Chapter 3
Sensors

3.1 A brief Introduction

As humans our bodies are full of sensors, we depend heavily on sensing the external environment through vision, sound and touch. These sensors respond to the *external* environment and are responsible for giving us information about what is out there. This could be for navigation through a building, driving (while avoiding obstacles) reading a menu and making a choice of food to eat. Sensors which respond to the external environment are called ***exteroceptive***. We also have built-in sensors which give us information about our bodies (and perhaps minds). We can sense when we have a toothache, when we are hungry, when we are tired or irritable. Sensors which respond to our internals are called ***proprioceptive***.

The same classification is true for a robot, typical internal variables are battery voltage, motor speed, load on the wheels. External robot sensors can provide measurements of distance, sound amplitude and pitch and light intensity. Speed can be measured by doppler effect (change in pitch when a sound bounces off a moving object), and computer Vision can lead to object recognition (and therefore avoidance), path following and visual ranging. In this chapter we shall but take a small taste of what is possible and reflect on what we can achieve in our lab.

3.2 Active Ranging

3.2.1 Ultrasonic Pinging

The operation of the HC-SR04 ultrasonic ping detector is shown in Fig.3.1. This is somewhat like the principles used by bats in echo location. A distance measurement cycle begins by emitting a pulse of 40kHz waves from the transmitter Tx. This pulse spreads out (with a beam-width of

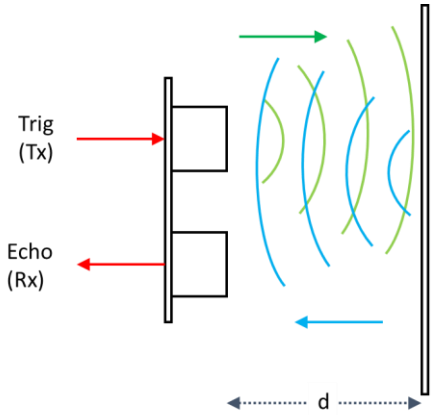


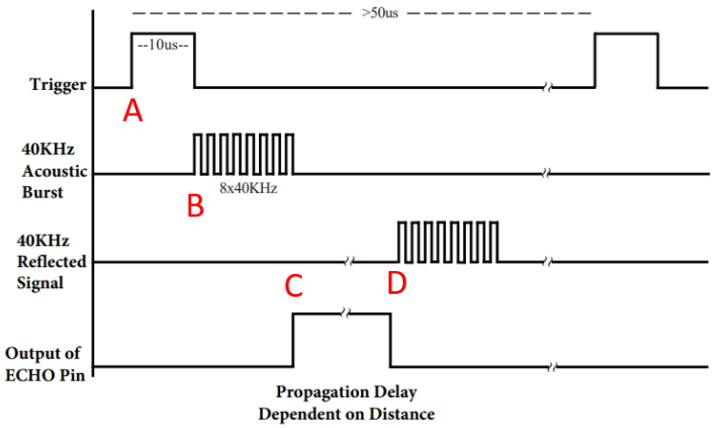
Figure 3.1 Ultrasonic Ping; green shows emitted pulse and blue reflected pulse.

about 30 degrees), and if it encounters an obstacle, then it is reflected and may arrive at the receiver Rx. The device gives us the time for the round trip which is (using the definition of velocity = distance over time)

$$t = \frac{2d}{v}$$

where *v* is the velocity of sound, around 330 m/s. Note the factor 2 since the round trip has distance *2d*. Fig.3.1 reminds us that we must *output* a trigger pulse to and *input* an echo pulse from the HC-SR04. Let's see the details and some code.

The diagram below shows the sequence of operations. At time 'A' we raise the value of trigger from LOW to HIGH and hold it there for 10μsec. Then, at time 'B' the device emits 8 ultrasonic pulses with frequency 40 kHz. When this is complete, the device raises its echo pin (which was LOW) to HIGH, telling our code the pulse has been emitted, time 'C'. When the pulse is reflected and enters the receiver, time 'D', then the ECHO pin goes low. So, by measuring the time the echo pin remains high, we know the time it takes for the pulse to do its round trip.



So, we can invert the above expression and calculate *d*.

$$d = \frac{vt}{2}$$

The code to do this is straightforward.

```
digitalWrite(HC_SR04_Tx,LOW);
delayMicroseconds(2);
digitalWrite(HC_SR04_Tx,HIGH);
delayMicroseconds(10);
digitalWrite(HC_SR04_Tx,LOW);
duration = pulseIn(HC_SR04_Rx,HIGH);
mm = 10*duration / 29 / 2;
```

The last line looks a little odd. This does several things; it converts duration (measured in microseconds) to seconds, then it converts the speed of sound 330 m/s to mm/s, and then it divides by 2 to take account of the round trip. The calculation has been done using integer arithmetic.

3.2.2 Limitations of Ultrasonic pinging

Think of an object around 2 metres away, this could easily be a wall in a room the robot must detect. It will take the ultrasonic pulse around 0.01 seconds (10 ms) to make its round trip. Our code must wait for this time to get the distance (unless we use interrupts), even then, the distance information is taking too long to be computed. A robot equipped with 10 such sensors arranged to measure distances all around its body must wait 0.1 seconds for this information to be available and moving at a leisurely speed of 100 mm/s, it would have travelled 10 mm and maybe suffered a collision. So, we must turn to faster approaches such as laser pings (LIDAR) which operate at the speed of light providing a speedup relative to sound of approximately $3 \times 10^8 / 3 \times 10^2$ which is about a million times.

3.3 Line Following

3.3.1 Aside – Resistors and Phototransistors

We have already encountered robot line following in Chapter 2 where we looked at the PID control algorithm. Now we need to focus on the details of the sensor system, so we can understand how to code the control algorithm. The following notes are from the point of view of an electronic engineer, so we have to get some basics in place, especially resistor circuits and circuits with a resistor and a phototransistor. Let's not worry about the theory, but rather look at some examples, plus a little dose of logic.

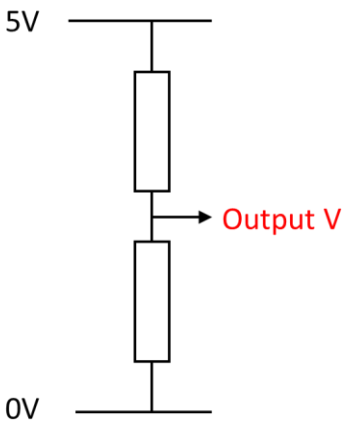
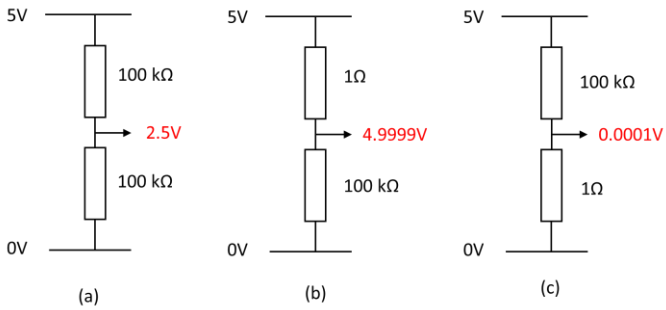


Figure 3.2 Voltage divider arrangement (resistor values not specified)

The starting point is *voltage*; we know that the Arduino can output a LOW signal (0 volts) and also a HIGH signal (5 volts) on a digital output pin. We also know it can input a signal on a digital input pin, and this can be HIGH (5 volts) or LOW (0 volts). So, voltage seems to be the key in understanding circuits. Now most input devices can be thought of as being configured as part of a *voltage divider*. This is shown in Fig.3.2 The rectangles are resistors, and these are connected between 5V and 0V (Gnd). There is an output voltage, so the question is what can this be? Well it can't be more than 5 or less than 0, since these values are not available at the input, so we conclude that the output voltage must be in the range 0.0 – 5.0. So, let's look at some concrete examples in the diagrams below.



In (a) we two identical resistors, so *logic* tells us the 5Volts must be divided into 2, so we get 2.5V out, it can't be

anything else. In (b) we have a small resistor at the top and a huge on at the bottom, the voltage out is almost 5V. This makes sense since the output is 'more connected' to the 5V rail than to the 0V rail (less resistance). Conversely in (c) the output is 'more connected' to the 0V rail, so the output is close to 0V. Of course, you may have notice that the larger voltage is found across the larger resistor.

Now let's turn to the phototransistor which is used for most light-sensing activities, such as in our line following scenario, this is shown in Fig.3.3 where two situations are shown. On the left (a) there is a low light level applied to the phototransistor (two green arrows) and in (b) there is a higher light level (four green arrows). We need to *know* something about a phototransistor

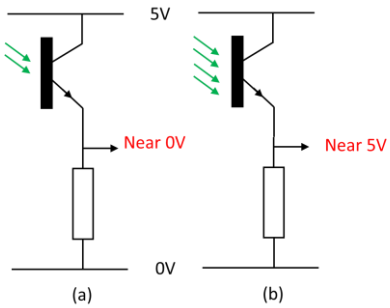


Figure 3.3 Phototransistor with (a) low level of light input, (b) high level of light input

> A phototransistor is like a resistor whose resistance changes with its light input:
> - Low light level – high resistance
> - High light level – low resistance

So, in Fig.3.3(a) the phototransistor has a high resistance, so following our argument above, the output will be nearer 0V. In Fig.3.3 (b) with a high light level, the phototransistor has a lower resistance, so the output is 'more connected' to the 5V rail, so the output is close to 5V. In other words, the output of this circuit can give us a binary value 'there is light' (5V) or 'there is no light' (0V).

3.3.2 A Simple Line Detector

Here we shall briefly discuss a simplistic line detector, but it's not the one we advise to use, it's a toy problem just to reinforce some of the thinking we have presented above, as a stepping-stone to a more serious line detector. Let's assume we have a robot with a left and right sensor as described above, see Fig. 3.4(a). The robot is following a dark line on a lighter background. We want to know how the above

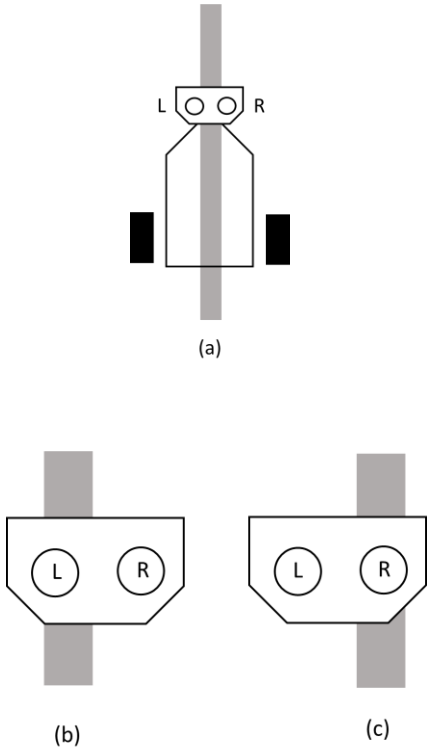


Figure 3.4 Simple line detection arrangement using phototransistors.

phototransistor circuit will give us information to keep the robot moving along the line.

In Fig.3.4(b) the robot has strayed to the right and must be instructed to rotate anti-clockwise. Let's have a look at the outputs of the L and R phototransistor circuit. The left one is darker, and therefore outputs a voltage close to 0V, and the right one is lighter, so outputs a voltage close to 5V. So, the left-right pair outputs {LOW, HIGH}. In Fig.3.4(c) we have the converse, the right phototransistor circuit is darker (outputting close to 0V) and the right one is lighter, (outputting close to 5V). So, the left-right pair outputs are {HIGH, LOW}. We can summarize this in the following table.

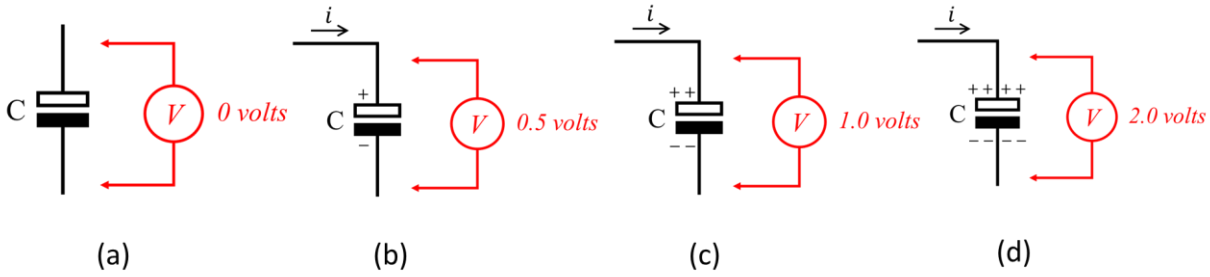
Situation		Sensors		Action
		L	R	
Fig.4(a)	on the line	HIGH	HIGH	continue forward
Fig.4(b)	off to the right	LOW	HIGH	rotate anti-clockwise
Fig.4(c)	off to the left	HIGH	LOW	rotate clockwise

This looks like potential input for a FSM as discussed in Chapter 2, providing the transit events between states. But as we also mentioned, this is a little brutal, and for smoother, more delicate control, we need a *continuous* input from our line sensor. So, let's see how we can achieve this. But first we need another step aside.

3.3.3 Aside – Capacitors and Phototransistors.

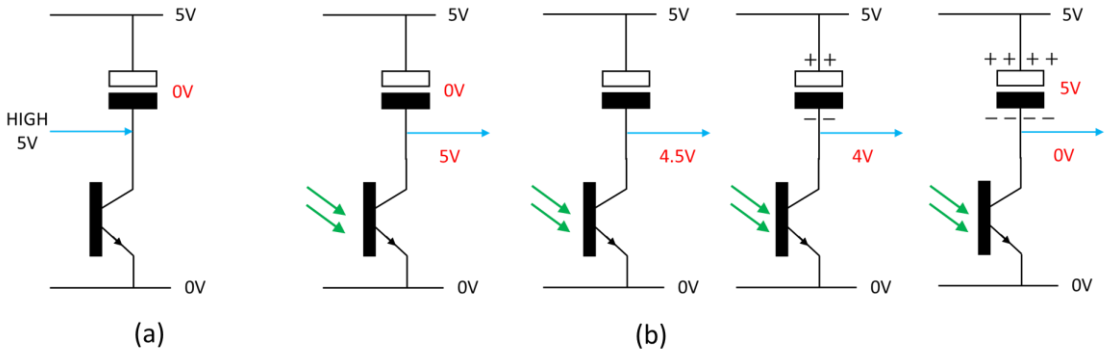
First let's turn to a capacitor; think of this like a bucket which accumulates charge when a current flows into it. When water flows into a bucket the water height increases and so does the pressure. When current flows into a capacitor the voltage (think pressure) similarly increases. Take a look at the diagram below.

In (a) the capacitor is empty, there is no charge so there is no voltage across it. In (b) current has started flowing into the capacitor, and you can see some charge has accumulated, so there is a small voltage across the capacitor.



As the current continues to flow so does the charge accumulate (c and d) and the voltage rises. You can see that voltage is proportional to charge.

So, here's the circuit for our delicate line detector with an explanation of how it works. The blue arrow line is a connexion to an Arduino input/output. In (a) this is configured as an output and is given a HIGH level. Therefore, there is no voltage across and capacitor which therefore does not hold any charge.



In (b) we see a time series of what happens when we remove the Arduino input, then the capacitor is free to charge, and as it charges, its voltage increases. So, the voltage across the phototransistor must decrease (since the voltage across both must always be 5V). Finally (rightmost circuit) there is 0V across the phototransistor.

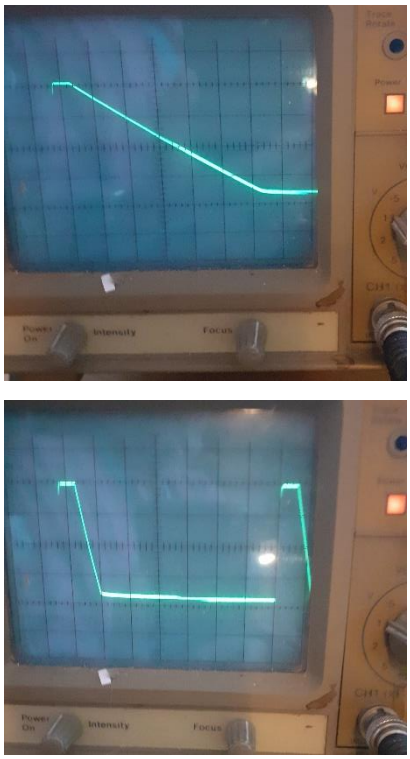


Figure 3.5 Oscilloscope traces of sensor response: Top dark surface, bottom light surface

How does this help us create a delicate sensor to detect lines? Well, remember that the resistance of a phototransistor depends on the amount of light falling on it, more light means less resistance. So, in the above circuit, when the phototransistor gets a lot of light, its resistance is small, so it the current into the capacitor is larger, it charges up faster, and so its voltage rises faster. Therefore, the voltage across the resistor drops faster. This is the voltage we measure, and we measure the time it takes to drop to near 0V. A smaller time means more light, so the sensor is looking at white; a larger time means less light, so the sensor is looking at black.

Fig.3.5 shows some actual measurements I made on the circuit. You are looking at oscilloscope traces of the voltage across the phototransistor (vertical axis) against time (horizontal axis). The top photo shows what happens when you present a dark surface to the sensor and the bottom shows a light surface. The times to go from 5V to 0V are about

dark surface	3 µsec
light surface	0.5 µsec

3.3.4 Coding all this Natural Nonsense

This all may seem complicated, perhaps you feel you have had a journey from Siberia to Nepal (via Hereford). So, let's look at some code, which may bring thinking together.

```
long RCTime(int sensorIn){
   long duration = 0;
   pinMode(sensorIn, OUTPUT);
   digitalWrite(sensorIn, HIGH);
   delay(1);
   pinMode(sensorIn, INPUT);
   digitalWrite(sensorIn, LOW);
   while(digitalRead(sensorIn)){
      duration++;
   }
   return duration;
}
```

We're looking at a function which returns the time **duration** for the circuit to respond to the light presented to the sensor. This code does the following

- Set the Arduino pin to output and set it HIGH. This discharges the capacitor.
- Wait 1ms to ensure the capacitor has had time to discharge.
- Set the Arduino pin to input and write a LOW to configure its internal operation.
- The while loop monitors the input voltage to the pin and loops until this is zero. During the loop, it increments the time **duration** variable.
- Finally, the function returns the value of **duration**.

In summary, our delicate sensor gives us a broad and continuous range of values from light (around 50) to dark (around 1000). This is an enormously useful range of values which we can easily use to compute the robot's error from the centre of a dark (or light) line.

Typical code to get the values from a left and right sensor and to compute some measure of error looks like this

```
senseL = RCTime(5);
senseR = RCTime(4);

error=((float)senseL-(float)senseR)/
((float)senseL + (float)senseR);
```

Note that senseL and senseR are **long** variable types and that these have been cast explicitly to **floats**. This is best programming practice. You may ask why the difference between the sensor reading is divided by their sum. Well, this is the expression

$$error = \frac{(sense_L - sense_L)}{(sense_L + sense_L)}$$

If the sensor values are the same (or close) this evaluates to close to zero. If one sensor value is large and the other is small then e.g., (1000 – 10)/(1000 + 10) = 0.98 which is less than 1. So this expression will produce error values in the range 0.0 – 1.0. This value is ‘normalized’ and it is very very useful to know it is in this range.