Collecting values of the receiver signal to the Arduino

Understanding the pulseIn () function

Reads a pulse (either HIGH or LOW) on a pin. For example, if value is HIGH, pulseIn() waits for the pin to go from LOW to HIGH, starts timing, then waits for the pin to go LOW and stops timing. Returns the length of the pulse in microseconds or gives up and returns 0 if no complete pulse was received within the timeout.

The timing of this function has been determined empirically and will probably show errors in longer pulses. Works on pulses from 10 microseconds to 3 minutes in length.

Syntax:- pulseIn(pin, value) ; pulseIn(pin, value, timeout)

Example Code:-

```
int pin = 7;

unsigned long duration;

void setup() {

  Serial.begin(9600);

  pinMode(pin, INPUT);

}

void loop() {

  duration = pulseIn(pin, HIGH);

  Serial.println(duration);

}
```

Understanding the attachInterrupt () function

The first parameter to attachInterrupt() is an interrupt number. Normally you should use digitalPinToInterrupt(pin) to translate the actual digital pin to the specific interrupt number. For example, if you connect to pin 3, use digitalPinToInterrupt(3) as the first parameter to attachInterrupt().

Inside the attached function, delay() won't work and the value returned by millis() will not increment. Serial data received while in the function may be lost. You should declare as volatile any variables that you modify within the attached function. See the section on ISRs below for more information.

Interrupts are useful for making things happen automatically in microcontroller programs and can help solve timing problems. Good tasks for using an interrupt may include reading a rotary encoder, or monitoring user input.

If you wanted to ensure that a program always caught the pulses from a rotary encoder, so that it never misses a pulse, it would make it very tricky to write a program to do anything else, because the program would need to constantly poll the sensor lines for the encoder, to catch pulses when they occurred.

Other sensors have a similar interface dynamic too, such as trying to read a sound sensor that is trying to catch a click, or an infrared slot sensor (photo-interrupter) trying to catch a coin drop. In all these situations, using an interrupt can free the microcontroller to get some other work done while not missing the input.

BOARD	DIGITAL PINS USABLE FOR INTERRUPTS
Uno Rev3, Nano, Mini, other 328-based	2, 3
UNO R4 Minima, UNO R4 WiFi	2, 3
Uno WiFi Rev2, Nano Every	All digital pins
Nano 33 IoT	2, 3, 9, 10, 11, 13, A1, A5, A7
Nano 33 BLE, Nano 33 BLE Sense (rev 1 & 2)	all pins
Nano RP2040 Connect	0-13, A0-A5
Nano ESP32	all pins

Using the above to read RC Receiver Signals using Arduino

In your Arduino sketch, you will set the pins you are using to read as input pins ***'pinMode(pin, INPUT)'*** and will read the value with the function ***'pulseIn(inputPort, HIGH, 2500)'***. This will return the duration of the pulse that, in practice, means a number between 1000 and 2000.

For negative values on your gimbals, we will read numbers from 1000 to 1499. For 0, we read 1500. For positive values we read numbers from 1501 to 2000. Of course, this can change a bit, depending on your transmitter configuration. Instead of dealing with this range we can use a very handy function, that translate one range to another ***'map(channelValue, 1000, 2000, -100, 100)'.***

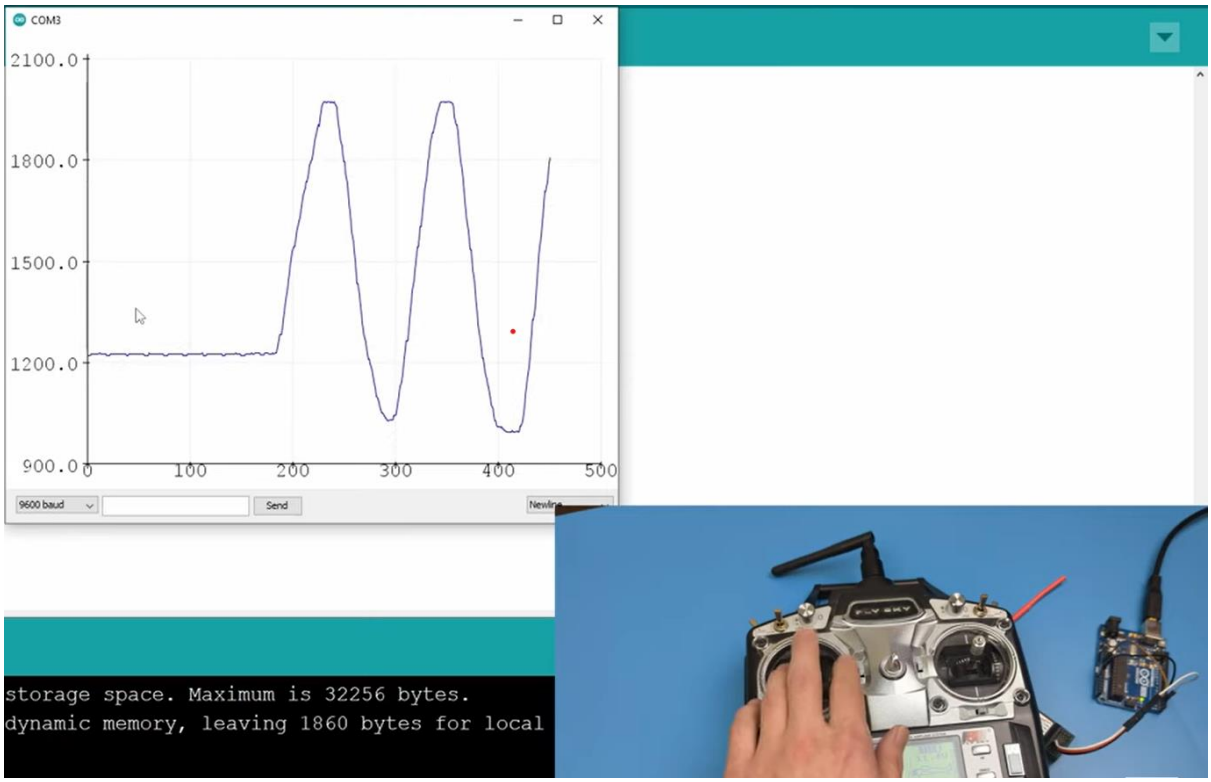
This will convert any input from the receiver to a proportional number between -100 and 100. If the receiver is off you will probably read the value 0, so we need to protect our code against that.

Basic RC Receiver PWM reader code

Code#1 (Being able to map the movement of the joystick via serial monitor)

```
#define RCPin 2

int RCValue;


void setup() {

Serial.begin(9600);

pinMode(RCPin, INPUT);

}


void loop()  {

RCValue = pulseIn(RCPin, HIGH);

Serial.println(RCValue);

}
```

Result#1

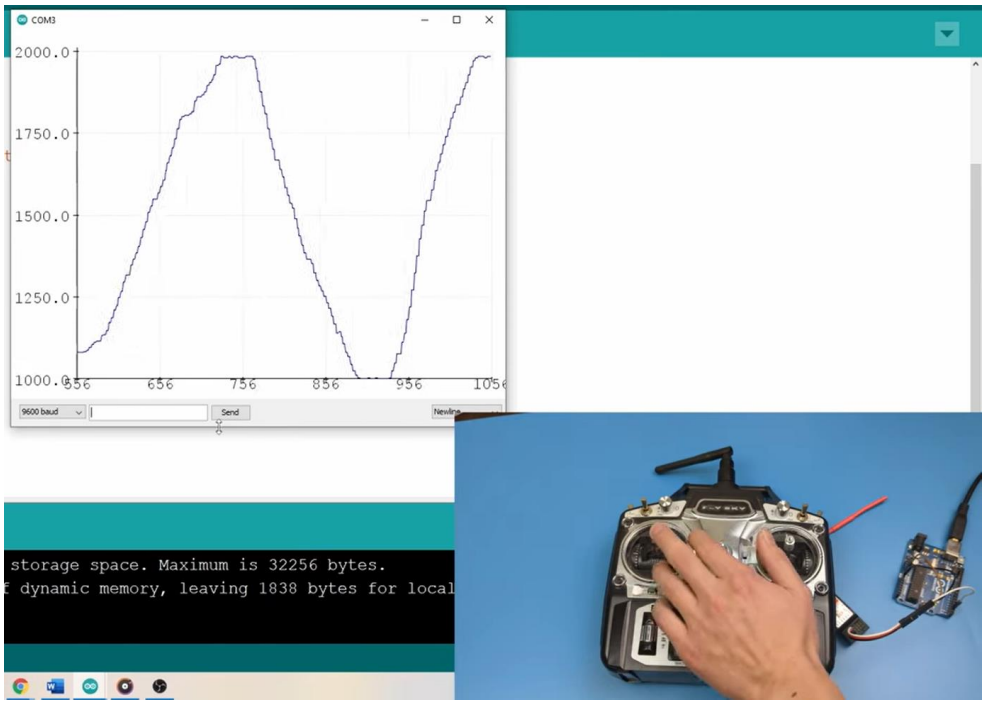


Code#2 (A more error free code without delays from the PulseIn function by using the attachInterrupt function)

```
#define RCPin 2

volatile long StartTime = 0;

volatile long CurrentTime = 0;

volatile long Pulses = 0;

int PulseWidth = 0;


void setup() {

serial.begin(9600);

pinMode(RCPin,INPUT_PULLUP);

attachInterrupt(digitalPinToInterrupt(RCPin),PulseTime,CHANGE);

}
```

```
void loop() {
if (Pulses<2000){
PulseWidth = Pulses;
}
Serial.println(Pulses);
}
void PulseTimer() {
CurrentTime = micros();
if (CurrentTime > StartTime){
Pulses = CurrentTime - StartTime;
StartTime = CurrentTime;
}
}
```

Result#2



Reading Multiple Channels of a RC Transmitter

The idea is to connect the receiver pins of each channel you want to read from, to a digital port in Arduino. It is very straightforward. The Arduino pins will act as input ports and the receiver needs to be powered with the 5v from Arduino too. This is the wiring schema to read the 5 channels from the receiver, the 4 channels for gimbals and 1 channel for an on/off switch.

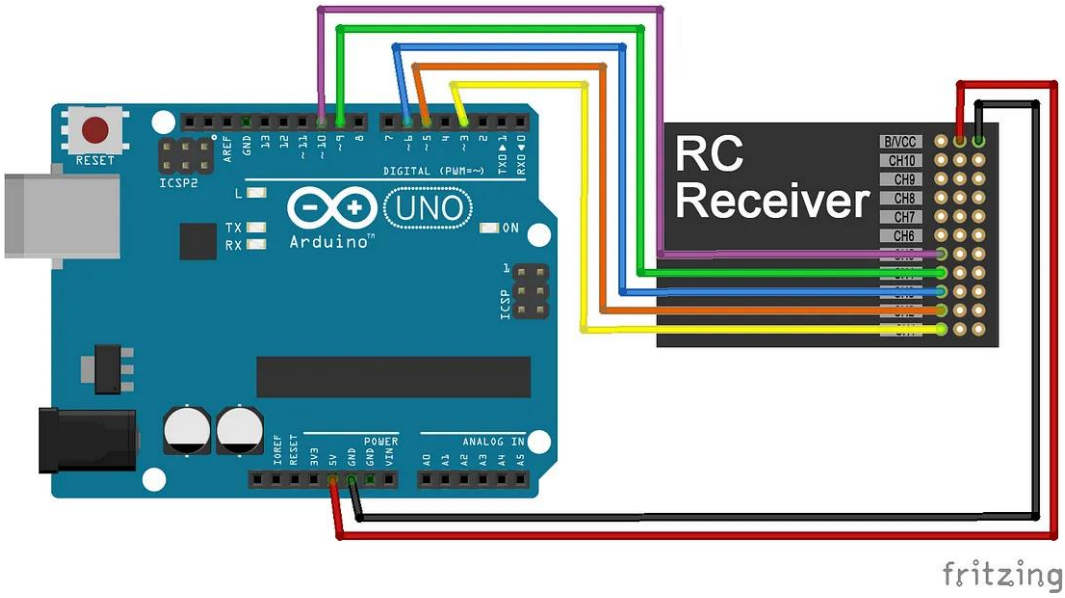
Another point of attention is the reverse feature from your transmitter. I used that for one of the available switches. As the transmitter enforces you to put all switches up before connect to the receiver, the 'off' value is the highest value (2000). So, you may think to revert this channel, to get 'off' value as the lowest value (1000). But if you turn on the receiver with the transmitter off by mistake, you may have an 'on' situation with no signal being read. I know you must never turn on your receiver with your transmitter off, but when you are developing, it just happens. So, it is better to prevent than use the reverse signal feature in your transmitter.

Code

```
#define CH1 3

#define CH2 5

#define CH3 6

#define CH4 9

#define CH5 10

// Read the number of a given channel and convert to the range provided.

// If the channel is off, return the default value

int readChannel(int channelInput, int minLimit, int maxLimit, int defaultValue){

  int ch = pulseIn(channelInput, HIGH, 30000);

  if (ch < 100) return defaultValue;

  return map(ch, 1000, 2000, minLimit, maxLimit);

}

// Red the channel and return a boolean value

bool redSwitch(byte channelInput, bool defaultValue){

  int intDefaultValue = (defaultValue)? 100: 0;

  int ch = readChannel(channelInput, 0, 100, intDefaultValue);

  return (ch > 50);

}
```

```
void setup(){
  Serial.begin(115200);
  pinMode(CH1, INPUT);
  pinMode(CH2, INPUT);
  pinMode(CH3, INPUT);
  pinMode(CH4, INPUT);
  pinMode(CH5, INPUT);
}
int ch1Value, ch2Value, ch3Value, ch4Value;
bool ch5Value;

void loop() {
  ch1Value = readChannel(CH1, -100, 100, 0);
  ch2Value = readChannel(CH2, -100, 100, 0);
  ch3Value = readChannel(CH3, -100, 100, -100);
  ch4Value = readChannel(CH4, -100, 100, 0);
  ch5Value = redSwitch(CH5, false);

  Serial.print("Ch1: ");
  Serial.print(ch1Value);
  Serial.print(" Ch2: ");
  Serial.print(ch2Value);
  Serial.print(" Ch3: ");
  Serial.print(ch3Value);
  Serial.print(" Ch4: ");
  Serial.print(ch4Value);
  Serial.print(" Ch5: ");
  Serial.println(ch5Value);
  delay(500);
}
```

Basic Circuit Connections



Using the above codes for turn movements of the RC Car

Now that we know how to read multiple channels of the receiver, we can code the basic movement of the RC Car (front, reverse, right and left) using multiple if statements where we can turn off and on certain motors depending on what signal the transmitter of the remote sends.

Code Final (Used in the competition)

```
double ch1=2;

int a=5; int b=6;

int enA=3;

double ch2=4;

int c=8; int d=9;

int enB=10;



void setup()

{

  Serial.begin(9600);
```

```
 pinMode(2,INPUT);
 pinMode(5,OUTPUT); pinMode(6,OUTPUT);
 pinMode(3,OUTPUT);

 pinMode(4,INPUT);
 pinMode(8,OUTPUT); pinMode(9,OUTPUT);
 pinMode(10,OUTPUT);
}

void loop()
{
ch1 = pulseIn(2,HIGH);
ch2 = pulseIn(4,HIGH);

if((ch1==0)&&(ch2==0))
{
 digitalWrite(a,LOW); digitalWrite(b,LOW);
 digitalWrite(c,LOW);digitalWrite(d,LOW);
 digitalWrite(enA,HIGH);digitalWrite(enB,HIGH);
 Serial.println("BOt is ON");
}

else if((ch1>1530)&&(ch2>1530))
{
digitalWrite(a,HIGH); digitalWrite(b,LOW);
 digitalWrite(c,LOW);digitalWrite(d,HIGH);
 digitalWrite(enA,HIGH);digitalWrite(enB,HIGH);
 Serial.println("MOVING FORWARD");
}

else if((ch1>1530)&&(ch2<1460))
```

```
{
  digitalWrite(a,HIGH); digitalWrite(b,LOW);
 digitalWrite(c,HIGH);digitalWrite(d,LOW);
 digitalWrite(enA,HIGH);digitalWrite(enB,HIGH);
 Serial.println("c3");
}

else if((ch1<1460)&&(ch2>1530))
{
  digitalWrite(a,LOW); digitalWrite(b,HIGH);
 digitalWrite(c,LOW);digitalWrite(d,HIGH);
 digitalWrite(enA,HIGH);digitalWrite(enB,HIGH);
 Serial.println("c4");
}

else if((ch1<1460)&&(ch2<1460))
{  digitalWrite(a,LOW); digitalWrite(b,HIGH);
 digitalWrite(c,HIGH);digitalWrite(d,LOW);
 digitalWrite(enA,HIGH);digitalWrite(enB,HIGH);
 Serial.println("MOVING BACKWARD");
}

else
{
 digitalWrite(a,LOW); digitalWrite(b,LOW);
 digitalWrite(c,LOW);digitalWrite(d,LOW);
 digitalWrite(enA,LOW);digitalWrite(enA,LOW);
 Serial.println("c6");
}
}
```