Aberystwyth Robotics Club - Elegoo Robot
Infrared Remote Control

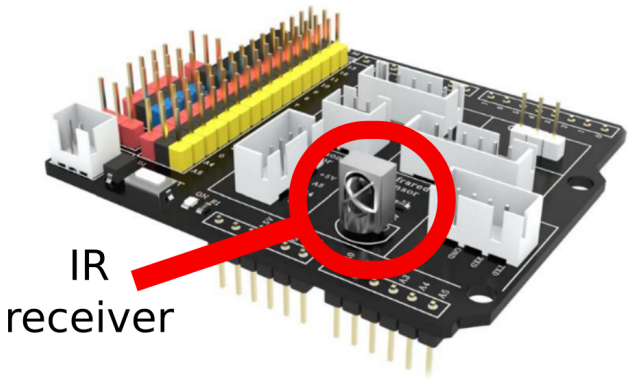
Introduction

This worksheet will enable us to control our robot using the infrared remote, changing the behaviour dynamically.

Infrared communication

Before we start, we need to add a library of code to communicate with the infrared.

1. Click on menu "Sketch"
2. Select "Include library"
3. Add .ZIP Library
4. Locate and add the provided "IRremote.zip" file.



Test program

To test that the library is loaded correctly and to help write our program, an example is provided with the library.

1. Click on menu "File"
2. Select "Examples"
3. Scroll down to "Examples from custom libraries" and "IRremote"
4. From the list of examples, select "IRrecvDemo"
5. Change the line "int RECV_PIN = 11;" to "int RECV_PIN = 12;" as our IR receiver is on a different pin.

6. Change the line "Serial.println(results.value, HEX);" to just "Serial.println(results.value);" removing the HEX parameter.

You should now be able to upload the program to your robot, and in the serial monitor see various numbers printed when pressing different buttons.

When a button is pressed on the remote, it sends a rapid burst of signals that are translated into a number to uniquely identify each button. Due to how this process works, there are two possible numbers for each button. For our program, we will want to make use of the direction arrows, the OK button and a couple of numbers. Draw up a table as follows:

Button	A	B
up arrow	16736925	5316027
...		
...		

Steering the robot

We can use the remote control to steer our robot. Below are the additions we need to make to our program:

1. At the top, add a line to include the IRremote library for program:

```
#include <IRremote.h>
```

2. Define the pin and some useful variables for interacting with the IR receiver:

```
#define RECV_PIN  12        // Infrared signal receiving pin
IRrecv irrecv(RECV_PIN);  // Specify the pin the IR receiver is
   connected to
decode_results results;   // A special variable provided by the
   library for interpretting the signals
unsigned long val;       // used to record the decoded value received
unsigned long preMillis;  // we will use this for a timer below
```

3. Define the set of codes recorded in the table above e.g.:

```
#define Fa 16736925    // FORWARD
#define Fb 5316027     // FORWARD
...
```

4. In the setup function, add the following line to start the receiver:

```
irrecv.enableIRIn(); // Start the receiver
```

5. In the loop function, if a signal is received we want to do something with it, otherwise we will stop the robot.

```
if (irrecv.decode(&results)){
  preMillis = millis();   //Record time at which signal was received
  val = results.value;    //The value indicating which button was
     pressed
  Serial.println(val);
  irrecv.resume();
  ...
}
else{ //Make sure the robot doesn't run away
  if(millis() - preMillis > 500){
```

```
      setStop();          //
      preMillis = millis();
    }
  }
```

To make sure the robot doesn't run away, we will tell it to stop half a second (500 milliseconds) after receiving the last signal.

6. To process the values and decide on an action we are going to use a new type of control structure called a Switch-statement. A switch statement takes a value and compares it against a series of cases. This is much like an if-statement, but is useful when there are pre-defined values as we have here. The outline structure is:

```
switch(val){  //input from remote
  case Fa:
  case Fb: setForward(); break; // without a break, it will continue
     to execute code for next case too
  case Ba:
  case Bb: setReverse(); break;
  ?????
  default: break; // end case for any other undefined input value
}
```

Fill in cases for the other buttons to drive the robot around with the remote. Remember the "break;" between cases for different buttons. Note that the button names cannot start with a numerical character, so for the number buttons you will need to write OneA... instead.

Changing modes

In the switch statement used above, we are calling functions to drive the robot. By putting the code for line following and obstacle avoidance into functions too, we can use those behaviours too.

Challenge: When line following or navigating a maze, we don't want to have to keep pressing the button every 500ms. Instead we want it to keep moving until instructed otherwise. However, if steering the robot we don't want it to run away. Consider how a function could be used to separate the steering.

Hint 1 Use the number buttons to set 'modes' of operation

Hint 2 Consider modes for line following, obstacle avoidance, maze navigation and steering

Hint 3 In the steering function use a loop to keep getting new readings until the 'stop' command is given

Hint 4 A 'while-loop' is like a 'for-loop' but just checks for a condition, e.g.:

```
while(!(val==stopA || val==stopB)){
  ...
 }
```

Where the ! means 'Not', == means 'equal to', and the || means 'OR', so can be read "While not value equals stop A or value equals stop B do...".