Chapter 1 (ctd)
Robot Kinematics

A brief Introduction – Picking up the Thread

Here we shall look at improving our work on movement using dead reckoning by employing a robot *proprioceptive* sensor which measures the position of the wheels.

Wheel encoder technology

The sensing device is called a 'wheel encoder', see Fig.1. As the wheel rotates, IRed light from the transmitter shown at the top passes through slots in the wheel, shown at the bottom. When the light hits a spoke, it is reflected to the IRed sensor and produces a pulse sent to the Arduino. So, as the wheel rotates, it sends a series of pulses to the Arduino shown in the diagram below.

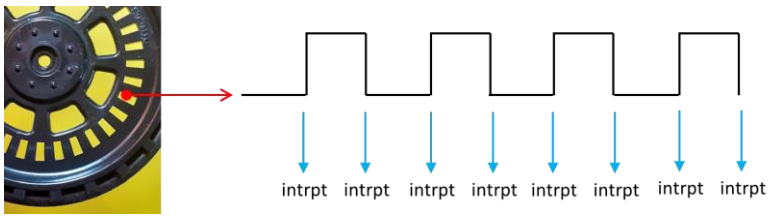


Figure 1 Wheel encoder: Top the IRed transmitter and receiver measures reflections from the spokes (bottom)

The pulses arrive at Arduino pins especially configured to receive hardware 'interrupts' and when a pulse arrives, the code breaks out of its current execution place and jumps to an *Interrupt Service Routine (ISR)* and execution continues there. When the ISR is complete, then execution returns to the place where it was forced to break out from. Each wheel has an interrupt, the ISR for the right wheel is shown below

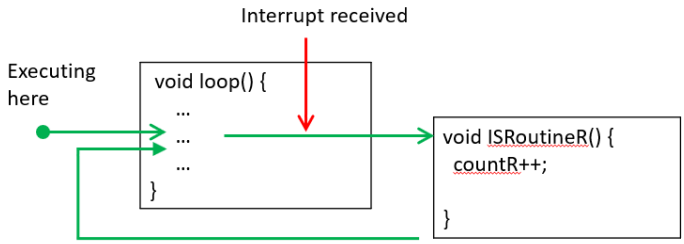
```
void ISRoutineR() {
  countR++;
}
```

You can see this ISR increments the value of **countR** i.e., every time the encoder sends a pulse, this value is

incremented, so the code knows how many steps the wheel has rotated. Here's an overview of how interrupts work. First the hardware pin is attached to the ISR, and the interrupt is configured to respond to a CHANGE (i.e., a *rising* or a *falling* pulse edge. This code is in **setup();**

```
attachInterrupt(digitalPinToInterrupt(EncoderR
_pin),ISRoutineR,CHANGE);
```

Here's what happens when the code is executing in **loop()** and a pulse arrives.



Code execution is shown by the green arrow with a blob. When the interrupt is received, execution transfers execution returns to where it left off. It is important to understand that this occurs at the level of hardware instructions, not lines of code. It is extremely fast.

Now we need to understand how many pulses are received in one wheel revolution, and therefore how far the wheel moves between pulses. There are 32 spokes in the wheel, and since the ISR responds to both rising and falling edges, then there are 64 pulses per wheel revolution. The angle rotated between pulses is a little under 6 degrees. But how far has the wheel moved? We know that

$$\Delta s = r\Delta\theta$$

so for a wheel radius of 33 mm, we have, calculating using radians

$$\Delta s = 33\frac{2\pi}{64}$$

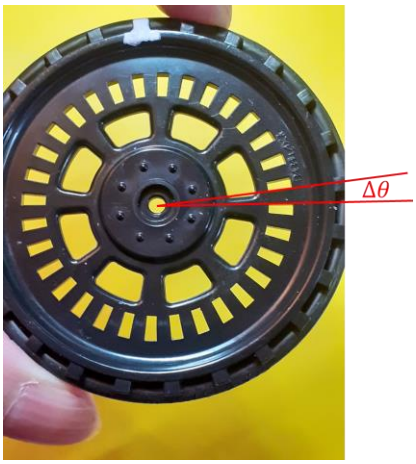


Figure 2 Angle shown between pair of rising and falling pulse edges

which works out to be 3.24 mm. This is the accuracy of any measurement of wheel travel we can make since it is the smallest step in distance we can possibly know.

Moving on a straight line of given distance.

Let's say we want the robot to move a desired distance. How do we use wheel encoders to make this happen? First we need to calculate the number of steps required, which is the number of pulses the encoders receive, setting **countL** and **countR**. The number of pulses is simply the distance gone divided by the step size,

$$n_L = n_R = \frac{desired\ dist}{\Delta x}$$

For example to get the robot to move 300 mm, we need to count 300/3.24 = 93 pulses, approx. Here's some example code which will do the job. We set the motor drives and switch on the servos. Then we monitor the actual counts and when they exceed the counts we need ($n_L = n_R$) then we stop the servos.

```
driveL = drive;
driveR = drive;
driveServos(driveL,driveR);

if((countL > nL) && (countR > nR))  {
  driveServos(0.0,0.0);
  servosDetach();
}
```

Moving on an Arc

We've already seen the maths for this, but let's revisit it here. The arrangement is shown in Fig.3 where the symbol θ_{Rob} refers to the rotation of the entire robot about its centre (in the middle of its axle). The arc is specified by this angle and also the radius of the curve. Remember the robot axle has length $2a$ as shown in the diagram.

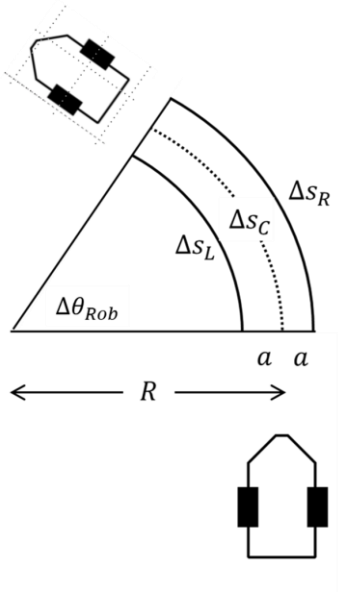


Figure 3 Geometry for moving on a curve with specified angle and radius.

The robot's wheels travel the following distances when traversing the arc,

$$\Delta s_L = (R - a)\Delta\theta_{Rob}$$

$$\Delta s_R = (R + a)\Delta\theta_{Rob}$$

The number of pulses is calculated by dividing the distances travelled by the step size Δx. So, we get

$$n_L = \frac{\Delta s_L}{\Delta x} = \frac{(R - a)\Delta\theta_{Rob}}{\Delta x}$$

and a similar expression for n_L. All quantities on the right are known, so it is easy to calculate the number of pulses for each wheel.

There is one additional factor we need to take into account. Not only is the right wheel making more steps, but it is also moving faster. Since both wheels must start and stop at the same time, the speeds are proportional to the number of steps. So, we have where *fwd* represents some *drive* to the servos,

$$\frac{fwd_R}{fwd_L} = \frac{n_R}{n_L}$$

The following code does all of this for us, where we are asking the robot to make a 90-degree arc of radius 300 mm.

```
desRadius = 300;
desDegrees = 90;
desTheta = desDegrees*(PI/180.0);

nLf =(desRadius - axleLen/2.0)*desTheta/dx;
nRf =(desRadius + axleLen/2.0)*desTheta/dx;

nL = (unsigned long) nLf;
nR = (unsigned long) nRf;
fwdL = 20;
fwdR = fwdL*nRf/nLf;
```

6 Robotics