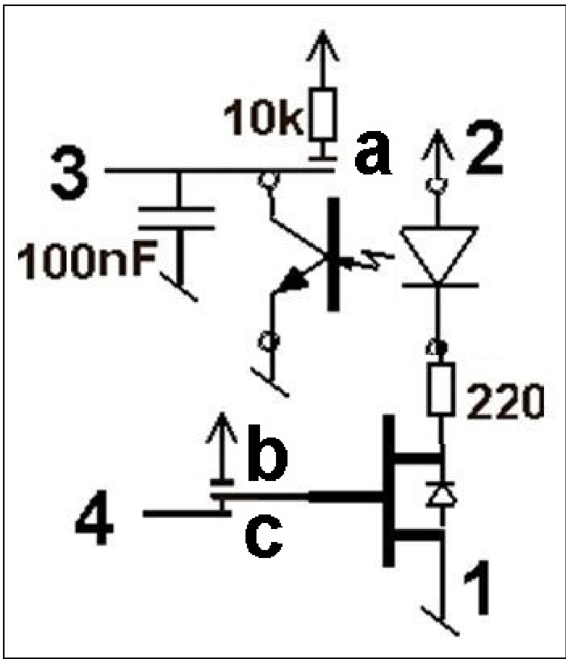
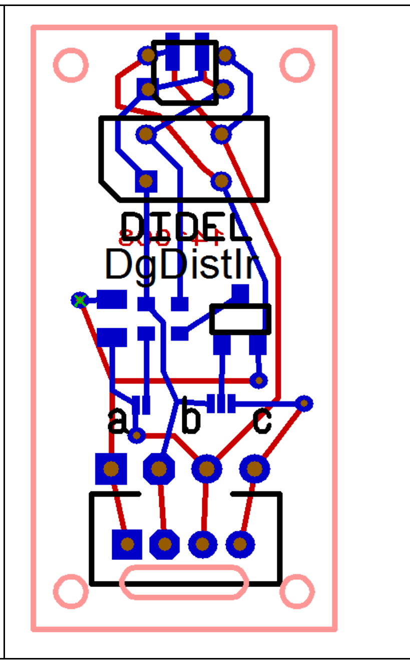
Digrove DgDistIr

Distance sensor till 20cm





Plusieurs documents expliquent le principe et documentent des kits similaires:
www.didel.com/coursera/LC7.pdf www.didel.com/xbot/xDist2Ir.pdf www.pyr.ch/coursera/CaptDistIrDoc.pdf
Les programmes de la partie qui suit sont faciles à comprendre.

The DgDistIr is an efficient device for measuring the distance of a reflecting media up to 1m in a not well eclaired room. The circuit has several options. Let's see one at a time. The photo reflective sensor is a LIT301/FR115, but is is possible to solder a miniature OSG105F if the detecting application is in the 0-50mm range.

.

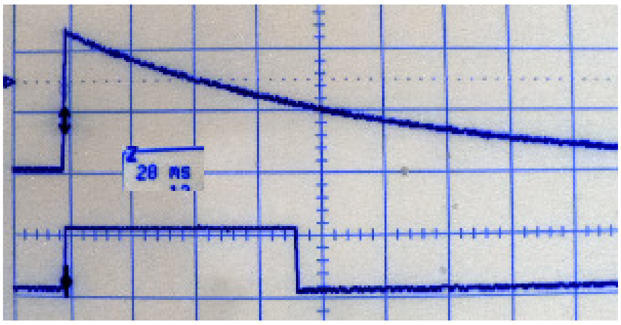
The DgDistIr has one output and one input. The input controls a transistor that power the IR led. Limiting resistor is 22 Ohm, a good compromise between max distance and power consumption. Control of the transistor is made from pin 4 through a solder drop on jumper **b**. A jumper allows to power continuously the transistor and have that input free. Put the solder drop on side **c**.

The phototransistor can be read by 2 different ways. The naive approach, that works in a limited range, connect a pull-up resistor to the phototransistor. On DgDistIr a 10 kOhm resistor is connected if you close the jumper **a** with a drop of solder. The middle point voltage is read on , See www.didel.com/doc/sens/DocIr.pdf (in french) to understand the limitation of that solution, not recommended for measuring distances, but OK for on/off control after adjustment.

The efficient way to read the light intensity is to load a capacitor and have it discharged by the photo-transistor. One measure the time until the signal is read as a "zero", that is a voltage lower than ~2.3V. The advantage measuring time is one can cover several decades.

Tests

Microcontrollers lines are bidirectionnal. Let us charge the capacitor with a "1" (5V) and then switch the line to an input.

The charging pulse of 50 microseconds is repeated every 100 ms. Inbetween, one measure input HIGH duration without pull-up.
`ChargeMode;`
`MeasureMode;`
`delay (100);`
 	
If you do not have a scope, you can copy the signal to Led13. Blink is shorter if more light.
`ChargeMode;`
`MeasureMode;`
`while (CapaHigh)  { Led13On; }`
`Led13Off;`
	Test1 If you have a scope, you see this
	

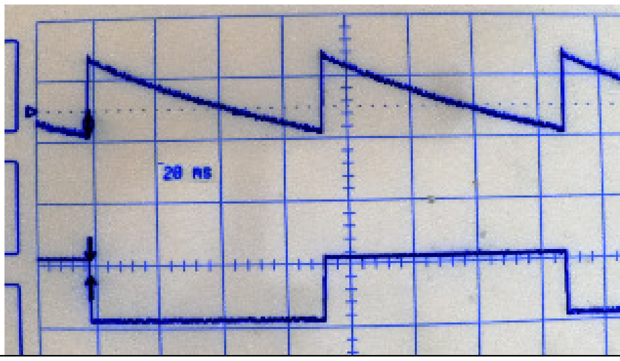
This is a C program, not a text! But the explicit names are prepared by a set of #define.

"pure" Arduino	C under Arduino or other compiler
Capa is connected to pin 14, PortC bit 0. Charge; set the direction out and active the line.	

`#define  Capa  14`
`#define  ChargeMode  digitalWrite (Capa,1);`
`    pinMode (Capa, OUTPUT)` *– on 1 line*
`#define MesureMode  digitalWrite (Capa,0);`
`    pinMode (Capa, INPUT)` *– on 1 line*
	`#define  bCapa  0 //PortC`
	`#define  ChargeMode  bitSet (PORTC,bCapa);`
	`    bitSet (DDRC,bCapa);`
	`#define  MesureMode  bitClear (PORTC,bCapa);`
	`    bitClear (DDRC,bCapa);`

Note: #define accept a list of instructions on the same line (not here for typographic reasons). It is not recommended to do more than 2. The last instruction has no ; In order to decide if the capa is still charged, one read the line. Above 2.7V, it is a HIGH	
`#define  CapaHigh  digitalRead(Capa)`	`#define CapaHigh PortC&(1<<bCapa)`
We need also to control Led13 as a debugging help	

`#define  Led13  13` // pin Arduino
`#define Led13On digitalWrite (Led13,1)`
`#define Led13Off digitalWrite (Led13,0)`
`#define Led13Toggle digitalWrite (!digitalRead(Led13))`
	`#define  bLed13  5` // PortB bit 5
	`#define  Led13On bitSet (PORTB,bLed13)`
	`#define  Led13On bitSet (PORTB,bLed13)`
	`#define  Led13Toggle PORTB^=(1<<bLed13)`

There is no need to wait when the capa is below the level. The led13 on Test2 change after every measure. Blinking is faster if more light:
`ChargeMode;`
`MeasureMode;`
`while (CapaHigh) {}`
`Led13Toggle`
With a 10nF capacitor, blinking may be too fast.
These programs (Test1, Test2 and followings) are available on www.didel.com/digrove/DgDistIr.zip
	Test2
	

Measuring distance

During the `while (CapaHigh)` it is easy to count and save the "distance" at the end.

Test3.ino	Test3b.ino //Variant

`ChargeMode;`
`MeasureMode;`
`while (cnt<100) {`
`  if (CapaHigh) { cnt++; }`
`  delayMicroseconds (100);`
`}`
`distance = cnt;`
`Serial.println (distance);`
	`ChargeMode;`
	`MeasureMode;`
	`while (CapaHigh) {`
	`  cnt++; delayMicroseconds (100);`
	`  if (cnt>100) break;`
	`}`
	`distance = cnt;`
	`Serial.println (distance);`

`Test4.ino` implement a Measure() function that update the distance variable, and is easy to call in your application. Of course, déclarations and set-up must be adapted.

This is a blocking program, with a 100 nF capacitor, it takes 50 ms and more if light is low.

There are 2 ways to accelerate:

1) reduce the capacitor value and the delays in measuring loops or max value.

2) increase the power of the IR diode.

On can use millis() to measure the duration of the pulse.
We see later how to use interrupts.

Use of millis()
Instead of counting, it is easy to measure the discharge time

```
Test3c.ino
   ChargeMode;
   MeasureMode;
   now = millis();
   while (CapaHigh) { ; }

   distance = millis() - now;
   Serial.println (distance);.
```

It gets more complicated if you need a time-out for too long discharges in the dark.

Ambiant light
A transistor controls the IR diode. This is not only to set current only during mesures, but also for measuring ambiant light, with no power on the IR led. If ambiant light is important, distance measure is biased. One can subtract ambiant light to the measure, it does work in a small range. The main interest of measuring ambiant light is to have an idea if the light conditions correspond to what has been defined.
Modify Test2.ino and Test3.ino to alternate measure with and without IRled active. (see Test2alt.ino and Test3Alt.ino on the zip).

Several sensors
It is easy to control several sensors in the loop of Test3 program. One counter define a maximum duration and one counter for every sensor is stopped when the corresponding capacitor is discharged.

```
   ChargeMode;         // charge all
   MeasureMode;        // all are inputs
   while (cnt<100) {
     if (Capa1High) { cnt1++; }
     if (Capa2High) { cnt2++; }
     if (Capa3High) { cnt3++; }
     delayMicroseconds (100);
   }
   distance1 = cnt1;
   distance2 = cnt2;
   distance3 = cnt3;
```

If you use plain Arduino, you have to create a function for Charge() and Measure(); #define are not comfortable for more than 2 instructions. If all capacitors are iôn the same port, it is easy to declare in C with appropriate logic operations.

Multitask
It is clear that interrupt (PWM, time, ...) add jitter to the measure. This will noticable only if the count is low, and anyway precision of the measure is rather good.
If one have to do something else during the measure, one can replace the delayMicroseconds (100) by calling a function that is calibrated to take about the same time, doing the required tasks.
Using the Arduino micros() function, if you are familiar, is not efficient. One need anyway to test the capacitor voltage and measuring the time add more instructions and jitter than a counter.

Interrupt
A timer can be devoted to counting the time. Every 100 us, the timer interrupt routine test the capa level and count or activate a flag to say the count value is the measure. Clearing the flag start a new measure, that is the interrupt routine is a state machine with 3 states: Charge, Measure, Wait on flag to be cleared.

100 us Timer1 Interrupt
Multi-interrupts are difficult to master. A single interrupt every 100 us can service many tasks, as shown in www.didel.com/xbot/Interruption100us.pdf (in French, will be translated if interest is shown).

In order to service two sensors two 8-bit global variables are updated every 40ms and give distance values between 0 and 255.
The state machine to be called by the interrupt every 100 us is given next.
Definitions and set-up are given below:

```
//DistIr.h  Libraire Uson pour
interruption
#define bDistG   0 //PORTC  pin14
#define bDistD   1 //PORTC  pin15
#define bLedIr   2 //PORTC  pin16
#define DirLedIr  DDRC |= 1<< bLedIr
#define LedIrOn   bitSet   (PORTC,bLedIr)
#define LedIrOff  bitClear (PORTC,bLedIr)

#define mCadist (1<<bDistG | 1<< bDistD)
#define ModeCha  DDRC  |=  mCadist; PORTC |=  mCadist
#define ModeMes  DDRC  &= ~mCadist; PORTC &= ~mCadist
#define CapaGHigh PINC  & 1<<bDistG
#define CapaDHigh PINC  & 1<<bDistD

void SetupDistIr () {
   DirLedIr;
   LedIrOn;
   ModeMes;
}
```

```
volatile byte cnti, cntG,cntD;
enum {Istart,Iatt,Icnt,Ifin}
eIr=Istart;
void DoDistIr () {  // cycle de 17us
 cnti++;
 switch(eIr) {
   case Istart:
   ModeCha;  // precharge
   cnti=0;
   eIr= Iatt;
   break;
  case Iatt:
   ModeMes;
   cntG = 0 ;  cntD = 0 ;
   eIr= Icnt;
   break;
  case Icnt:
   if (CapaGHigh)  cntG++ ;
   if (CapaDHigh)  cntD++ ;
   if (cnti==0) eIr= Ifin;
   break;
  case Ifin:
   DistIrG = cntG;
   DistIrD = cntD;
   eIr = Istart;
   break;
 }  // end switch
```

Using miniature sensors
It may be useful to detect distances close to 1mm. The DistIr PCB can accept the miniature OSG-105F sensor horizontal or at the edge of the PCB.