DD2426 Robotics and Autonomous Systems

Project notes B
April 10 2008

- Outline
  - Robot soccer rules
  - Hardware documentation
  - Programming tips
  - RoBIOS library calls
  - Image processing
  - Construction tips

Robot soccer rules

The rules are on the course webpage and will not be printed and handed out, since....

The rules document might change slightly as more clarifications are added.

The rules is therefore a “living” document and is best viewed directly from the webpage where the latest version will be available.

Please let me know if you think something is missing in the document.

Hardware documentation

A link to datasheets for many of the components used in the course is on the course web page. Look under the project submenu

It can be useful to know that we have

- M4 Eyebot controller boards
- 4 mbit FLASH memory
- Cameras with the Omnivision 6620 chip
- A FIFO camera buffer

CPU properties

The 68332 has no floating point instructions. All floating point math is done in software.

It has 32 bit registers and instructions but only a 16 bit data bus.

→ *Use 16 bit fixed point math whenever possible*

→*avoid trigonometric functions (use lookup tables)!*

Integer data types include:

`int` 32 bits

`short` 16 bits

`char` 8 bits

All are available as signed and unsigned.

For example to calculate *a*/*b* (`a` and `b` 8 bit) use:

```
unsigned char a, b;
unsigned short c;
…
c=256*a/b;
```

This calculates the result using fixed point math and puts it in `c` with 8 bits of ‘decimals’.

CPU properties (cont'd)

Instruction		Head	Tail	Cycles
ADD(A)	Rn, Rm	0	0	2(0/1/0)
ADD(A)	⟨FEA⟩, Rn	0	0	2(0/1/0)
ADD	Dn, ⟨FEA⟩	0	3	5(0/1/x)
AND	Dn, Dm	0	0	2(0/1/0)
AND	⟨FEA⟩, Dn	0	0	2(0/1/0)
AND	Dn, ⟨FEA⟩	0	3	5(0/1/x)
EOR	Dn, Dm	0	0	2(0/1/0)
EOR	Dn, ⟨FEA⟩	0	3	5(0/1/x)
OR	Dn, Dm	0	0	2(0/1/0)
OR	⟨FEA⟩, Dn	0	0	2(0/1/0)
OR	Dn, ⟨FEA⟩	0	3	5(0/1/x)
SUB(A)	Rn, Rm	0	0	2(0/1/0)
SUB(A)	⟨FEA⟩, Rn	0	0	2(0/1/0)
SUB	Dn, ⟨FEA⟩	0	3	5(0/1/x)
CMP(A)	Rn, Rm	0	0	2(0/1/0)
CMP(A)	⟨FEA⟩, Rn	0	0	2(0/1/0)
CMP2 (Save)*	⟨FEA⟩, Rn	1	1	3(0/1/0)
CMP2 (Op)	⟨FEA⟩, Rn	2	0	16 - 18(X/1/0)
MUL(S/U).W	⟨FEA⟩, Dn	0	0	26(0/1/0)
MUL(S/U).L (Save)*	⟨FEA⟩, Dn	1	1	3(0/1/0)
MUL(S/U).L (Op)	⟨FEA⟩, Dl	2	0	46 - 52(0/1/0)
MUL(S/U).L (Op)	⟨FEA⟩, Dn:Dl	2	0	46(0/1/0)
DIVU.W	⟨FEA⟩, Dn	0	0	32(0/1/0)
DIVS.W	⟨FEA⟩, Dn	0	0	42(0/1/0)
DIVU.L (Save)*	⟨FEA⟩, Dn	1	1	3(0/1/0)
DIVU.L (Op)	⟨FEA⟩, Dn	2	0	<46(0/1/0)
DIVS.L (Save)*	⟨FEA⟩, Dn	1	1	3(0/1/0)
DIVS.L (Op)	⟨FEA⟩, Dn	2	0	<62(0/1/0)
TBL(S/U)	Dn:Dm, Dp	26	0	28-30(0/2/0)
TBL(S/U) (Save)*	⟨CEA⟩, Dn	1	1	3(0/1/0)
TBL(S/U) (Op)	⟨CEA⟩, Dn	6	0	33-35(2X/1/0)
TBLSN	Dn:Dm, Dp	30	0	30-34(0/2/0)
TBLSN (Save)*	⟨CEA⟩, Dn	1	1	3(0/1/0)
TBLSN (Op)	⟨CEA⟩, Dn	6	0	35-39(2X/1/0)
TBLUN	Dn:Dm, Dp	30	0	34-40(0/2/0)
TBLUN (Save)*	⟨CEA⟩, Dn	1	1	3(0/1/0)
TBLUN (Op)	⟨CEA⟩, Dn	6	0	39-45(2X/1/0)

From http://robotics.ee.uwa.edu.au/eyebot/ → Controller → Hardware → Data Sheets → MC68332 CPU32 Manual p. 8-16

CPU properties (continued)

Execution speed is of course most important for code that runs often, such as calculations on every pixel in an image.

This is where you should put most effort into optimizing your code.

Lookup tables can be very useful in this context. One example is finding the colour class of a pixel from RGB values (a three-dimensional table). Another is calculating a sine with a table and linear interpolation.

Compiling and linking

…can be done in one step with `gcc68`. Each `.c` file gives an object (`.o`) file, and all of them are then linked together with any libraries used to a `.hex` file.

Linking against an external library:

Specify library name with `-l` option last on line as in

```
gcc68 -o rob.hex rob.cc img.cc –lm
```

for the math library (which is not recommended for memory and speed reasons). This is not necessary for Robios library calls.

The Robios libraries and headers are under the `/usr/local/eyebot/RoBIOS` directory.

Or use `make` after copying `Makefile` and `Makeinkl` from the `~/eyebot/eyedemo` directory. Modify the names of all `.o` files in the `Makefile` to reflect the `.c` files you use.

V-ω interface

Allows you to set the speed (v) and rotational velocity (ω) of the robot, as well as make it go to a specified point, along a line or arc, etc.

It also keeps track of x,y position and rotation.

Initialize with

```
VWInit(VW_DRIVE, timescale);
```

where timescale determines the frequency of the controller, 100/`timescale` Hz.

Start the PI controller with

```
VWStartControl(handle,Vv,Tv,Vw,Tw);
```

The 2005 winner robot Robbe used 2, 0.5, 7, 0.007 for the control parameters.

The HDT must be adapted for you robot. See the *Quadrature encoder* and *WV Drive* sections of `http://robotics.ee.uwa.edu.au/eyebot/` → HDT.

Camera and image processing

Two ways to get an image from the camera

- Standard Robios call (82x62 **cutout**)

```
CAMGetColFrame()
```

- Full frame (~176x144)

```
CAMGetFrameRGB()
```

Use **subsampling** (using every other pixel in every other row; see example code) of the full frame image to get a wider field of view.

If you exclude areas of the image where you know you will not find your target, it could be possible to use all pixels without subsampling in the remaining part.

You can also use subsampling at a first stage, and then use all pixels in a few target candidate regions.

Analogue video output

After reset and before any RoBIOS call concerning the camera, an analogue video signal is present on the two unconnected pins on the camera board.

Use the video monitor an coax cable in the lab to view the B/W output. Only one of the two possible ways to insert the connector gives a good image on the monitor.

Note however:

- The framerate and resolution is much higher than what you will get digitally.
- Autobrigthness works much faster due to the higher framerate.
- Mostly usefull for focusing the lens and aiming the camera.

Camera parameters

You can set the frame rate of the camera with the `CAMSet()` call. Usable rates are:
`FPS7_5 FPS3_75 FPS1_875`

Note that the usable frame rate depends on running background tasks. E.g. when using v-ω motor control, set the camera speed to: `CAMSet (FPS1_875, 0, 0);`

There is also a `CAMMode()` call to turn auto-brightness on and off. **This does not work on our hardware.**
This means auto-brightness is always on.

This is a problem as the camera setting changes while playing.

Let the auto-brightness setting stabilize (faster at a higher frame rate) before starting a game or taking test images.

Colour classification

To find the different targets (ball, goals, and opponent), pixels in the image must be checked to see if they have the colour of a target.

RGB-values change a lot depending on the lighting, shadows, etc. They can instead be mapped to a HSV (Hue-Saturation-Value) colour representation, where the hue component is more stable for an object, such as the ball.

Hue:

blue – green – yellow – orange – red – purple

Saturation:

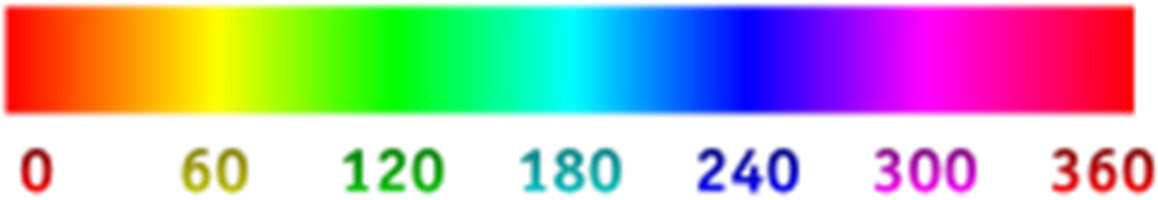
white – pink – red

Value (intensity):
Changes proportional to power of lights

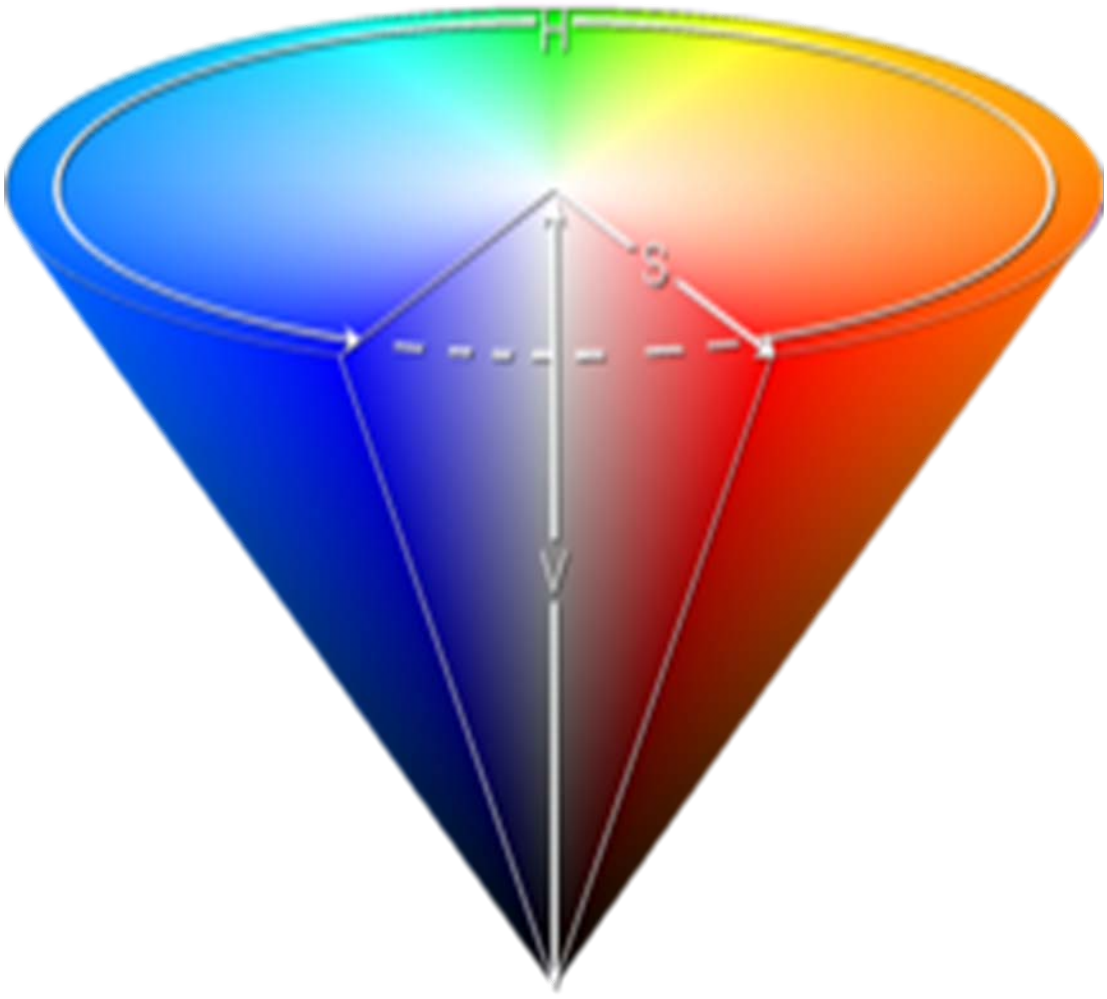
black – grey – white

The HSV colour space

Hue scale



HSV “cone”



Colour classification (cont'd)

The camera pictures are tinted yellow because of low blue sensitivity.

Before the hue is calculated this can be partly remedied by *white point compensation.* The camera is aimed at something white and the resulting RGB values are stored as r_w, g_w b_w.

Then the values

$$\frac{256 \cdot r}{r_w}, \frac{256 \cdot g}{g_w}, \frac{256 \cdot b}{b_w}$$

are used instead of the r, g, and b from the camera.

Better results can be achieved if subtracting the RGB values obtained for something completely black before doing white point compensation.

By looking at the colour values of a series of objects ranging from black over grey to white off-line in Matlab, you can do non-linear compensation for the best results.

Camera quality (or rather the lack of it...)

The quality of the cameras is bad. You will have to make do with:

- Low framerate. Maybe only one fps.
- Low resolution. A few kilopixels are all you will have time to process.
- Changing brightness levels. Auto-brighness is not your friend.
- Really bad colours. Take test images after auto-brightness has stabilized and try to classify the colours (manually at first) in Matlab.

Many test images, and manual or automatic classification off-line will give the best results for colour classification. Converting to HSV space first might be an advantage if the lighting changes at the competition.

For automatic classification try e.g. non-linear SVM (`lookfor svm` in Matlab).

Colour tracking

The colour tracking demo program (`track.hex`) uses the hue value of the target. Computing the hue for each pixel takes too much time, so a 3D lookup table from RGB to hue is used. See the source code for how to build such a table.

The program finds sequences of matching pixels in rows. Between good matches some not as good matches are allowed, and even an occasional non-matching pixel.

Instead of line based colour tracking, x and y histograms can be used efficiently (see the Eyebot book in the lab). Just computing the centre of mass can also work well.

Manually flashing the OS

If Robios stops working because the FLASH memory has been accidentally erased (could happen from electrical abuse), it must be manually reinstalled.

The serial download mechanism will no longer work, and Robios and an HDT must be flashed using the parallel BDM cable and a DOS program. You will find it on a CD with instructions written on it in the lab. The BDM cables are in the corner close to the whiteboard in the lab.

See also
`http://robotics.ee.uwa.edu.au/eyebot/`
→ HDT for instructions. (Remember we have the 4 mbit FLASH rom.)

On this page you can also find **documentation for all HDT entries** under 'Description of all HDT entries'.

The 'semantics' types are defined in
`/usr/local/eyebot/RoBIOS/include/hdt_sem.h`

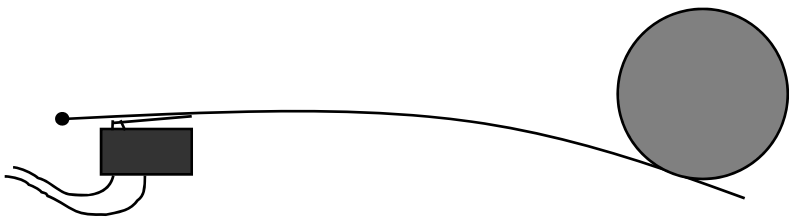
Construction tips

- Build a robust robot. Repairing it during a game gives the opponent a good opportunity to score.
- Make sure the robot will not get stuck on the opponent. A sheet metal border around the robot can be a good idea.
- Build the robot incrementally with lots of testing. Test matches against real opponents are very useful!
- Time is limited. A too complicated design might have to be abandoned due to lack of time.
- Only include a new sensor or actuator if you have good use for it.

Making a bumper switch

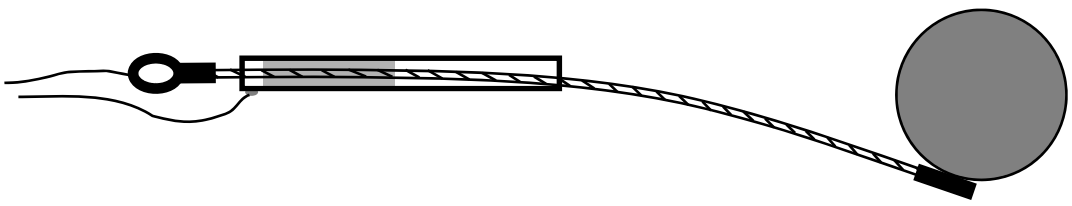
Antenna connected to a microswitch or keyboard switch (a few can be found in the lab).

- Hard to make switch react to pressure from all directions.
- Switch easily breaks if not mounted cleverly.

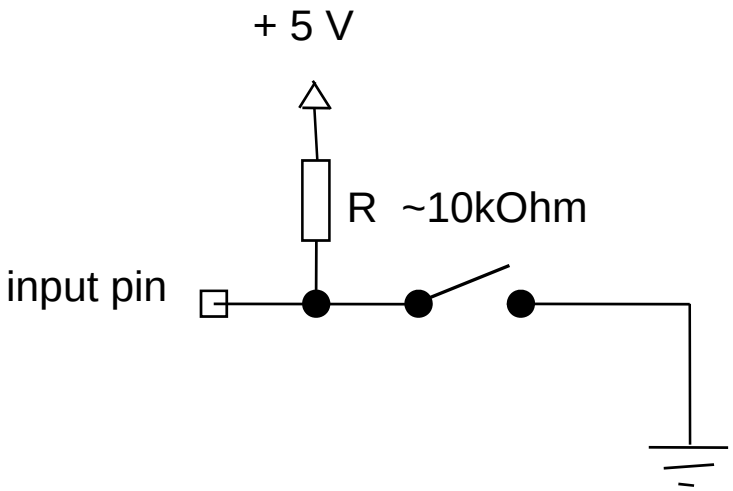


Flexible steel wire mounted in a metal pipe

- Reacts to pressure from all directions.
- Easily manufactured from parts available in the lab.



Connecting a switch (again)



Don’t leave the input pin unconnected at any switch position.

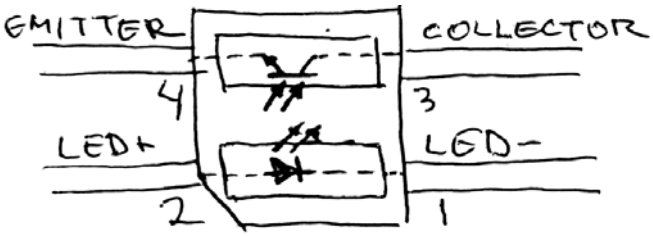
Use the proper connectors (‘crimp terminals’) and plastic housings or heat shrink tubing. Ask for a demonstration of the connectors and the special ‘crimp tool’.

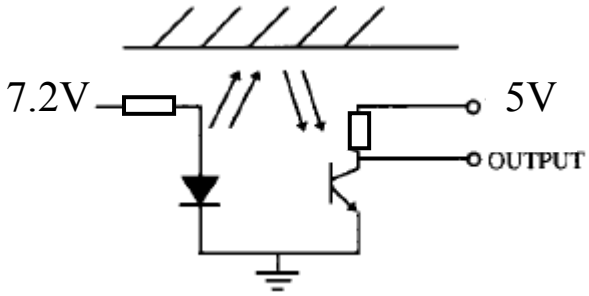
Use the bumper entry in the HDT to enable the `BUMPCheck()` call which gives you a time stamp for bumper events.

Alternatively you can just use the input latches with the `OSReadInLatch()` call.

Reflex detectors

Reflex detectors can be used to sense the reflectivity of a nearby object. The ones in the lab are for a distance of 1-7 mm.





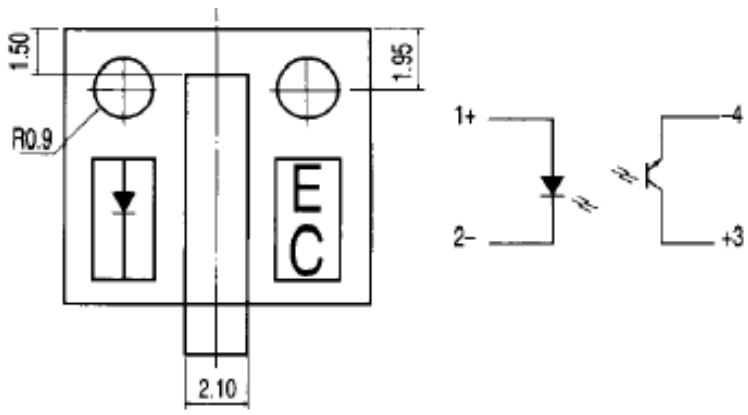
Connect the LED part of the detector in series with a resistor (150-1000 Ohm) to battery voltage.
(The voltage over the LED is fairly constant at about 2V independent of the current, and the max allowed current is about 40 mA.)

The emitter of the phototransistor part of the detector should be grounded, while the collector connects to +5V through ~10 kOhm resistor.

The output signal is then at the collector, which can be connected to an analog input of the Eyebot.

Ask for help if you need it!

Slotted opto detectors



An opaque object in the slot will block the light and thereby affect the output signal.

Electrically, the slotted opto detectors are equivalent to the reflex detectors, and they can be connected in exactly the same way.

Just note how the pins are arranged (it is marked on the capsule as in the figure above).

These slotted opto detectors can be used for making simple optical encoders

The detector can also be combined with a lever, which is depressed into the slot when it makes contact with an object. This could be an alternative for whiskers or bumper switches on your robots.