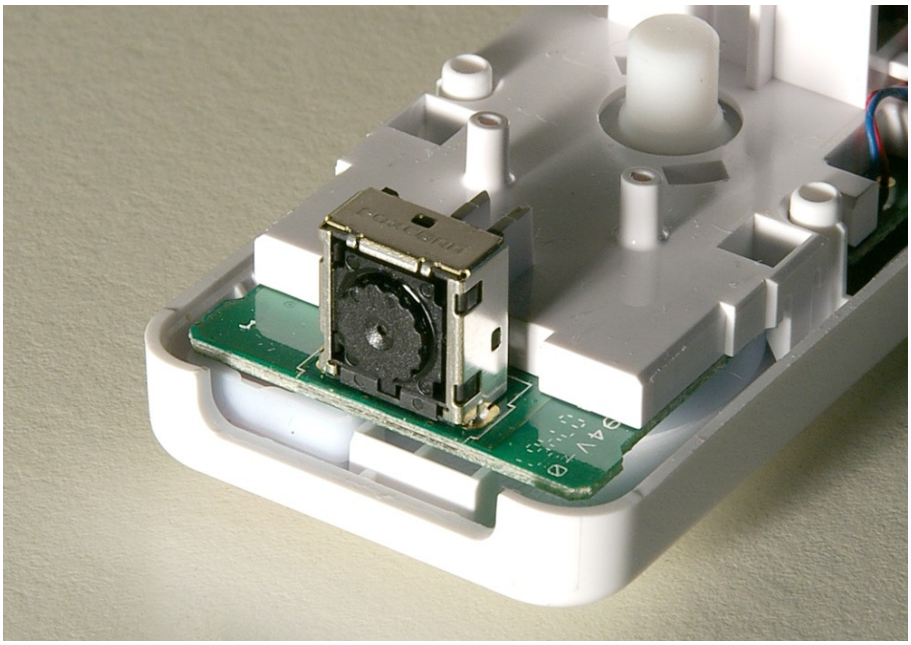
SOBRERO Mathieu
ROCHE Stéphane

Groupe F

COMPTE RENDU E&R_II
HACK CAMERA DE LA WIIMOTE



SOMMAIRE

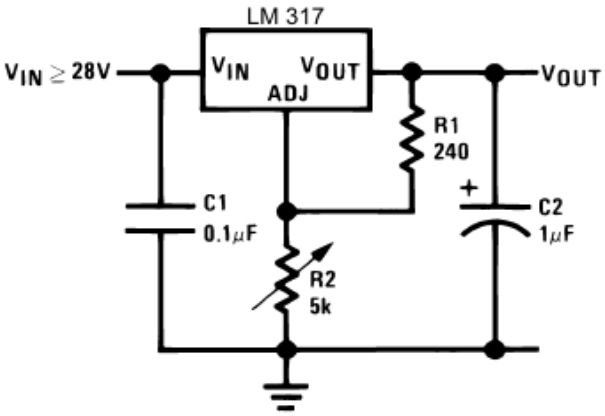
Introduction:
Le contenu de notre travail à consisté à mettre en œuvre une partie matérielle (vrapage d'une carte) et logicielle (programmation d'un microcontrôleur PIC), pour commander la caméra de la wiimote. C'est une caméra infrarouge PIXART possédant un processeur intégré qui délivre les positions X et Y des points pouvant aller de 1 à 4 simultanément.

I-La partie hardware

A/L'alimentation

La caméra fonctionne avec une alimentation de 3,3V. Nous avons alors câblé un régulateur LM 113 qui fournira cette tension pour satisfaire à cette condition primordiale.

Schéma de câblage:

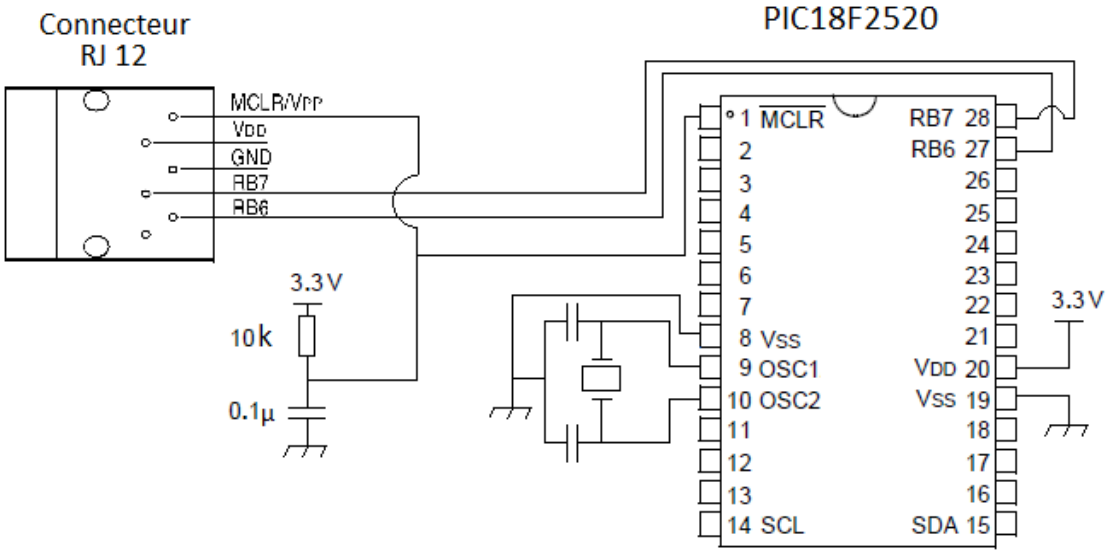


Pour avoir une régulation de 3.3V, nous avons mis une résistance R2=410 Ω.

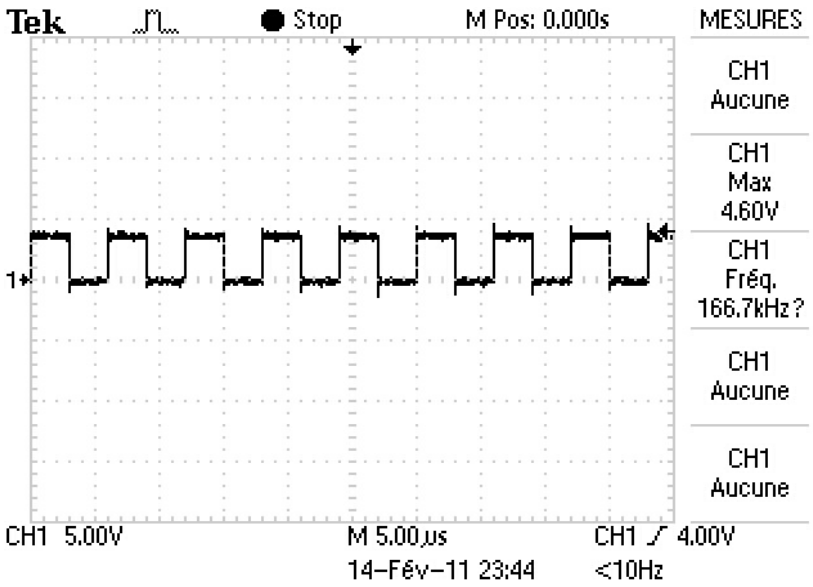
B/Le microcontroleur : PIC18F2520

Pour communiquer avec la caméra nous avons programmé un pic18F2520, qui possède un port série synchrone que nous utiliserons en bus I2C.

Schéma de câblage:



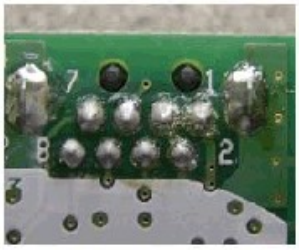
Pour vérifier la bonne réalisation du schéma et le bon fonctionnement du pic nous avons simulé sur une de ses sorties (ici RB2) le changement d'état d'une LED.



C/ Connexion de la camera

Tout d'abord le technicien a posé la camera sur un support, afin de ne pas l'endommager lors de la mise sous tension de la carte. Nous vérifions toujours au voltmètre les tensions d'alimentation à l'emplacement de la camera avant de la positionner.

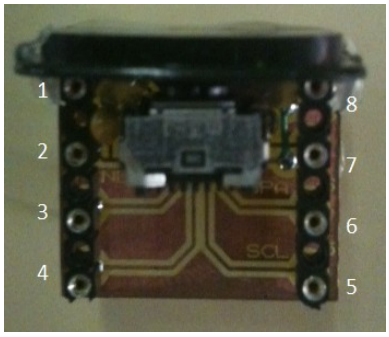
Nous avons communiqué au technicien le schéma d'implantation de la camera suivant (vu de dessous) :



1 – Vcc (+3.3V)
2 – GND
3 – GND
4 – no connection
5 – SCL (I2C BUS)
6 – SDA (I2C BUS)
7 – CLK (25MHz)
8 – #RESET

<u>Voici le support réalisé:</u>

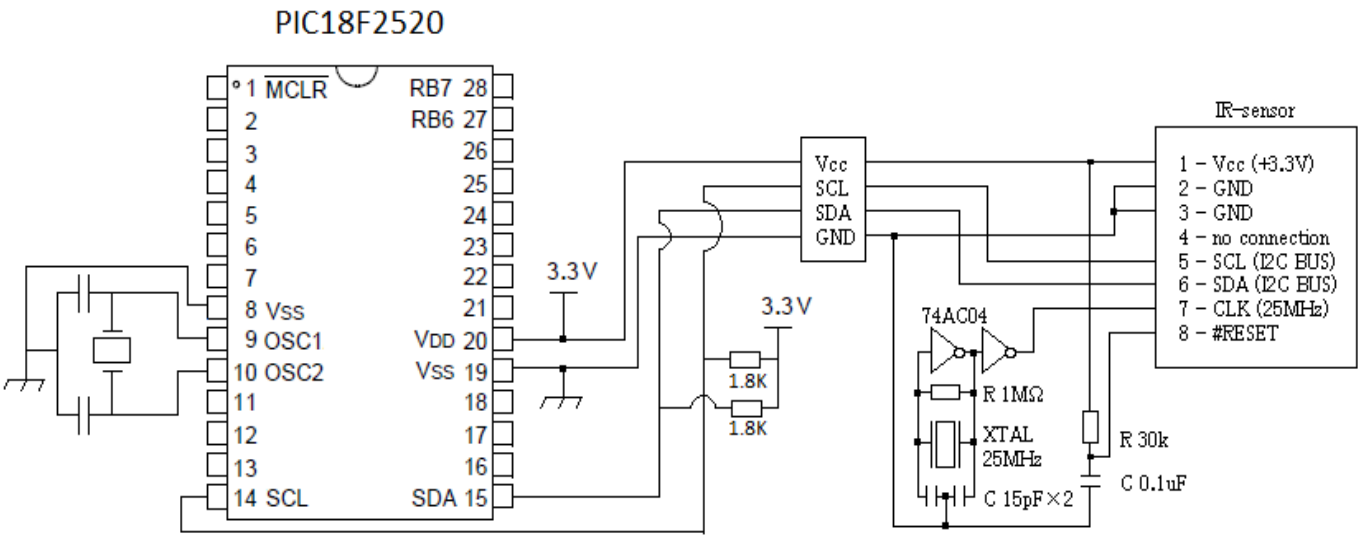
- Vu de dessus



- Vu de 3/4

Nous avons fixé le filtre infrarouge sur le support afin de ne relever que les points infrarouges et ne pas être perturbé par d'autres sources lumineuse.

Nous avons ensuite connecté la camera grâce au schéma de câblage suivant trouvé sur internet:



La communication entre la caméra et le PIC se fait par le Bus I2C, qui est constitué d'un fils d'horloge (SLC) et d'un fils de donnée (SDA). Nous utiliserons le PIC comme maitre et la camera comme esclave cette hiérarchie impose de respecter des codes de programmation spécifiques.
Comme source lumineuse nous avons utilisé des diodes infrarouges sous une tension de 11V avec des résistances de 220Ω pour avoir une tension de 50mA.

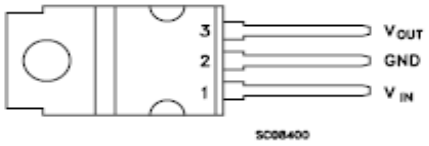
D/ Réception des données sur le PC

Afin de relever et de visualiser les données de la camera nous avons utilisé la liaison série puisque le PIC 18F2520 possède un port série asynchrone. Il nous a fallu tester la transmission car le PIC est alimenté en 3.3V et le Max232 en 5V. Nous avons validé les échanges d'octets en transmission et en réception sur cette liaison avec un programme.
Avec le logiciel Hyperterminal communiquant avec le port série nous affichions les données transmises du PIC, elles mêmes envoyées de la camera.

E/Les changements suite aux problèmes

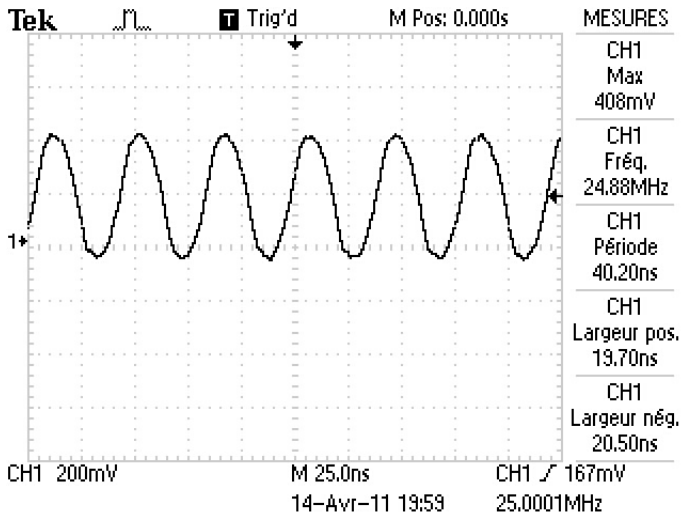
Nous nous sommes rendu compte que la régulation n'était pas satisfaisante, en effet la tension était légèrement supérieur à 3.3V, et surtout très bruitée. Pour y remédier nous avons remplacé le montage de régulation par un régulateur L4931C33 qui délivre directement 3.3V, et pour obtenir une alimentation plus lisse, nous avons mis un condensateur de découplage de 0.1µF tantal.

Schéma de connexion:



De plus nous avons remarqué que le signal de l'horloge pixel de la caméra était parasité et oscillé entre des niveaux trop élevés. Afin de l'améliorer nous avons modifié le montage en augmentant la valeur de la résistance mais sans résultat. Nous avons alors remplacé ce montage par un oscillateur ACHL 25 MHz fonctionnant avec une tension d'alimentation de 3.3V (datasheet Annexe1). Le résultat était bien meilleur.

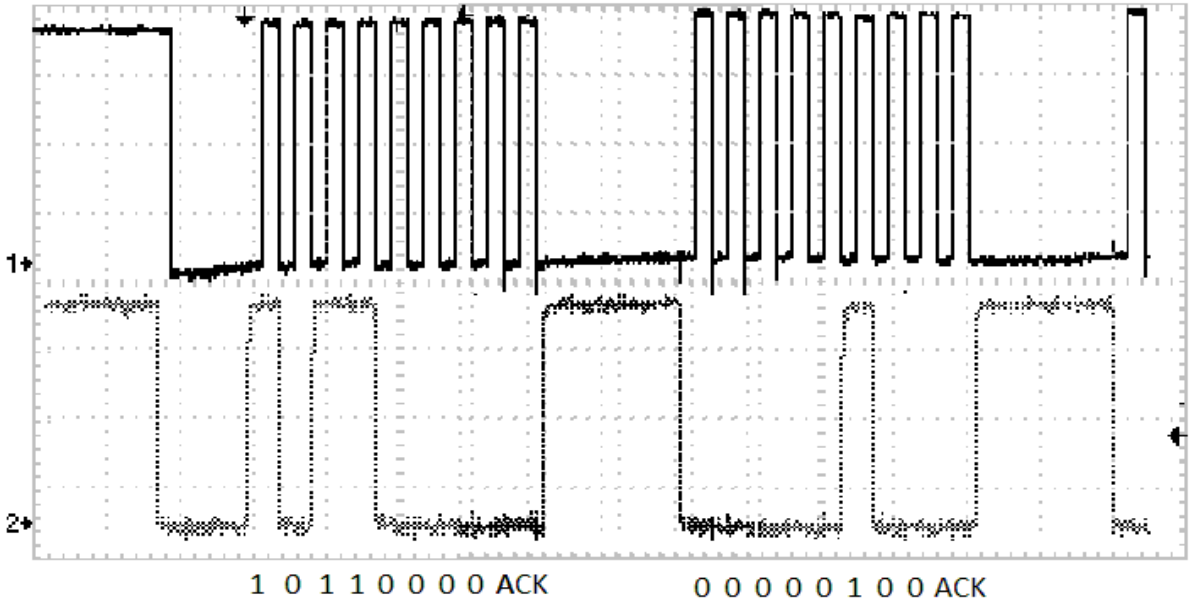
Signal d'horloge:



E/ Visualisation du Bus I2C

Les tensions d'alimentations et le signal d'horloge de la camera été corrigé cependant nous ne visualisions toujours pas les coordonnées des points. Nous avons alors "espionné" le bus I2C avec l'oscilloscope afin de savoir si la communication entre le maitre et l'esclave s'effectuait correctement.
La méthode consiste à l'aide de deux sondes, de visualiser sur la voie 1 le signal d'horloge (SLC) et sur la voie 2 le signal des données (SDA). A synchroniser l'oscilloscope sur la voie 1 et appuyer sur les touches RUN/STOP et Sync Unique afin de mémoriser les premiers échanges (il aurait fallu plus de mémoire à oscilloscope pour relever plus d'échanges).

Voici le relevé à l'oscilloscope:



Les données échangées sont 0xB0 et 0x04 qui correspondent bien à la partie d'initialisation de la camera le premier octet correspond à l'adresse d'écriture de la camera et le deuxième au premier registre d'activation.
Le neuvième bits (ACK) à la suite des données envoyées et émis par la camera qui atteste la bonne réception des données.

II-La partie software

A/Initialisation de la camera

Afin d'initialiser la camera, 7 commandes sont requises.
- Activation de l'horloge pixel (patte 7): on rentre les données 0x06 dans le registre 0x13.
- Activation de la camera: on rentre les données 0x06 dans le registre 0x1A.
- Début de configuration de la camera : on rentre les données 0x01 dans le registre 0x30.

On règle ensuite la sensibilité, cette étape ce déroule en deux parties.
- On configure le bloc 1 de sensibilité composé des registres 0xb00000-08 (9 octets)

Registre	Mémoire	
0xB00000	INCONNU	Valeur 2(standard) ou 7(sensibilité maximale).
0xB00001	0x00	
0xB00002	0x00	
0xB00003	0x71	
0xB00004	0x01	
0xB00005	0x00	
0xB00006	MAXSIZE	Taille maximale d'un point. Valeur comprise entre 0x62 et 0xc8.
0xB00007	0x00	
0xB00008	GAIN	Gain du capteur qui est grand pour des valeurs petites.

- On configure le bloc 2 de sensibilité composé des registres 0x0001A-1B (2 octets)

0xB0001A	GAINLIMIT	Doit être inférieur à la valeur du GAIN
0xB0001B	MINSIZE	Taille minimale d'un point. Valeurs allant de 3 à 5.

Nous avons utilisé le bloc de configuration suivant trouvé sur internet testé pour fonctionner.

Block 1									Block 2	
02	00	00	71	01	00	aa	00	64	63	03

- Ensuite on détermine le mode de déclaration des données nous avons choisi le mode étendu: on rentre la valeur 3 dans le registre 0x33.

0x33: Core Buttons and Accelerometer with 12 IR bytes

This mode returns data from the buttons, accelerometer, and IR Camera in the Wii Remote:

```
(a1) 33 BB BB AA AA AA II II II II II II II II II II II II
```

BBBB is the core Buttons data. AA AA AA is the Accelerometer data. The 12 II bytes are from the built-in IR Camera.

On remarque que ce registre est composé d'autres octets que ceux pour la camera (accéléromètre, boutons). Ce sont les 12 octets « II » qui nous intéresse et que l'on traitera plus bas.

- On clos la configuration de la camera : pour cela on rentre les données 0x08 dans le registre 0x30.

B/Récupération des données

Nous nous somme inspiré de programmes existant, utilisant les fonctions de l'Arduino, que nous avons retranscrit pour une réception en I2C.

```
StartI2C(); IdleI2C(); // Le maitre (PIC) génère un bit de START
WriteI2C(ADR_CAM_WRITE); IdleI2C(); // Connexion à l'adresse d'écriture 0xB0
WriteI2C(0x36); IdleI2C(); // Adresse de stockage des données
StopI2C();IdleI2C();// Le maitre génère un bit de STOP

StartI2C(); IdleI2C();// Le maitre (PIC) génère un bit de START
WriteI2C(ADR_CAM_READ); IdleI2C();// Connexion à l'adresse de lecture 0xB1
for(i=0;i<16;i++)
{
       data_buf[i]=ReadI2C(); IdleI2C(); // Reception de 16 octets
       if(i<15)
       {AckI2C(); IdleI2C();}  // Le maitre accuse la bonne reception des octets
       else
       {NotAckI2C();IdleI2C();} // Le maitre émet un NotAck pour le dernier octet reçu
}
StopI2C();IdleI2C(); // Le maitre génère un bit de STOP
```

On notera qu' après chaque action (StartI2C,WriteI2C,StopI2C) on attend que celle ci soit fini avant de passer à la suivante, en écrivant IdleI2C().
Dans le mode étendu de réception des données, la camera retourne 3 octets par point, soit au total 12 octets parmi les 16 réceptionnés. Ils correspondent à l'emplacement X et Y d'un point et une valeur approximative de ca taille. Le format des données reçu est le suivant, cette structure est répétée 4 fois.

BYTE	BIT 7	6	5	4	3	2	1	BIT 0
0	X[7:0]							
1	Y[7:0]							
2	Y[9:8]		X[9:8]		S[3:0]			

Une fois ces données reçu nous les avons traitées en faisant des masques/décalages afin de récupérer les coordonnées X et Y des points sur 10bits. Le premier octet data_buff[0] est vide, les données sont contenues à partir de data_buff[1] jusqu'à data_buff[12] .

```
//coordonnées du blob 1
       X1 = data_buf[1];
       Y1 = data_buf[2];
       s = data_buf[3];
       X1 += (s & 0x30) <<4;
       Y1 += (s & 0xC0) <<2;
//coordonnées du blob 2
       X2 = data_buf[4];
       Y2 = data_buf[5];
       s = data_buf[6];
       X2 += (s & 0x30) <<4;
       Y2 += (s & 0xC0) <<2;
```

```
//coordonnées du blob 3
       X3 = data_buf[7];
       Y3 = data_buf[8];
       s = data_buf[9];
       X3 += (s & 0x30) <<4;
       Y3 += (s & 0xC0) <<2;
//coordonnées du blob 4
       X4 = data_buf[10];
       Y4 = data_buf[11];
       s = data_buf[12];
       X4 += (s & 0x30) <<4;
       Y4 += (s & 0xC0) <<2;
```

Voici l'affichage sur le PC :

```
blob1(X;Y) 1023 1023  blob2(X;Y) 1023 1023  blob3(X;Y) 1023 1023  blob4(X;Y) 1023 1023
blob1(X;Y) 857 253  blob2(X;Y) 258 260  blob3(X;Y) 1023 1023  blob4(X;Y) 1023 1023
blob1(X;Y) 861 354  blob2(X;Y) 263 372  blob3(X;Y) 1023 1023  blob4(X;Y) 1023 1023
```

La camera envoie la valeur 0xFF lorsque que le blob n'est pas détecté.

C/Tracking des points

Le tracking des points consiste à associer à chaque point détecté un numéro permanent afin de suivre son évolution. La réalisation ce cette algorithme est basée sur une recherche des distances minimales entre les coordonnées enregistrées et celle venant d'être relevées. Pour cela nous avons définie une structure de blob.

```
typedef struct blob blob;
struct blob{
            int X; // Coordonnées en X
            int Y; // Coordonnées en Y
            };
blob tab[4],tab_mes[4]; // Données enregistrées , Données mesurées
```

Les premières coordonnées reçues sont stockées dans le tableau tab[],les données suivantes sont stockées dans le tableau tab_mes[]. Ainsi on compare les distances entre les points afin de déterminer les nouvelles coordonnées des blob enregistrés.
Nous avons discerné deux types de traitement:

- Dans le cas d'une disparition de point : on compare les données enregistrées dans tab_mes[] à celle dans tab[].

Par exemple on calcul la distance du premier point dans tab_mes[0] à chacun des 4 points enregistrés dans tab[0..3]. On l'assigne alors au blob le plus prêt (distance la plus faible), puis on mais un flag sur ce blob pour que l'on ne le compare plus. Et on procède de même pour les points à traiter restant.

- Dans le cas d'une apparition de point (ou nombre inchangé) : On procède comme précédemment sauf que l'on compare les données enregistrées dans tab[] aux nouvelles enregistrées dans tab_mes[].

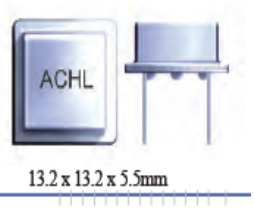
Cependant le programme présenté quelques problèmes pour le traitement des 4 points.

III/ Perspective d'utilisation sur robot

Tout nos essais précédent on étaient réalisés à une distance proche de la camera (environ 1mètre). L'implantation de la camera sur un robot nécessite quelle puisse détecter les points à une grande distance (11mètres). Nous avons alors commencé par déterminer le nombre de diodes infrarouges et l'intensité à fournir pour que la camera détecte 1 point à cette distance. Il a fallu 16 diodes avec une intensité de 70mA pour avoir un résultat satisfaisant.
Le robot sera en mouvement il devra détecter les points de la balise pour se déplacer cependant les diodes étant positionnées sur un plan rectiligne il y a de forte chance pour qu'il sorte du faisceau et perde ainsi de vu la balise.
Nous avons donc évalué la marge de manœuvre possible dans laquelle le robot peut détecter la balise (composé d'un seul point) toujours pour une distance de 11 mètres. En considérant que la camera soit en face de la balise elle peut s'écarter d'approximativement de 25° sur la gauche et la droite, passé cette intervalle elle ne détecte plus la balise.
L'idée est par la suite de constituer une balise de 3 points alignés horizontalement afin que le robot se dirige en fonction du point du milieu qu'il devra positionner le plus au centre de l'écran, et en fonction de la distance pixel entre les deux autres points des extrémités, d'évaluer la distance jusqu'à la balise et ainsi d'adapter ca vitesse.

Annexe1

HALF SIZE DIP LOW VOLTAGE 3.3VHCMOS/TTL COMPATIBLE CRYSTAL CLOCK OSCILLATOR



ACHL Series

RoHS Compliant

FEATURES:

- Tri-state Enable/Disable option
- Low supply voltage 3.3V
- HCMOS and TTL compatible
- Tight symmetry option 50% +/-5%

APPLICATIONS:

- Clock signal sources for digital chips and microprocessors
- Low power applications

STANDARD SPECIFICATIONS:

Parameters		Minimum	Typical	Maximum	Units	Notes
Frequency Range		0.400	-----	160.000	MHz	
Operating Temperature		0	-----	+70	°C	See options
Storage Temperature		-55	-----	+125	°C	
Overall Frequency Stability		-100	-----	+100	ppm	See options
Supply Voltage (Vdd)		2.97	3.3	3.63	V	
Input Current	0.400MHz ~ 24.999MHz	-----	-----	20	mA	
	25.000MHz ~ 99.999MHz	-----	-----	40		
	100.000MHz ~ 16000MHz	-----	-----	80		
Symmetry (@ 1/2Vdd)		40	50	60	%	See options
Rise and Fall Time (Tr/Tf)	0.400MHz ~ 24.999MHz	-----	-----	10	ns	
	25.000MHz ~ 99.999MHz	-----	-----	5		
	100.000MHz ~ 16000MHz	-----	-----	4		
Output Load		-----	-----	15	pF	See options
		-----	-----	5	TTL	
Output Voltage	VOH	0.9*Vdd	-----	-----	V	
	VOL	-----	-----	0.1*Vdd	V	
Tri-state Function	VIH	2.2	-----	-----	V	"1" or Open: Oscillation
	VIL	-----	-----	0.8	V	"0": Output disable (Hi Z)
Disable/Enable time				100	ns	For option A only
Start-up Time		-----	1	10	ms	
Phase Jitter RMS (12~20MHz)		-----	-----	1	ps	Reference only. Please contact Abracon for specific frequencies
Aging		-5.0	-----	+5.0	ppm	@+25°C First year

*** Overall frequency stability includes initial tolerance, temperature stability, and aging.**

Annexe1 (suite)

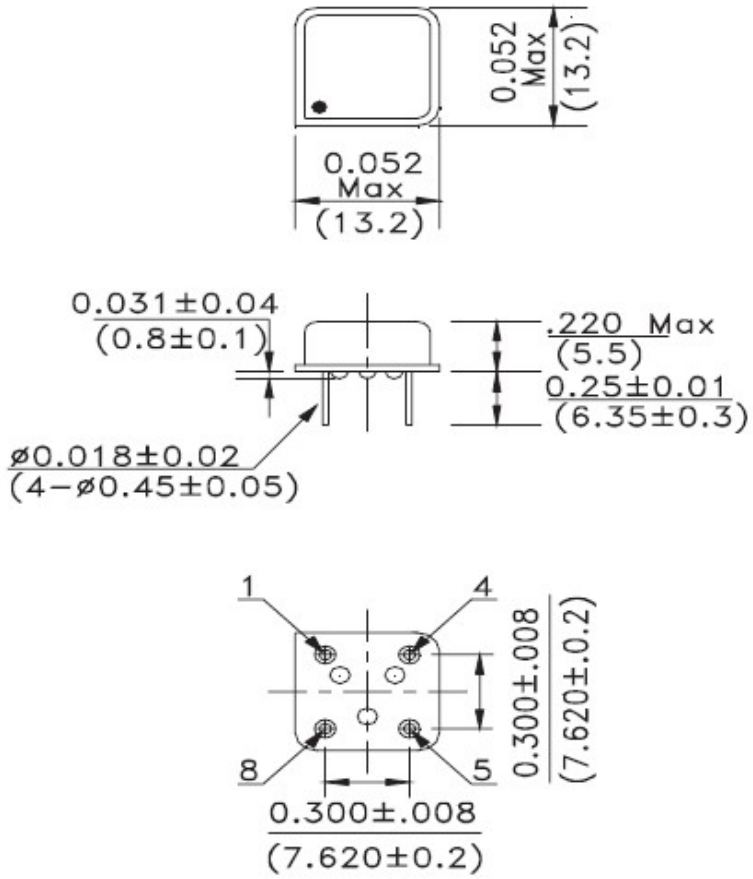
HALF SIZE DIP LOW VOLTAGE 3.3VHCMOS/TTL COMPATIBLE CRYSTAL CLOCK OSCILLATOR

ACHL Series

Pb | RoHS Compliant

13.2 x 13.2 x 5.5mm

OUTLINE DIMENSIONS:



PIN	FUNCTION
1	NC or Tristate Enable/ Disable
4	GND
5	Output
8	VDD

Note: Recommended using an approximately 0.01uF bypass capacitor between PIN 4 and 8.

Dimensions: inches (mm)

Annexe 2 (Source)

Wiimote Description

http://docs.google.com/viewer?a=v&q=cache:vH3xcAJg8eMJ:wiimoteproject.googlecode.com/files/2%2520Background.doc+camera+ir+pixart&hl=fr&gl=fr&pid=bl&srcid=ADGEESikSX21LgA2suZkVluVDxW9WdMA0IjTu8dDlVP917_PkOLikCE2gNB3szuQNvinoyh12ZCjqqmfgl7yyZfjhoq-kOjwpKDWxSR0fJo5yjwZDRq5_-eh0cfkkV39GEp1vfMrci9R&sig=AHIEtbQHGeBMp0pTrdNfaltme55TJoaOMQ

Capteur IR

-Registre

http://wiki.wiimoteproject.com/IR_Sensor

http://wiibrew.org/wiki/Wiimote

-Schéma câblage

http://stephenhobley.com/blog/wp-content/uploads/2009/02/wiicam_schem.gif

-Programmation I2C

http://procrastineering.blogspot.com/2008/09/working-with-pixart-camera-directly.html

-Programmation Arduino

http://www.stephenhobley.com/blog/2009/03/01/pixartwiimote-sensor-library-for-arduino/

http://www.stephenhobley.com/blog/2009/02/22/pixart-sensor-and-arduino/

-KAKO

http://translate.google.fr/translate?u=http%3A%2F%2F66.196.80.202%2Fbabelfish%2Ftranslate_url_content%3F.intl%3Dus%26lp%3Dja_en%26trurl%3Dhttp%253a%252f%252fwww.kako.com%252fneta%252f2008-009%252f2008-009.html&sl=en&tl=fr&hl=&ie=UTF-8