Autonomous robots project course FAQ

TS and WS autumn 2005

1 Introduction

This document briefly describes the computer and electronics used in the course "Robot Design Course" at the IDE master program. Especially the *DSP 6211* together with the *robotnic*[1]interface card. The document is not a tutorial, but can be regarded as a FAQ of how to work with the hardware.

2 Electronics

2.1 The digital and the analog input

The digital inputs include a pull up resistor. Thereby can a switch directly be inserted between the digital inputs and ground just like the photo transistor in the figure below.

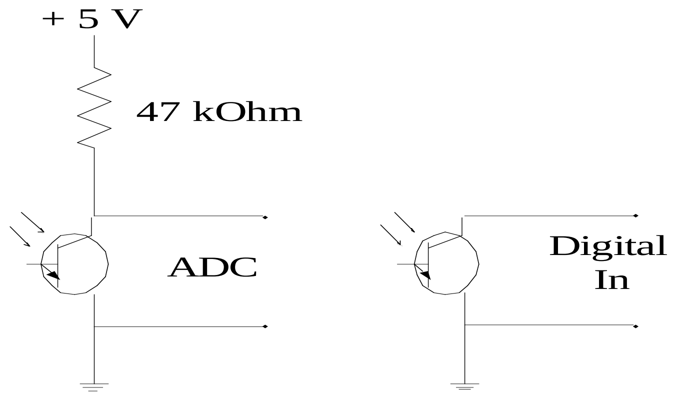


Figure 1. Photo transistor

The analogue input (ADC) has an impedance of 20 kOhm which influences the use of photo transistor. These transistors are feed with 5 Volt via a 47 kOhm resistor. But this is in the same size as the input impedance giving a non direct relation light to input voltage. So, if the transistors are used as an optic-switch then by placing the diodes close to each other, it is possible to use the digital input instead.

2.2 Sensors

The sensors are used as following

- switches: between digital in and ground
- phototransistor: see Figure 1
- photo interrupter: see Figure 2.

[1]*This is an interface card developed at Halmstad University by Björn Åstrand, Tommy Salomonsson and Ruben Rydberg 2001.*

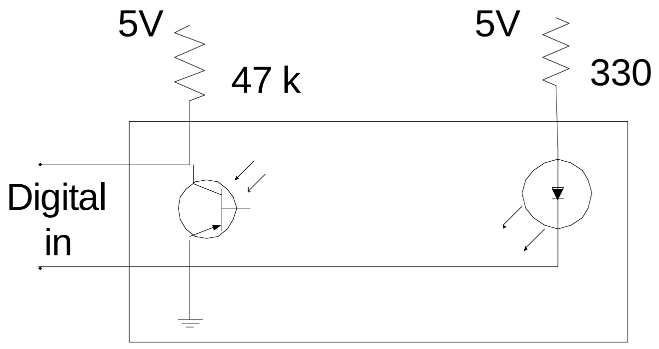


Figure 2: Photo interrupter

2.3 Actuators

To run the DC-motor the power (7.2V) must be connected, if not uncertain behaviour will occur.

- The DC-motors are controlled via an onboard H-bridge. The maximum output current is thereby from each bridge 2Ampere. This gives a high speed and care must be taken so that the *LEGO* parts not are damaged.
- The servo motors are dc-motors with an inherent position controller. The reference input is an angle. A value between 0-255 written to the output address gives an angle of approximately 0-90 degrees. This signal is transformed to pulse in a PLD circuit where the pulse width determines the rotation angle.

2.4 Digital output

The digital output gives 3.3 volt and 10 mA and the LED should be feed via a 330 Ohm resistor. The same yields the photo interrupter. A remainder: the longer leg of the diode is the anode (plus).
Remark: If the output by accident is short cut, the DSP will be destroyed! Please be careful.

3 Camera and taking a picture

Camera constrains

- Timing
- Synchronising

A photo shot is accomplished by using the function `GrabbImageRGB(&Image,sync)` which returns and copies the image to the *Image* address. There are also several existing commands (functions) to control the performance of the camera.

3.1 Initializing

- `ResetCamera()` A electrical signal goes low and the camera is hardware reset and `ResetChip()` is a software reset.
- `ConfigImageType` Defines among other where to save the image, and what size and type. The general type to use is RGB. The size is defined with height, width and decimation which can be used when resizing or cutting in the picture. But doing this chancing the view6000.cfg file must also be changed for correct PC-loading of the image.
- `InitCamera` Configures the camera registers.

3.2 Adjusting the cameras colour settings

To adjust the colour settings in the camera it is most practical to have a white paper response of R=G=B= 0.9

(90 %). But in some cameras (as the one we use here) the colour values saturates due to an internal nonlinearity. So instead, a grey printed paper with the grey level values R=G=B=0.5 (50 %) can be used. By using some functions the internal camera colour settings can be adjusted.

- `AGCGainControl` amplifies all R,G and B values simultaneously and
- `BlueGainControl, RedGainControl` adjusts Blue and Red one at the time.

To control the colour settings the command `Autoadjust` must be set to disabled. This command also disables the brightness control. Don't use this function more then necessarily those function take “long” time to execute.

Why bother? Using autoadjust the camera adjusts to the light each time a picture is taken. If you are only interested in relative colour value it is easier to lock the amplification a specific value. It is far more complicated to use the autoadjustemet function. The specific value is chosen using the Read functions. Then the most critical part is to choose value so the colour values don't saturate.

3.3 Indoor filter

This is set using `BandFilterEnable(x)`. This manages the sensitivity for different wavelengths. This is because of the different light in (x=1) and outdoors (x=0).

3.4 How to take a picture with the camera

A picture is taken using `GrabbImageRGB(&Image,sync)` and is saved on the address defined in `Image` (usually on the SDRAM). This address is set `ConfigImage`. The argument *sync* controls if the picture is read by the DSP in a synchronized mode or not. Usually must no care be taken if the whole read sequence is finished or not since the difference between two pictures is small normally.

The image is saved as a matrix of *unsigned int*’s (32bit). Each pixel is stored in four bytes: { non-info, blue, green, red }, witch makes the sampling sequence of one colour simple. As for example below getting the blue, green and red values from a pixel in the image:

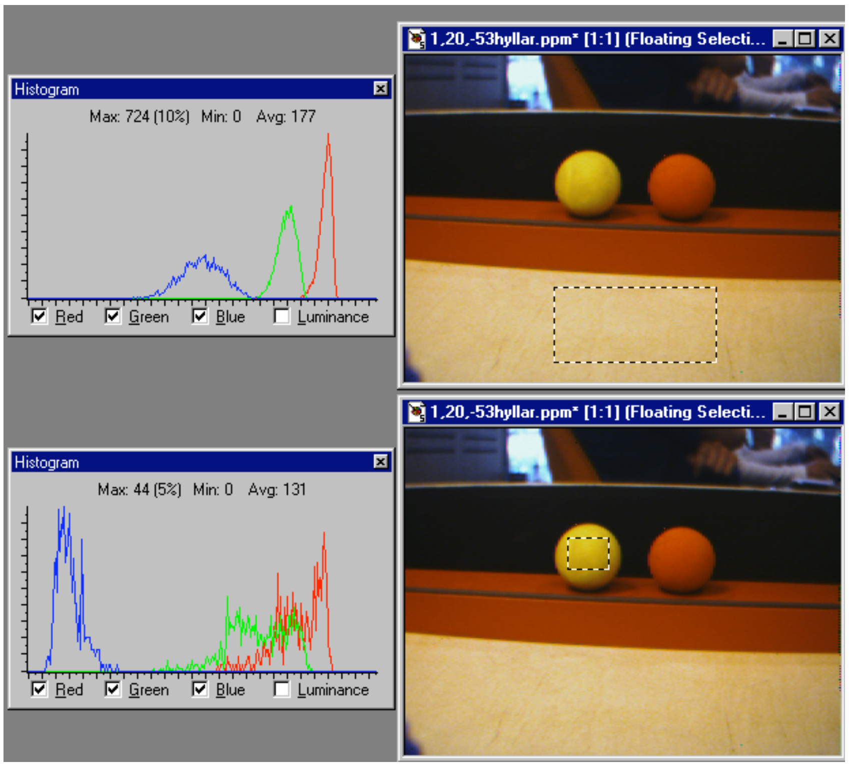
```
for (row=0;row< IMAGE.HEIGHT ; row++){
  for (column=0; column < IMAGE.WIDHT; column ++){
          pixel = Image[row* IMAGE.WIDHT+column];
          blue = (pixel&0xFF0000)>>16;
          green = (pixel&0xFF00)>>8;
          red = (pixel&0xFF);
   }
 }
```

3.5 Find a colour

Using Paint Shop Pro

The most straight forward approach to get a pixel RGB value is to use the Dropper function from the tool palette. But here are some tips of how to use Paint shop pro to find a “good” colour to colour segment.

- The *Histogram* function is found as a button in the standard toolbar menu at the top. With this function a histogram for each colour in the picture can be plotted. Se figure 3.
- If you are interested in a specific area you should use *Selection* from the tool palette. But, you must click twice on the selected area to tell the program which area is interesting. This function is useful to see the variations of a specific colour. Se figure 3.
- To see the different contents of the three colours it is possible to view them separately with *Colours->SplitChannel->Split to RGB*.



Histogram
Max: 724 (10%) Min: 0 Avg: 177
Red
Green
Blue
Luminance
1,20,-53hyllar.ppm* [1:1] (Floating Selecti...
Histogram
Max: 44 (5%) Min: 0 Avg: 131
Red
Green
Blue
Luminance
1,20,-53hyllar.ppm* [1:1] (Floating Selecti...

Figure 3: Snapshot from Paint Shop Pro

The histogram to the left is corresponding to the selected area in the image to the right. To classify colour from the yellow ball and the floor the blue colour space can be used. The floor has higher intensity in blue then the yellow ball.

3.6 Numerical representation

During the development of algorithms the programming is done on different computer platforms, both PC and DSP which can influence the numerical representation. The language such as MATLAB or C is also affecting this.

- Floats and integers
- Normalisation 0-1 or 0-255

4 Dead reckoning

The most usual and simple a mobile robot is navigating is by using *dead reckoning*. This is when the robot counts the number of pulses generated of the wheels in order to keep track of its position. There exits several ways to use coordinate systems in dead reckoning. But, we will only mention the technique where the robot has one coordinate system and the room where it is moving has another.

The principle of dead reckoning is to sample pulses from the wheels. From these the distance is, which the robot has moved, calculated. This includes both rotational and translatational movement since the last sample. Together these samples form a path and the position of the robot can be predicted. An example of a robot path is shown in Figure1.

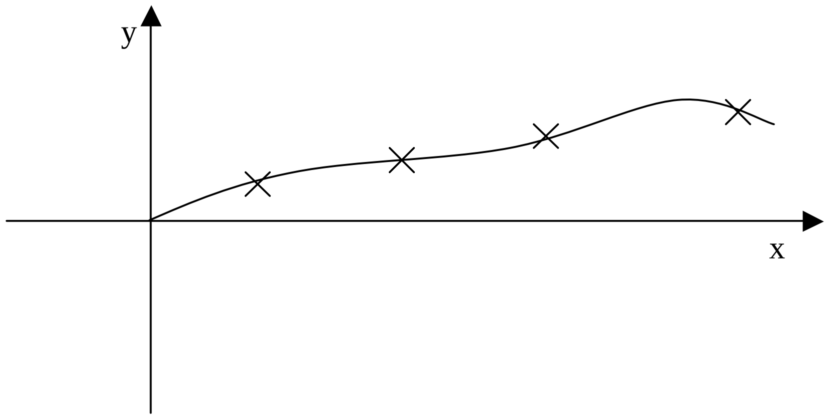


Figure: 3 Samples of the movement in room coordinates.

The drawback of dead reckoning is the integration of small errors a large error after several samples.

4.1 Encoders

Encoders are not only used for pulse counting, but also for the detection of the wheels rotation direction. A solution of this is using relative encoders which give two pulse trains.

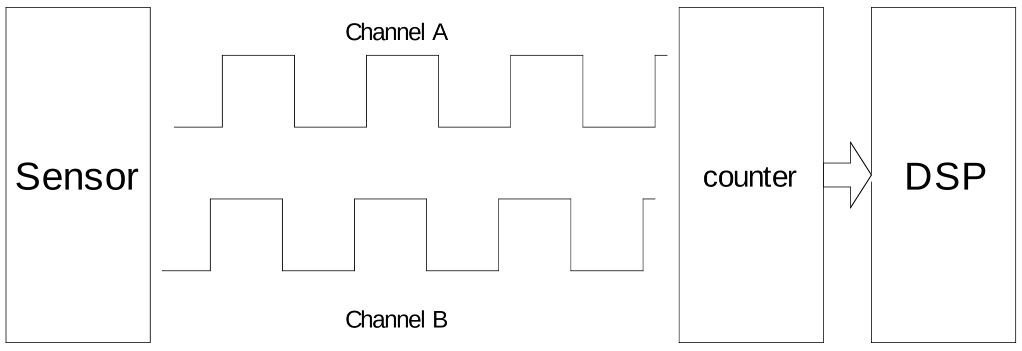


Figure4: Pulses from an encoder.

The pulses from *channel A* are delayed for one direction and in the other direction is *channel B* delayed. A counter then increases and decreases it.

Value ending of the phase information

4.2 Derive dead reckoning from rotation and translation

As model at the prediction of the current position using dead reckoning we use a robot with one axis.
Define the new position at instant *k* in x and y-directions as

$$y(k) = y(k-1) + \Delta y(k)$$
$$x(k) = x(k-1) + \Delta x(k)$$

where Δy and Δx are small moves in the y and x-direction and are found from the encoder values. If Δz is the distance between two sampled values at *k* and *k-1*. Δz is actually the approximated straight line of the real robot path and is

$$\Delta z = \frac{\Delta L + \Delta R}{2}$$

where ΔL and ΔR are derived from the encoder values as

$$\Delta L = \frac{Left\_encoder(k) - Left\_encoder(k-1)}{Const\_[pulses/mm]}$$
$$\Delta R = \frac{Right\_encoder(k) - Right\_encoder(k-1)}{Const\_[pulses/mm]}$$

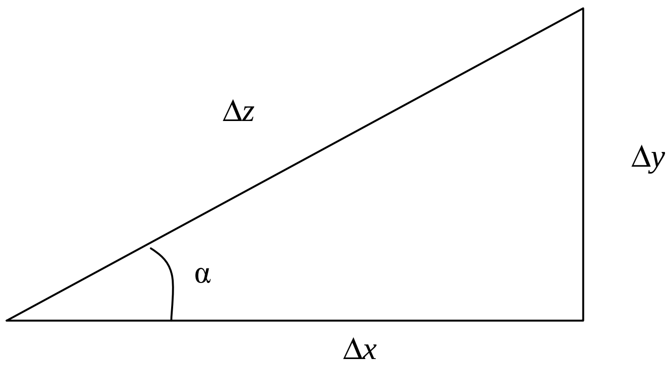


Figure 5: Robot translation

So this gives Δx and Δy :

$$Sin(\alpha) = \frac{\Delta y}{\Delta z}$$
$$Cos(\alpha) = \frac{\Delta x}{\Delta z}$$

where α is the angle robot to global coordinate system and is found with

$$\alpha(k) = \alpha(k-1) + \beta(k)$$

Here is β the change of angle since last sample. Figure6 describes the rotation of a robot, seen from above with one axel and wheels. The dotted lines correspond to the new position due to the change of angle. β derived from

$$Sin(\beta) = \frac{b}{r}$$

where *b* is the circular path of the wheels and *r* is half of the distance between the wheels. For simplicity, since β is small, is b approximated with a straight line. This is valid for high sample frequencies and then *b* is derived from $b = \Delta R - \Delta L$.

Since ΔL and ΔR known from above are all variables found.

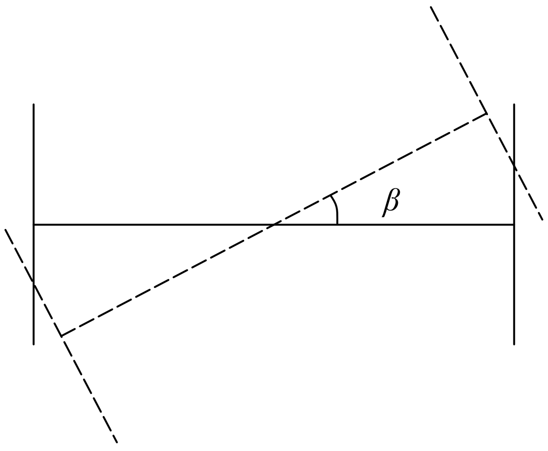


Figure 6: Robot rotation (Altering angle β)

5 Find an object in a binary image

To find an object in a binary image can be done in several ways. One way is by histogram another way is by the formula “centre of gravity”.
The centre of gravity of an object in a binary image can be found by the formula below.

$$C_x = \frac{1}{n} \sum n_x x$$

$$C_y = \frac{1}{n} \sum n_y y$$

Where n is the numbers of pixel that belongs to the object.
n_x are the numbers of pixel in position x
n_y are the numbers of pixel in position y

6 The program in pseudo code

```
Init_DSP_and_memory
Init_Camera
Autoadjust_Off_
Allocate_Memory_Image
While(1){
   Wait_for_Tick (optional)
   Take_Picture
   Colour_Segment
   Filter_Graylevel (optional)
   ThresHold
   Filter_Binary (optional)
   Classify_Object
   Find_Object
   Take_Some_Action
}
```

A nice and simple approach is to use a variable that describes in what state the robot is in. For an example the robot can be in a state like find object, bring object, find home, go home and so on. Depending in what state the robot is in the robot has different colour to segment and different object to classify.