

Project Lecture B

- Lecture notes on the Project part 2

Robot soccer rules

Programming tips

Robios library calls

Image processing

Construction tips

Robot soccer rules (based on the Robocup F180 league rules)

Construction

- Safety
- Shape
- Colours and markers
- Locomotion
- Wireless communication
- Dribbling
- Kicking

The game

- The referee
- The duration of a match
- The start of a game
- Scoring
- Repairing a robot

Offences

- Holding the ball
- Kicking the ball into the air
- Improper defence of the goal

The competition

- Seeding
- The tournament

Construction part 1

Safety

A robot must not have in its construction anything that is dangerous to it, other robots or humans.

Shape

A robot must fit in a 180 mm diameter vertical cylinder at all times, except for flexible parts that cannot stop the opponent or guide the ball. All such flexible parts must be above the height of the ball.

Colours and markers

All robots must carry a colour marker in the form of a 4 cm high purple paper cylinder visible from all directions. The diameter must be at least 10 cm. The symmetry axis of the cylinder must be vertical, and the lower edge must be between 5 and 10 cm above the floor.

No other colours that can be confused with the ball, goals and field are allowed on the robots.

Construction part 2

Locomotion

The robot must not harm the felt surface of the field when it moves. *Keep this in mind when constructing sliding support points.*

Metal spikes, Velcro, and sand paper are specifically prohibited for the purpose of locomotion. Some suitable supports are now available in the lab.

Wireless communication

All forms of communication with the robot are forbidden, except when the robot is taken off the field for repair.

Kicking

Kicking devices are allowed provided all rules are followed at every stage of the kicking.

Examples of rules to keep in mind are *Safety*, *Shape*, and *Holding the ball*.

Construction part 3

Dribbling

Dribbling devices, which actively exert backspin on the ball, and which keep the ball in contact with the robot, are permitted under certain conditions.

The plane of the exerted spin must be perpendicular to the plane of the field. Vertical or partially vertical dribbling bars, known as side dribblers, are not permitted.

This means any force actively exerted on the ball, and in the plane of contact, must be strictly vertical. Suction devices are therefore forbidden.

The game part 1

Number of players

There is only one robot on each team.

The referee

Starts the game and repositions the ball to the centre point after each scored goal. He also moves the ball to an arbitrary neutral position in case of a deadlock during the game.

The duration of a match

A match will be three, four or five minutes long, for group play, semifinals and final respectively. There are no breaks.

The start of a game

When the game starts both robots must be at their own goal lines with some solid part of their body. The ball is at the centre point.

If both teams in a match prefer to play against the same goal, a coin is tossed to decide the matter.

The game part 2

Scoring

A goal is scored whenever the entire ball is beyond the line defined by the wall fronts at each side of the goal.

Repair

A robot may be taken off the field for a minimum of 30 seconds for repair. This is the only case in which communication with the robot is permitted. After 30 seconds the robot may be put back into play at any time, in that of the defensive corners that is farthest from the ball.

Offences

Holding the ball

A robot is holding the ball if it takes full control of the ball by removing any of its degrees of freedom. This would be the case if for example a suction device prevented the ball from rolling when the robot moves.

Furthermore, at least 80% (75% of the diameter) of the area of the ball must be outside the convex hull of the robot when viewed from above.

A robot holding the ball will be sent off the field for 20 seconds.

Kicking the ball into the air

A robot that kicks the ball more than five centimeters into the air is sent off for 20 seconds.

Improper defense of the goal

A robot may not cross its own goal line with any solid part of its body. If it makes contact with the ball under such conditions a goal is awarded to the opponent.

The competition part 1

Seeding

The competition starts with a seeding round where all robots get two minutes each to score as many goals as possible unopposed. An extra minute is allowed if a robot has not scored in two minutes, but if the robot still does not score it is excluded from the competition.

At the start and after scored goals the ball is put somewhere on the centre line but not on the centre point as in a real game. The robot must not be manually aimed at the ball; it must face the centre point at the start.

To pass the course, a team must demonstrate unopposed scoring at several initial ball positions.

| Number of robots | Group play | Seeding# in groups | Semifinal |
|------------------|------------|--------------------------|-----------|
| 10 | 4+3+3 | (1,4,10,5)(2,9,7)(3,8,6) | 2+1+1 |
| 9 | 3+3+3 | (1,7,6)(2,8,5)(3,9,4) | 1+1+1 |
| 8 | 4+4 | (1,8,4,5)(2,7,3,6) | 2+2 |
| 7 | 4+3 | (1,3,6,5)(2,7,4) | 2+1 |
| 6 | 3+3 | (1,6,4)(2,5,3) | 2+2 |
| 5 | 5 | (1,2,3,4,5) | 4 |
| 4 | 4 | (1,2,3,4) | 4 |
| 3 | 3 | (1,2,3) | 2 (final) |

The competition part 2

The tournament

The robots are divided into groups depending on their success in the seeding round.

The competition part 3

The tournament

In group play, two points are awarded for a win, and one for a draw. In the finals “golden goal” is used to settle a game on an equal result at full time (i.e. the game continues until one robot scores).

If four robots from two or three different groups enter the semifinals they never meet a robot from their own group, and group winners meet robots placing second if possible. If three robots enter there is another round of group play in place of ordinary semifinals. In this case the golden goal rule is not used.

Hardware documentation

A link to datasheets for many of the components used in the course is now on the course web page.

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!*

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⟩, DI | 2 | 0 | 46 - 52(0/1/0) |
| MUL(S/U).L (Op) | ⟨FEA⟩, Dn:DI | 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

CPU properties

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.

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.

See

<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/robios/hdt_sem.h`)

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.
```

This is not necessary for Robios library calls.

The Robios libraries and headers are in the `/usr/local/robios` directory. They are automatically found by the `gcc68` script.

Robios system calls

For a list of all the calls built into Robios, go to
<http://robotics.ee.uwa.edu.au/eyebot/>
→ Controller → Libraries

Image Processing

Key Input

LCD Output

Camera

System Functions

Multitasking

Semaphores

Timer

Download and RS-232

Audio

PSD Sensors

Servos and Motors

V-Omega Driving Interface

Bumper / Infrared Sensors

Latches (digital I/O)

Parallel Port

A/D Converter

Radio Communication

Compass

TV Remote Control

(The mystical ‘semantics’ types are defined in
`/usr/local/robios/hdt_sem.h`)

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, $100/\text{timescale}$ Hz.

Start the PI controller with

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

The 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 in a first stage, and then use all pixels in a few target candidate regions.

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 camera speed 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.

The new wide angle lenses are also bright enough to **confuse auto-brightness completely** when all the strong lamps are aimed down on the field. The result is an all black output image.

Colour classification

To find the different targets (ball, goals, and opponent), all 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:

red – pink – white

Value (intensity):

Changes proportional to power of lights

black – grey – white

Colour classification (cont'd)

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

Before the hue is be 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

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

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

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

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.

More advanced colour classification that was used in last year's competition include *principal component analysis* (Electric Sheep) and a *support vector machine* (Robbe). See their reports on the web!

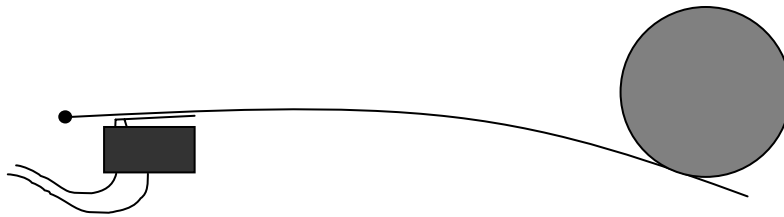
Construction tips

- 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

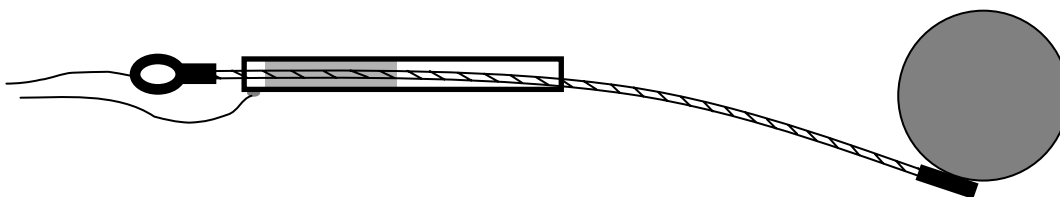
Antenna connected to a microswitch or keyboard switch (a few can be found in the lab).

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

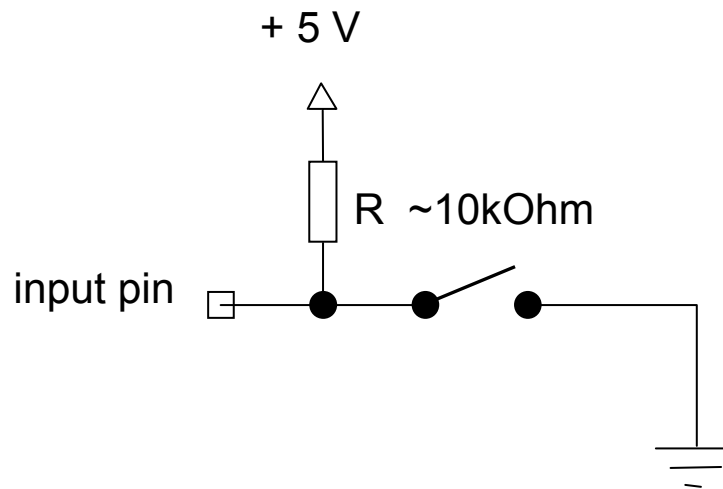


Flexible steel wire mounted in a metal pipe

- 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.