’s clear that the problem occurred because of not controlling the current flow properly through resistors or manual detection.

(b)

What we can do is :

Change this code-

```
pinMode(pinA, INPUT);
pinMode(pinB, INPUT);
```

To

```
pinMode(pinA, INPUT_PULLUP);
pinMode(pinB, INPUT_PULLUP);
```

Or

Use a compatible resistor that will do the task of PULLUP . This ensures the pins are pulled to HIGH by default, and the encoder switches them LOW when rotated preventing random noise.

(2)

In order to detect the pulse per Revolution he may add a buzzer or sound system that will let him know if the rover is speeding or the wheels are rotating. But in this case probably “ONE-REVOLUTION COUNT” can be very useful.

ONE-REVOLUTION COUNT:

This may be the most reliable way to get encoder **PPR (pulses per mechanical wheel revolution)** using only the rover.

We need to measure how many encoder pulses are produced when the wheel makes exactly one full revolution. Then we convert that into PPR depending on your counting mode.

First what we can do is:

1. We Put a small, high-contrast mark on the wheel rim. This will be the start/stop marker.

2. Then we lift the wheel so it spins freely so that we cna ensure the wheels are moving but rover is still like a statue much like smooth transistion.

3. Ensure encoder signal wires are short and solid and enable PULL_UPS that we need in our previous problem to avoid floating noise.

4. Then we use the MCU serial monitor or logging to observe the pulse count live.
5. If we find no button we'll use a switch in code instead.

Let's come to the measurements part:

We reset pulse counter by serial. Then slowly rotate the wheel until the mark returns to the start position after exactly one full rotation.Then we'll stop. Then we mark the reading of the displayed pulse count `Ex： C1`. We'll repeat this process at least 5 times (C1, C2, C3, C4, C5) in order to find average datas,mean μ\muμ and standard deviation σ\sigmaσ. If values are identical then it's considered: done.

The more datas we can collect the more accuracy we can get.

Now to count PPR there are several processes. For example:

If counted the only rising edges of AttachInterept then measured value = **PPR_rising**. That equals the number of rising edges per wheel revolution.

If counted `CHANGE` on both A and B which is a full qudrature decoder then we actually counted **4 × mechanical PPR** .

This is how we will convert the datas to mechanical PPR:

- If used RISING only then `PPR_mech = measured`.
- If you used CHANGE on A and B then `PPR_mech = measured / 4`.
- If you used both edges of A (2×) → `PPR_mech = measured / 2`.

Now, after doing the above tasks If PPR_mech looks unreasonably large (>1000) for a small encoder likely encoder is on motor shaft or you used 4× counting.

Then to check: We'll have to rotate motor shaft one rev and count pulses and compare to wheel-one-rev count to find gear ratio.

FOR ODOMETRY:

We will use the mechanical PPR (pulses per wheel revolution) in odometry. If we used quadrature decoding in our rover code then we'll use the register value per wheel revolution directly in distance calculations.

odometry distance per pulse = wheel_circumference / (mechanical_PPR × multiplier),

where multiplier is 1 if PPR is mechanical per rev, or 2/4 if counting mode multiplied it.

Then we'll measure diameter and C(count) = π × d. Then distance per pulse = C / PPR_mech .

We may have to divide for for different multipliers such as: X2 , X4 ……

(3)

Why encoder-only distance is inaccurate?

Encoders measure how much the wheel rotates, not how far the rover's body actually moves. On rough or uneven terrain, the wheels can slip, skid, or lift off the ground. When that happens, the encoder still counts pulses because the wheel is turning, but the rover might not be moving as much — or even at all. This causes the rover to overestimate or underestimate the real distance.Other issues like uneven wheel sizes, tire wear, or loose ground can also make the readings drift over time. As a result, relying only on encoders leads to inaccurate odometry, especially over long distances or bumpy surfaces.

How to fix or reduce this error

1. **We can use an IMU (Inertial Measurement Unit):**

   It measures acceleration and rotation, helping detect when the rover is moving or slipping. Combining IMU data with encoder readings improves accuracy.

2. **We can Add GPS (when signal is available):**

   GPS provides real-world position data that can be used to periodically correct encoder drift.

3. **We can use Sensor Fusion algorithms:**

   A software method that merges encoder, IMU, and GPS data to give a smoother and more reliable position estimate.

4. **We may use Slip Detection and Calibration:**

   The software can compare left and right wheel speeds or IMU motion to detect slipping and adjust calculations accordingly.