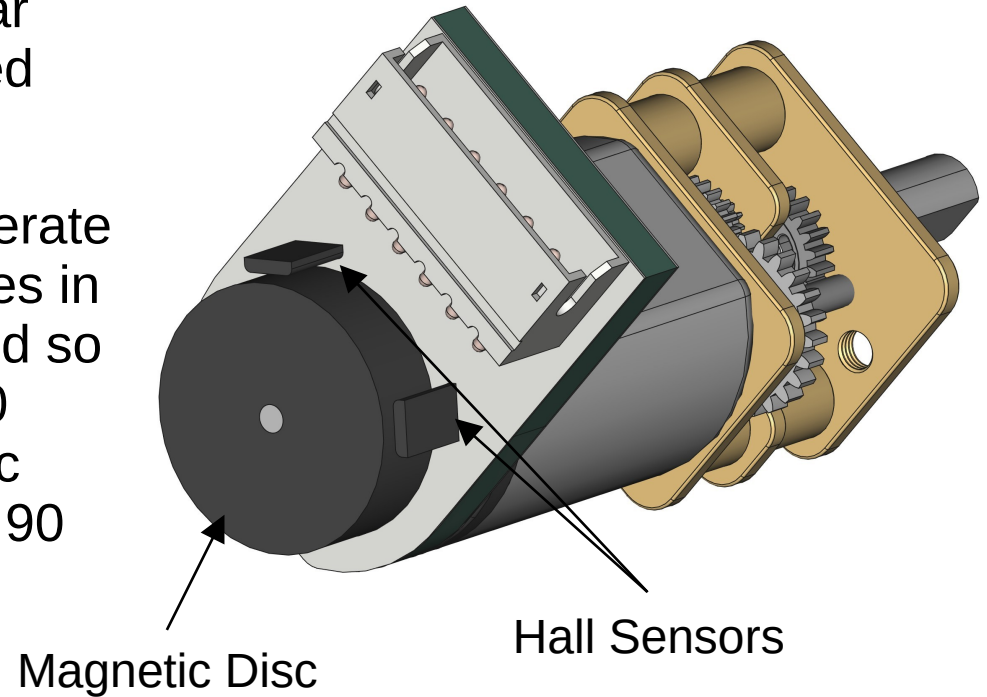


Wheel Encoders

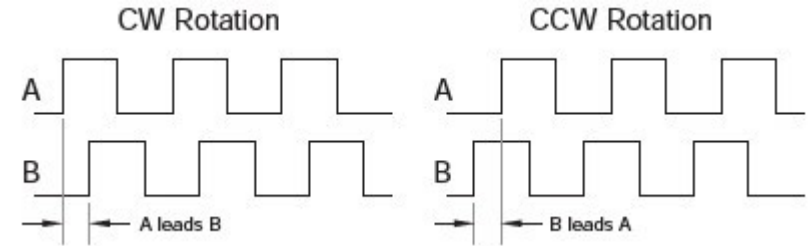
Motor Encoder

- Each motor has an encoder. It consists of a disc magnet installed on the rear shaft of the motor. The disc is divided into 7 poles.
- There are two hall sensors that generate a pulse when a magnetic pole passes in front. The hall sensors are positioned so the pulses generated are phased 90 degrees from each other. As the disc rotates, the hall sensors generate a 90 degree square wave.



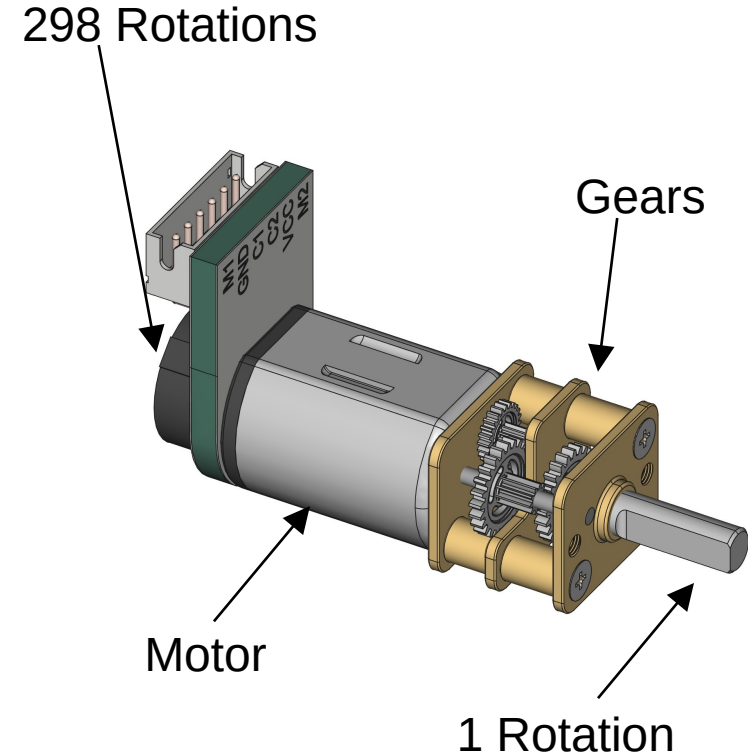
Motor Encoder

- The waveforms to the right show how the two hall sensors generate their waveforms. One leads the other by 90 degrees.
- The direction can be determined by the change in phase.
- The microcontroller checks the phase on the rising edge of signal A. If signal B is low, the motor is rotating clockwise. If signal B is high, the motor is turning counter clockwise.



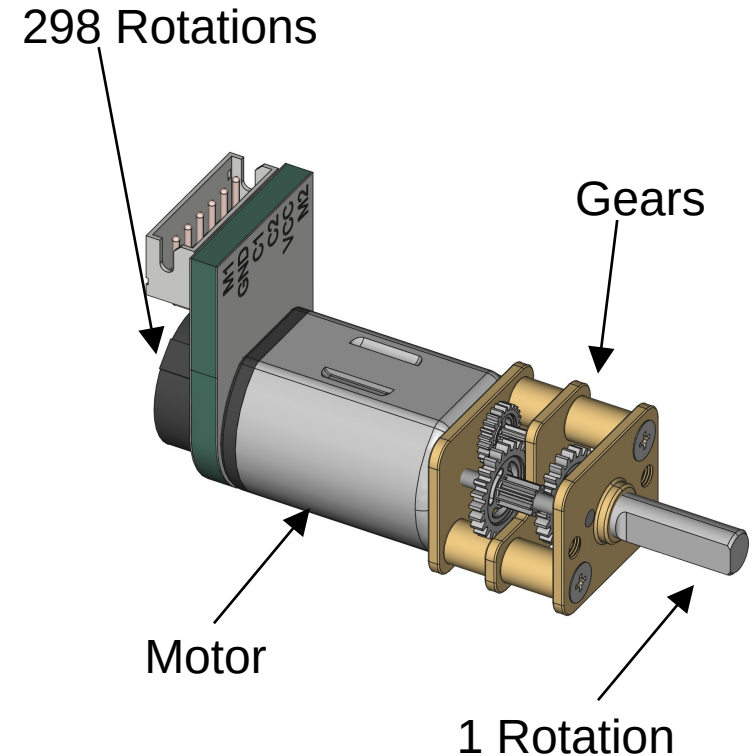
Determining Distance

- The wheel encoder is on the shaft of the motor that drives the gears.
- The gears has a 298 to 1 reduction which means the tire rotates once every time the motor rotates 298 times.
- With 7 pulses per rotation and 298:1 gear ratio, the total number of encoder pulses per output rotation is 2086 pulses.
- The diameter of the wheel is 98 mm. What is the distance per pulse the wheel will travel?



Determining Distance

- To determine the distance per encoder pulse, the circumference of the wheel needs to be calculated.
- The circumference is calculated by $\pi * \text{diameter}$.
- For a 98 mm wheel, the circumference is 307.876 mm or 0.307876 meters.
- With 2086 pulses per revolution, the circumference is divided by 2086 to get the distance per pulse which is 0.000147591 meters or 0.1476 mm.



Capturing Wheel Movement

- Looking at the `ugv.py` module, the `get_encoders()` function will be used to get the current encoder values for each wheel.
- The function first reads 16 bytes from the ugv board microcontroller. Each wheel encoder value is 32-bits which is 4 bytes.
- The 16 bytes are then unpacked using the **`struct.unpack()`** function. The `'<llll'` indicates how the bytes are formatted. The `<` symbol indicates the 32-bit words are in little endian format which means the least significant byte is sent first. The letter `l` indicates the data is 4 bytes long and there are four of them.
- The four values are returned as a tuple.

```
import struct

class ugv:
    def __init__(self, bus):
        self.i2c = bus

    def get_encoders(self):
        m = self.i2c.readfrom(25, 16)
        me = struct.unpack('<llll', m)
        return(me)

    def drive(self, lf, lr, rf, rr):
        v = struct.pack('bbbb', lf, lr, rf, rr)
        l = self.i2c.writeto_mem(25, 0, v)
```

Capturing Wheel Movement

- Enter the program to the right into Thonny and save it as **rover6.py**.
- You can remove the wheels from the rover or hold it up.
- Make sure the power switch is on.
- Run the program.
- It should start printing out encoder values.
- You can stop the program and switch power to off.
- All the values should be increasing. You will notice the values on each line are a little different from each other. This is due to the differences in the manufacturing of the motors.

```
import ugv
import time
from machine import Pin, SoftI2C

i2c = SoftI2C(sda=Pin(4), scl=Pin(5))
drv = ugv.ugv(i2c)
drv.drive(100,100,100,100)
while 1:
    enc = drv.get_encoders()
    print(enc)
    time.sleep(0.1)
```

Driving the Distance

- This program will wait for parameters to indicate the distance to travel and the travel speed.
- Create a new program and call it **rover6a.py**.
- The program will be written as a class since we will add to it and modify it.
- After the module imports, a class called UGV is created.

```
import ugv
import time
from machine import Pin, SoftI2C
import select
import connect

class UGV():
    def __init__(self):
        i2c = SoftI2C(sda=Pin(4), scl=Pin(5))
        drv = ugv.ugv(i2c)
        last_enc = [0,0,0,0]
        distance = 0
        sk, addr = connect.connect()
        while 1:
            d, a = sk.recvfrom(64)
            d = d.decode().split(' ')
            left = int(d[0])
            right = int(d[1])
            new_dist = int(d[2])
            enc = drv.get_encoders()
            dist, last_enc = self.get_distance(enc, last_enc)
            distance += dist
            dst_dist = distance + new_dist
            drv.drive(left, left, right, right)
```

Driving the Distance

- In the `__init__()` function, the I2C interface is instantiated and the ugv board module is instantiated.
- A list called `last_enc` is created. It will hold the previous encoder values which will be used in calculating the distance traveled by the rover.
- `distance` is created which will track the overall distance traveled.
- Next, the rover waits for a network connection.

```
import ugv
import time
from machine import Pin, SoftI2C
import select
import connect

class UGV():
    def __init__(self):
        i2c = SoftI2C(sda=Pin(4), scl=Pin(5))
        drv = ugv.ugv(i2c)
        last_enc = [0,0,0,0]
        distance = 0
        sk, addr = connect.connect()
        while 1:
            d, a = sk.recvfrom(64)
            d = d.decode().split(' ')
            left = int(d[0])
            right = int(d[1])
            new_dist = int(d[2])
            enc = drv.get_encoders()
            dist, last_enc = self.get_distance(enc, last_enc)
            distance += dist
            dst_dist = distance + new_dist
            drv.drive(left, left, right, right)
```

Driving the Distance

- In the while loop, the program waits for values from the laptop.
- Once received, the data is converted to a string and then split using a space as the delimiter.
- The left and right speeds, and distance to travel values are converted from strings to integers.
- The current encoder values are collected and any distance change is calculated and added to the distance.
- The target distance is calculated and the motors activated.

```
import ugv
import time
from machine import Pin, SoftI2C
import select
import connect

class UGV():
    def __init__(self):
        i2c = SoftI2C(sda=Pin(4), scl=Pin(5))
        drv = ugv.ugv(i2c)
        last_enc = [0,0,0,0]
        distance = 0
        sk, addr = connect.connect()
        while 1:
            d, a = sk.recvfrom(64)
            d = d.decode().split(' ')
            left = int(d[0])
            right = int(d[1])
            new_dist = int(d[2])
            enc = drv.get_encoders()
            dist, last_enc = self.get_distance(enc, last_enc)
            distance += dist
            dst_dist = distance + new_dist
            drv.drive(left, left, right, right)
```

Driving the Distance

- Continuing from the last page, a second while loop is used to check on the distance traveled and compare it to the target distance.
- The encoder values are collected, the distance is traveled is calculated and the distance traveled is updated.
- The distance is compared with the target distance and if the rover has traveled long enough, the rover is stopped and the **break** exits the while loop back in to the main loop waiting for the next set of values.

```
while 1:
    enc = drv.get_encoders()
    dist, last_enc = \
        self.get_distance(enc, last_enc)
    distance += dist
    print(distance, dst_dist)
    if distance >= dst_dist:
        drv.drive(0,0,0,0)
        break
    time.sleep(0.02)
```

Driving the Distance

- After the `__init__()` function, another function is added. This function calculates the distance traveled between encoder values collected.
- The function takes two arguments, the current encoder values and the last encoder values.
- It calculates the distance each wheel traveled and then adds the absolute value and averages them.
- The function returns the distance traveled and the current encoder values.

```
def get_distance(self, encoder, last):  
    dest = [0,0,0,0]  
    for i in range(4):  
        dest[i] = encoder[i] - last[i]  
        last[i] = encoder[i]  
    dist = (abs(dest[0]) + abs(dest[1]) + \  
           abs(dest[2]) + abs(dest[3]))/4  
    return dist, last
```

UGV()

Driving the Distance

- The absolute values of the difference between the encoder and the last encoder values lets us calculate the distance the wheels travel no matter which direction. This more like an odometer calculation.
- Lastly, the class is called which runs the code.

```
def get_distance(self, encoder, last):  
    dest = [0,0,0,0]  
    for i in range(4):  
        dest[i] = encoder[i] - last[i]  
        last[i] = encoder[i]  
    dist = (abs(dest[0]) + abs(dest[1]) + \  
           abs(dest[2]) + abs(dest[3]))/4  
    return dist, last
```

UGV()

Commanding the Distance

- Moving on to the laptop, enter the program into the Python IDLE text editor and save it as **controller.py**.
- The program creates a UDP socket port.
- It sends a start command to the robot to trigger the **connect()** function.
- The loop then waits for a user input. Enter four numbers separated by a space. The first value is the left wheel speed. The second value is the right wheel speed. The third value is the left wheel distance encoder count. The last value is the right wheel distance encoder count.
- Remember, one wheel rotation is 2086 counts. Wheel direction is controlled by the sign of the speed value.
- Run the program on the robot, connect the laptop WiFi to the robot and run this program.
- Enter the values to move the robot different distances.

```
import socket

s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.sendto(b'start', ('192.168.4.1', 10000))

while 1:
    print("Enter speed and distance")
    a = input()
    s.sendto(a.encode(), ('192.168.4.1', 10000))
```

Laptop Program

- Remember, one wheel rotation is 2086 counts. Wheel direction is controlled by the sign of the speed value which has a range of -127 to 127.
- Run the program on the robot, connect the laptop WiFi to the robot and run this program.
- To go forward 1000 encoder steps enter:
 - 50 50 1000
- Enter the values to move the robot different distances and directions. Try negative values on the speed inputs.

- Knowing the diameter of the wheel and the number of encoder counts for one revolution, calculate the distance traveled in four wheel revolutions. Command the robot to move that distance and measure the actual distance traveled. Compare to the calculated distance.
- You may need to adjust the speed of each wheel to get the robot to travel in a straight line.
- Determine how many encoder counts are needed to turn 90 degrees in place. Command the robot to turn in place and verify it turned 90 degrees. Try both directions and see if there is any difference.

