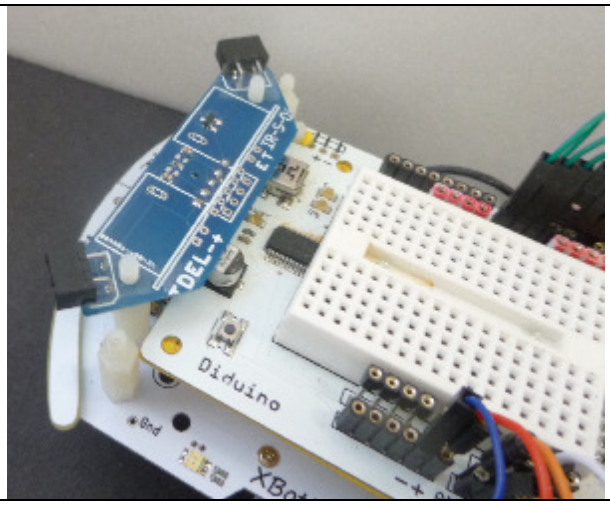
Infrared distance sensor Xdist2IR

Translated from https://www.didel.com/xbot/Dist2Ir.pdf

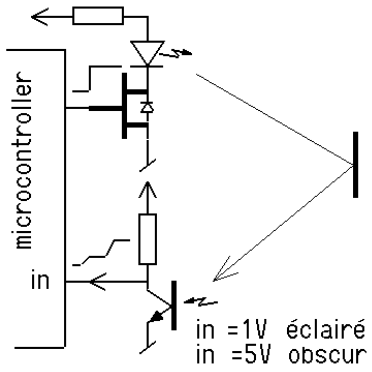
Infrared reflection sensors have the advantage of being small, inexpensive and easy to implement. But they are sensitive to ambient light (especially spotlights) and are difficult to calibrate. They are also suitable only for short distances, which depend on the size of the sensor, its optics, the power emitted, possible filters. General explanations are given in French under www.didel.com/doc/sens/DocIr.pdf

The Dist2R sensor was developed for the XBotMicro, to measure the distances to the right and to the left. It is easily used on another robot platform by pulling 4 or 5 wires to replace the connector below the circuit. Power supply is 3 to 5V 70 mA when the IR diodes are illuminated
A digital output controls the IR lighting (optional), and 2 digital inputs read the distance with a simple procedure.
The module has been designed to add an SR05 type ultrasound sensor. Two wires must be added toward two free pins, see SR05 documentation.



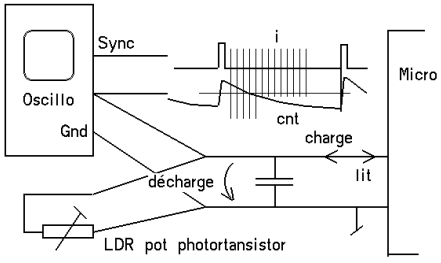
Principle

The principle of a reflection sensor is to illuminate the obstacle with an infrared LED, and to measure the reflected light with a photodiode or a phototransistor. The illuminated object retransmits an energy inversely proportional to the square of the distance.
One need a resistor to fix the current through the LED that illuminates the scene. The resistance of the phototransistor is usually measured with a voltage divider. For the measurement accuracy to be maximum, the measured voltage must be close to 2.5V. This involves a tuning potentiometer and a limited measuring range.



Measurement by discharge of a capacitor

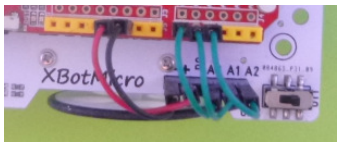
An elegant solution for measuring a variable resistor is to charge a capacitor in parallel with the resistor and measure the discharge time. The microcontroller charges the capacitor in a few microseconds. It must then switch the input port, and measure the discharge time till the logic level is zero. Less light mean higher resistance and longer discharge, but parasitic resistance and ambient light limit the measurable distance. The value of the capacitor is such that the processor counts the time with sufficient precision for a short time so that the robot does not move too much between two measurements. In the dark, the capacitor does not discharge, and a counter limits the measurement to for example a value of 100. If one counts every 100 microseconds, the duration of the measurement is 10 ms.

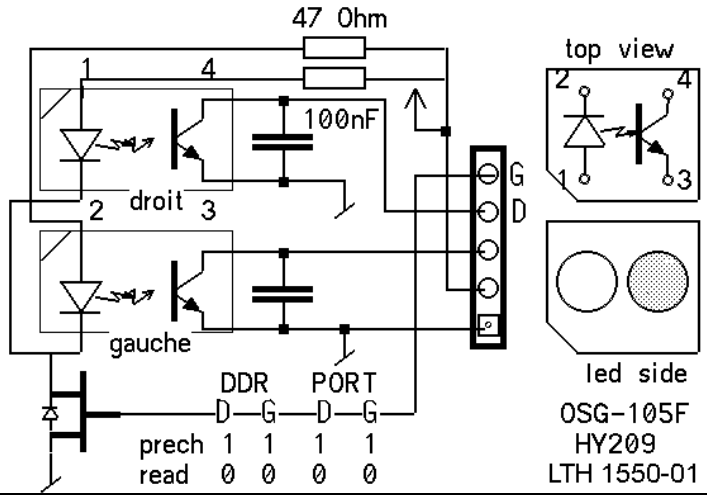


To reduce the effect of ambient light, we can make a first measurement without IR lighting, then a second measurement with lighting, the difference corrects a little, but what is important is to check that the ambient light gives a value very different from the value illuminated by IR. If not, the sensor is misdirected or must be protected by well positionned caches.

Schematic

The two IR diodes are in parallel, to be 3V compatible, the diode voltage drop is about 1.3V. A transistor controls the current, which must be important. The diagram gives the wiring on the connector. These signals are found on the front connector of the XBotMicro. The usual wiring uses Arduino pins 14 / A0,15 / A1,16 / A2 in digital mode.





Different wiring requires changing the pin's numbers at the beginning of the definition file.

Definitions

In addition to the definitions of the pins and their actions, for the Left and Right IR signals, two directions macros called DirCha (output for loading the capacitors) and DirMes (inputs for measuring the transition to the 0 state) are defined. They control the load / discharge mode for the capacitors. The set-up initialize the two G D bits on the PORTC.
The XBot documentation encourages C-compatible programming on other microcontrollers. The use of Arduino function works,but is slower.

```
// XBotDistIrDef.h
#include <Arduino.h>
#define bDistG  0 //PORTC  Arduino pin 14/A0
#define bDistD  1 //PORTC  pin15
#define bLedIr  2 //PORTC  pin16
#define LedIrOn   bitSet   (PORTC,bLedIr)
#define LedIrOff  bitClear (PORTC,bLedIr)
#define CapaCha  PORTC |= 1<<bDistG | 1<< bDistD \
                 PORTC &= ~(1<<bDistG | 1<< bDistD)
#define DirMes   DDRC  &=~(1<<bDistG| 1<< bDistD)
#define CapaGHigh PINC & 1<<bDistG
#define CapaDHigh PINC & 1<<bDistD

void SetupDistIr () {
   DirLedIr;
   DirMes;
   PORTC |= 1<<bDistG | 1<< bDistD
}
```

```
// XBotDistIrDef.h
#define DistG  14
#define bDistD 15
#define bLedIr 16
#define LedIrOn digitalWrite(LedIr,1)
#define LedIrOff digitalWrite(LedIr,0)
#define DirCha pinMode(DistG,1); pinMode(DistG,1);
#define DirMes pinMode(DistG,0); pinMode(DistG,0);
#define CapaGHigh digitalRead (DistG)
#define CapaDHigh digitalRead (DistD)

void SetupDistIr () {
   pinMode (LedIr,1);
   DirMes;
   digitalWrite(DistG,1);
   digitalWrite(DistG,1);
}
```

Program's source can be found under **https://www.didel.com/xbot/Dist2Ir.zip**

Software

For the measurement, one precharges during 100 us, then one switches in input mode, and one reads the value which decreases exponentially. As long as the threshold is greater than state 1 (about 1.9V to 5V) one counts. The counters are updated every 100 microseconds, depending on the capacitor value.

```
byte cnt=0, cntG=0, cntD=0 ;
for (byte i=0; i<MaxVal; i++) {
  if (CapaGHigh) cntG++ ;
  if (CapaDHigh) cntD++ ;
  delayMicroseconds (100);  // adjust together with MaxVal
}
```

Using Arduino micros() is not efficient for 2 channels.
If max distance is 100, the max acquisition time is10ms and the distance is about 60mm with a 100 µF capacitor. During this time, the robot will move 1 cm if it is fast (1m / s). Good.

One can increase the max count to 200. It will double the time and increase the distance if ambiant light is low. It is easy to modify the loop and get out when the two capa are discharged.

Test program

The program measures continously and displays the values on the terminal every second. This dissipates too much, 10 measures per second would be enough.
Instead of #include, one can of course insert the instructions of the definition file instead.
Program's source can be found under **https://www.didel.com/xbot/Dist2Ir.zip**

```
//TestDist1.ino
#include "XBotDistIrDef.h"

void setup() {
  SetupDistIr ();
  Serial.begin(9600);
}
// variables globales
byte mesureG, mesureD;
#define MaxVal 100  // max 250
void GetDist () {
   volatile byte cntG=0, cntD=0 ;
   DirCha ;  // precharge
   delayMicroseconds (100) ;
   LedIrOn ; DirMes ;
   cntG = 0 ;  cntD = 0 ;
   for (byte i=0; i<MaxVal; i++) {
      delayMicroseconds (100) ;
      if (CapaGHigh)  cntG++ ;
      if (CapaDHigh)  cntD++ ;
   }
   LedIrOff ;
   mesureG = cntG ;
   mesureD = cntD ;
}
```

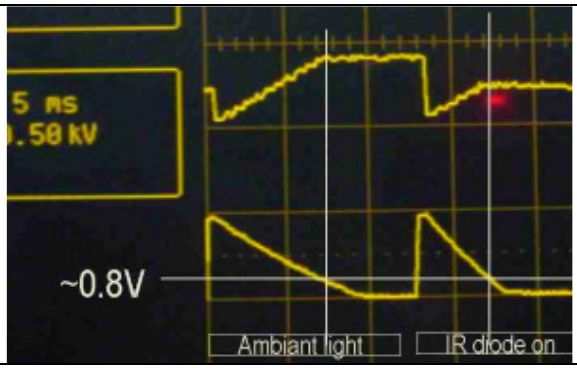
```
volatile byte cntAffi;
void loop() {
  GetDist () ;
 // tous les 100x 10ms, on affiche
  cntAffi ++ ;
  if (cntAffi > 100) {  // on affiche
    cntAffi = 0 ;
    Serial.print("Distance ");
    Serial.print(mesureG);
    Serial.print(" ");
    Serial.println(mesureD);
  }
}
```

Test distance max mesurées (valeur 100)

papier blanc 70 mm
papier noir 72 mm
boîte plastique jaune 48mm

La distance minimale est de ~10mm → valeur 2

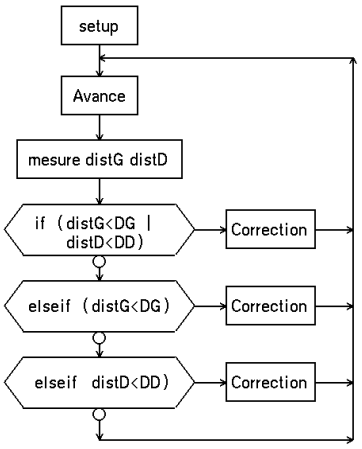
In this diagram, the program has been completed by transferring the 8-bit cntG value to a port having a digital-to-analog converter (Microdule DA8). The output of the converter shows the counting stairs, which stop when the capacitor voltage drops below 0.8V.
This also shows that the capacitor should discharge, not charge.



The ambiant light can be measured by removing the LedIrOn instruction. The TestDist1b.ino program measure illuminated and unlit; interesting to compare changing the orientation.

Avoid obstacles

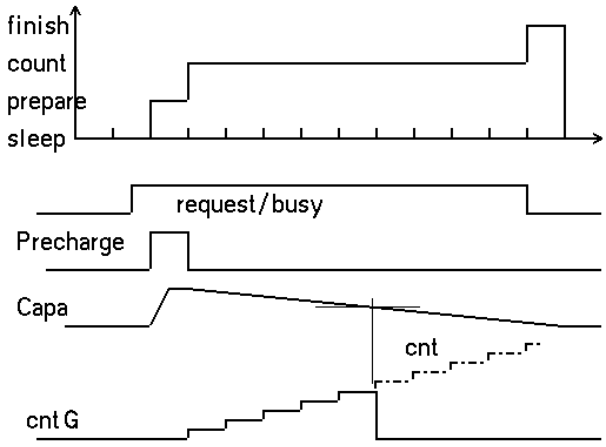
A simple solution based on obstacle avoidance with whiskers decides that there is an obstacle if the distance is less than a predefined value. The motors can be controlled in all or nothing, but a speed slowed by PWM may be preferable, to better adjust the parameters.
In the program, we added the XBot definitions and figured out how to roll back the robot with the Arduino PWM on 2 channels:
See www.didel.com/robots/MotorControl.pdf



Interrupts

To be called every 100 microseconds per interrupt, the GetDist () function must be transformed into a state machine.

```
volatile byte etat=0;
volatile byte cnt, cntG=0, cntD=0 ;
void GetDist () {
switch (etat) {
  case 0:
   CapaCha ;  DirCha ;  // precharge
   cntG=0; cntD=0; cnt=0;
   etat=1; break;
  case 1:
   LedIrOn ; DirMes ;
   cnt++;
   if (CapaGHigh)  cntG++ ;
   if (CapaDHigh)  cntD++ ;
   if (cnt>100) { LedIrOff ; etat=2; }
   break;
  case 2:
   mesureG = cntG ;
   mesureD = cntD ;
   etat=0; break;
  }  // end switch
}
```



The test that calls this every 100 microseconds is given in TestDist3.ino

Let's use the Timer2 to create an interrupt every 100 microseconds, as explained under
The call of GetDist () per interrupt takes ~ 5us, so slows down the main program by 5% (but not the delay () routine, handled by interrupt). During the interrupt, the PFM and other tasks will be managed as needed. The test program displays the values read on the terminal every second using a delay (1000).

The interrupt routine can now call this tested module without additional debugging.

```
ISR (TIMER2_OVF_vect)
{
  TCNT2 = 65;  //  100 us
    GetDist ();
}

void SetupTimer2() {
  cli();
  TCCR2A = 0; //default
  TCCR2B = 0b00000010;
  TIMSK2 = 0b00000001;
  sei();
}
```

The program is now clear with well-separated functions.

```
//TestDist4.ino  Distance par interruption
#include "XBotDef.h"
#include "XBotDistIrDef.h"
#include "XBotDistIr.h"
#include "ISRTimerDistIr.h"
void setup() {
  SetupXBot ();
  SetupDistIr ();
  SetupTimer2 ();
  Serial.begin(9600);
}
void loop() {
    delay (1000);// tous 1s, on affiche
    Serial.print("Eclaire ");
    Serial.print(mesureG);
    Serial.print(" ");
    Serial.println(mesureD);
}
```

Wall tracking

We have all the tools to program movements of the robot. Take the example of following a wall. It is necessary to enslave the pfm on the distance of the right wall. The measured value is between 20 and 100 (10 to 70mm). If it is greater than 50 (far), slow down the right engine. less than 40, it must be accelerated.

The program is simple using the X library by limiting the value of the sensor to the max value of the pfm.
TestDist1.ino add

```
void loop() {
  pfmD = distIrG; // braitenberg
  pfmG = distIrD;
}
```

Obviously, there is half a page of declaration and 5 inserted files.
Video underhttps://www.youtube.com/watch?v=AULplRbHatU
Using libX www.didel.com/xbot/LibXDist2Ir.pdf
Program under www.didel.com/xbot/LibXDist2Ir.zip