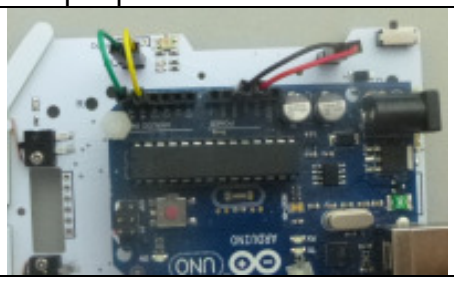
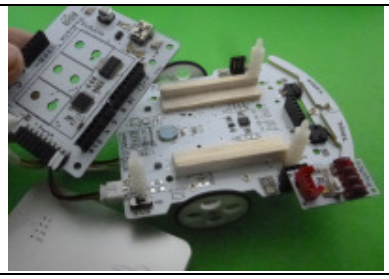
Xplus – an efficient mobile robot platform controlled by I2C

This document describes the software control of the Xplus.
Refer to www.didel.com/XplusConnect.pdf for the wiring options with Arduino and Raspberry.
Sample pictures:


Xplus is an I2C slave controlled by 14 commands. Transfers are compatible with SMbus/I2C. Use Wire lib on Arduino, Python lib on Raspberry.
Depending on the command, 8 or 16bit transfers are executed, read or write. Block transfers are possible on Python. I2C 7-bit address is 0x20. It can be changed, but not so easily for pedagogical reasons.
The function for a generic 16-bit transfer with Wire lib is

```
void Xwrite16 (int8_t cmd,  int16_t data16)  {
    Wire.beginTransmission(AdX;
    Wire.write(cmd);
    Wire.write(data16>>8);
    Wire.write(data16&0x00FF);
    Wire.endTransmission();
}
```

As example, for setting the motors speed, one need to send the 2 bytes, speedL and speedR, in this order, preceeded by command 3.
`Xwrite16 (3,(speedL<<8)+speedR);` will do the job.
We have defined easy to remember functions within the Xplus.h "library".

```
WriteSpeed (spedL,speedR);
```

You can redefine our names to be closer from your programmong habits (modify Xplus.h).

Programming model

This is the survey of all the commands, detailed later.

Command	Read	Write	Function
0 1byte	**Id=0xD0**		`id = ReadId ();`
1 1byte	**Status** **bit 0** **bWhiskL** **bit 1** **bWhiskR** **bit 6** **bSpeedOkL** **bit 7** **bSpeedOkR**		**`stat = ReadStatus ();`**
2 2bytes		**PfmLR**	`WritePfm (pfmL,pfmR);`
3 2bytes		**SpeedLR**	`WriteSpeed (spedL,speedR);`
4 2bytes	**PosL**		**`posL = ReadPosL ();`**
5 2bytes	**PosR**		**`posR = ReadPosR ();`**
6 4bytes	**PosBlk**		
7 4bytes		**Tell**	`WriteTell (v16);`
8 1byte		**ModeSens** **bit 0** **Ana01** **bit 1** **Ana23**	`WriteModeSens (mode);`

		bit 2	Uson etc	
9 2bytes	**A0 A1**			`a0a1 = ReadA0A1 ();`
10 2bytes	**A2 A3**			`a0a1 = ReadA0A1 ();`
11 1byte	**Uson**			`dist = ReadDistUson ();`
12 1byte	**DistIr**			`dist = ReadDistIr ();`

cmd #0 - Identifier

The identifier is usually read with direct access (address read, data) in the initialization process of a complex system. It can also be read by the command 0.

Arduino	`id = ReadId();`
Python	`id = read_byte (AdX)` (Only for Identifier)

cmd #1 - Status

This order gives the state of the whiskers. If left whisker is depressed, bit 1 of status variable read is 1. If right whiskers, bit 0 is on. If both, one read 0b11 = 3.

Arduino	`whisk = ReadStatus();`
Python	`whisk = read_byte_data (AdX,1)`

Bits 6 and 7 are used in conjunction with the speed control (see cmd #3).

Exemple Arduino	`whisk = GetStatus();`
	`if (whisk &(1<<bWhiskL) {`
	... avoid left obstacle
	`}`

cmd #2 – Pfm

Command 2 write the two byte that define the speed of the motors:

pfmL pfmR, value –128 à +127

Arduino	`WritePfm (pfmL,pfmR);  //-128 .. +127`
Python	`write_word_data (AdX,2,pfmLR)`

One need to wait 60 us before sending a new cmd #2 or #3

Exemple Arduino	Speed is increased up to the maximum, and then a sharp stop
	`for (byte i=0;i<128,i++) {`
	`  WritePfm (i,i);`
	`}`
	`WritePfm (0,0); // motor stop`
	`for(;;);  // program stop`

cmd #3 - Speed

Command 3 uses a logical speed, only 20 discrete values

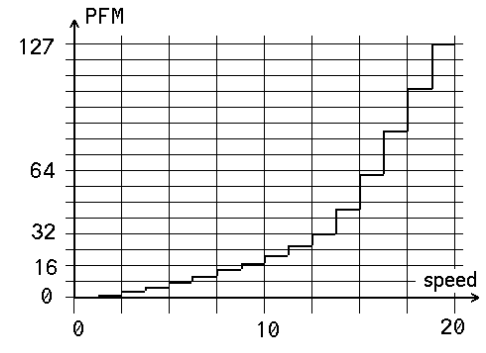
Arduino	`WriteSpeed (speedL,speedR); // -20 ..+20`
Python	`write_word_data (AdX,3,speedLR)`

One need to wait 60 us before sending a new cmd #2 or #3

Speed has 20 values that defines PFM values according an exponential correspondance, as shown on the drawing.

This is the table

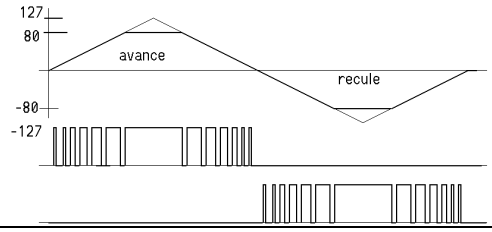
```
volatile int8_t taSpeed[]= { \
-127,-113,-99,-85,-72,-61,-51,-42,-35,-29, \
-24,-20,-17,-14,-11, -9,-7,-5,-3,-1, \
  0, 1, 3, 5, 7, 9,11,14,17,20,24, \
 29,35,42,51,61,72,85,99,113,127}; }
```

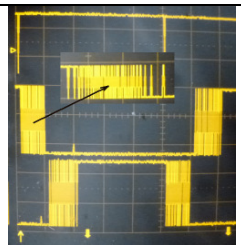


Speed adds a nice feature to your robot: constant acceleration. There are gradual transitions from one speed to another (2 seconds to switch from +20 to -20).

The following program demonstrates the progressive variation of speed.

```
void loop () {
    spd = 20;
    WriteSpeed (spd,spd);
     delay (1000);
    spd = -20;
    WriteSpeed (spd,spd);
     delay (1000);
} // end loop
```





Test to be done:replace SetSpeed (spd) ; by SetPfm (pfd) ; pfd = 127; (same max speed) and see the interest of constant acceleration mode.

cmd #4 - PosL cmd #5 - PosR Encoder position

The encoders (48 pulses per turn) mesure distances multiple of 2mm. The up-down counters posL posR are 16 bits signed variables transferred by the commands of the same name.. Max distance is 60 metres. One can write the position encoders any time.

Arduino	`posL = ReadPosL();`
	`WritePosL (newPosL);`
	`posR = ReadPosR();`
	`WritePosR (newPosR);`

Python	`read_word_data (AdX,4,newposL)`
	`write_word_data (AdX,4,newposL)`
	`read_word_data (AdX,5,newposR)`
	`write_word_data (AdX,5,newposR)`

Arduino example: One need to do move 100mm, that is 50 steps. Counters are cleared, motors are started. One test the distance in a loop and stops the motors when distance is reached.

```
#define Dist 100/2
WritePosL (0);
WritePosR(0);
#define Speed 20
WriteSpeed (Speed,Speed);
while (1) {
   if (ReadEncoL() > Dist) {WriteSpeed (0,Speed);}
   if (ReadEncoR() > Dist) {WriteSpeed (Speed,0);}
}
```

Comment: We hope the 2 motors of your robot have a difference in their characteristics: the movement will not be done in a straight line. The way the program is written, it is clear that one motor will stop before the other, giving a bad final angle to the robot!. Do improve by testing only one distance to and stop both motors, or add the distances.

Using the encoder to control the speed, hence the direction, is probably not possible due to the low resolution of the encoder. And it is not an important objective. Do not hope for a robot that move precisely. Close you eyes and move in a straight line, counting your steps. Did you reached your target?. Robots also need the information of environment sensors to behave correctly.

cmd #6 - Block read of PosL PosR (4 bytes) or block write

Read a block is easy with Python. The command asks the 4 bytes that correspond to the two positions. (not tested). That command can be used with the Wire library, if one dummy read cycle (block length of 4) is inserted before reading the 4 bytes.

Arduino	One can define a read or write of 5 bytes (length #4 received or send before the 4 significant bytes)
Python	`write_block_data (AdX,6,newposLR)` `posLR = read_block_data (AdX,6)`

cmd #7 - Tell

The optional DiTell display on its special connector is usefull to show variables, sensors, program state. It is an efficient aid to debugging,

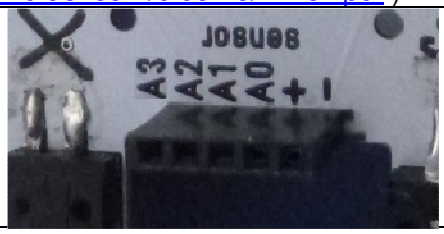
Arduino	`WriteTell (v16);`
Python	`write_word_data (AdX,7,v16)`

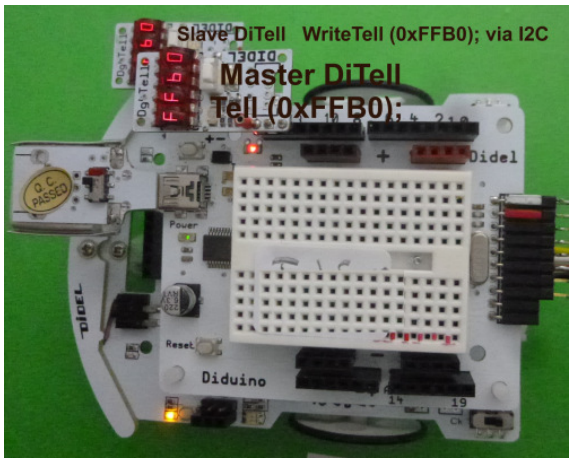
Master can also get a DiTell display

You will notice on some sample programs the `Tell(v16);` function that display a number on the master board.

`WriteTell (v16);` calls for an I2C transfer with command #7.

Tell (V16); is a blocking function that transfer the data on pin 13 of any Arduino board (see http://www.didel.com/diduino/DiTell.pdf)





The next commands control the Xbot sensors plugged on the front connector. The ModeSens variable must be set according to what has been plugged in.

cmd #8 - ModeSens Selection of the Xsens sensor on front connector

Arduino pins 14 à 17 (A0 à A3) are available and can be used for any sensor with analogue inputs. Xbot sensors Uson, DistIr, Servos are supported also with several combinations of functions.

Arduino	`WriteModeSens(v8);`
Python	`write_byte_data (AdX,8,v8)`

ModeSens		
0 Default, all inputs	3 SelUson	6 SelDistIr
1 SelAna01	4 SelUsonServos na	7 SelDistIrUson
2 SelAna23	5 SelUsonAna23	8 SelDistIrServo na
		9 SelDistIrAna23
10 SelOut, all outputs	Na not applicable yet	

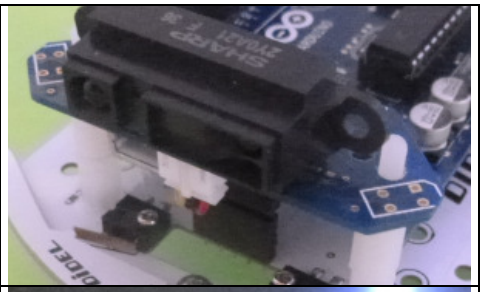
Mode 0 does not provide access to PortC<0..3>. These modes may be redefined.

cmd #9 et #10 - An0-An1 An2-An3

The pins of the front connector are analog inputs by default, and their 8-bit values are made available with **cmd #9 et #10.**

Xsensors ‘’Suivi’’, ‘’Piste’’, ‘’PSD’’ use analogue inputs and do not have special command.

Note the 2 power pins on the front connector. It makes it easy to connect a protentiometer or any smart sensor you dream of.

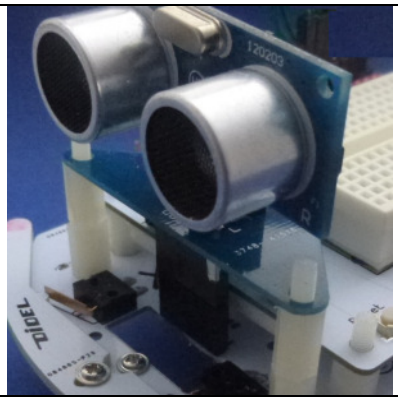


Arduino	`v16 = GetAna01 ();config: WriteModeSens (SelAna01);` `v16 = GetAna23 ();config: WriteModeSens (SelAna23);`
Python	`v16 = read_word_data (AdX,9)` `v16 = read_word_data (AdX,10)`

One need to wait 60 us before sending a new cmd.

cmd #11 Ultrasonic sensor SR05/SR04

Trig output must be on PortC pin 1 = A1, Echo input on A0. Direct connection on front connector is not possible.



Arduino	`v8 = ReadDistUson ();` `config: WriteModeSens (SelUson);`
Python	`v16 = read_byte_data (AdX,11)`

Wait 60 us before sending the next command

The distance is measured every 50ms. Unit is 1cm. If the sensor is not installed, this distance is null.

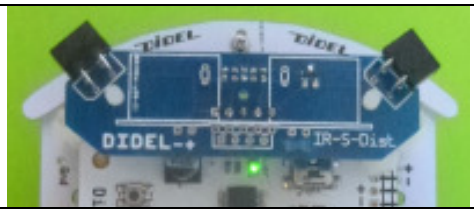
cmd #12 DistIr sensors

The two sensors are connected on pin A0 and A1.

Arduino	`v16 = ReadDistIR ();` `config: WriteModeSens (SelDistIr);`
Python	`v16 = read_word_data (AdX,12)`

Wait 60 us before sending the next command

cmd #12 returns left and right distances, measured every 50ms. Values are 2 (1-2 cm) to about 50 for a 10-20 distance, depending on the ambient light and the obstacle IR albedo.



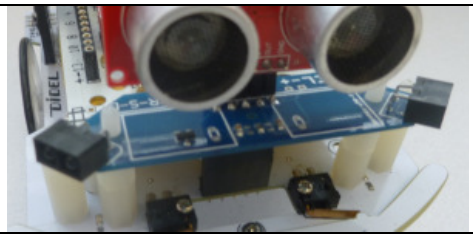
cmd #11 and #12 Ultrasonic and DistIr sensors

Arduino	`v8 = ReadDistUson ();`
	`v16 = ReadDistIR ();`
	`config: WriteModeSens (SelDistIrUson);`

Python	`v8 = read_byte_data (AdX,11)` `v16 = read_word_data (AdX,12)`

Wait 60 us before sending the next command

The DistIrUson sensor accept both sensors. The software redirect the Trig and Echo signals on pins A2 A3.



cmd #13 et #14 Servos
One or two servos can be connected on the A2 A3 pins. There are 25 discrete positions.

Arduino	`WriteServo1 (v8);`
	`WriteServo2 (v8);`
	`config: WriteModeSens (SelServo01);`

Python	`write_byte_data (AdX,13,v8))` `write_byte_data (AdX,14,v8))`

Too few sensors? Yes, we are limited by the number of pins around the AVR328.
See the X+Go, same I2C commands, but 3 microcontrollers on board control the sensors, the motors/encoders and a display.

Xplus.h Arduino/C definitions and functions

Definitions

```
#define AdX 0x20
#define Id 0  // commands
#define Status 1
 #define bWhiskL 0
 #define bWhiskR 1
 #define bSpeedOkL 6
 #define bSpeedOkR 7
#define Pfm 2
#define Speed 3
#define PosL 4
#define PosR 5
#define BlkPos 6
#define Tell 7

#define ModeSens 8
 #define SelAna01 1
 #define SelAna23 2
 #define SelUson  3
 #define SelUsonAna23 5
 #define SelDistIr 6
 #define SelDistIrUson 7
#define Ana01  9
#define Ana23  10
#define Uson   11
#define DistIr 12
#define Uson   13
```

Fonctions and examples of use

```
v8 = GetId();
v8 = ReadStatus();

WritePfm (pfmL,pfmR);   //-128 .. +127
WriteSpeed (speedL,speedR); // -20 ..+20

v16  = ReadPosL ();
v16  = ReadPosR ();
WritePosL (v16);
WritePosR (v16);

WriteTell (v16++);

SetModeSens (SelUson);
v16  = ReadAna01();
v16  = ReadAna23();
v8   = ReadUson ();
v16  = ReadDistIr ();

Tell (v16) ; local on master, not I2C
```

Jdn 160608