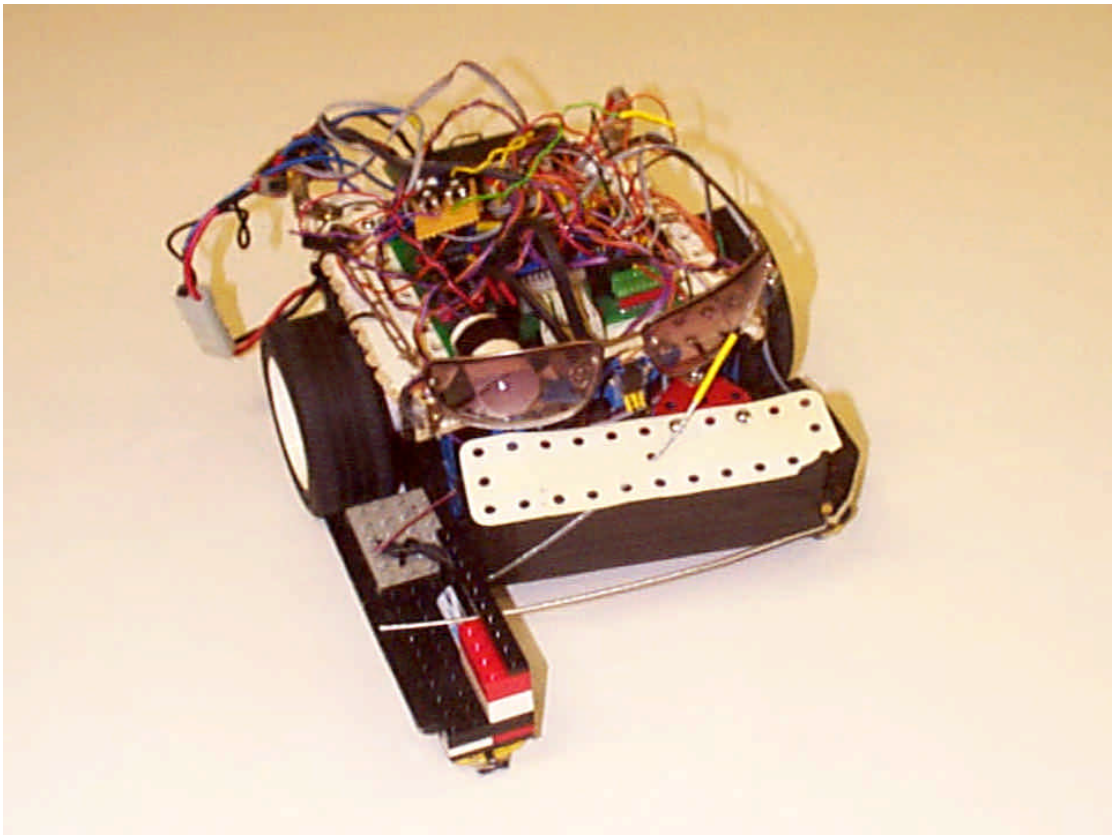
The Matris Reengineered

A project report in the course 2D1426, Robotics and Autonomous Systems



By
Alexander Enstedt
Magnus Jansson
Ville Lumikero
Christian Schüldt

030528, KTH

Abstract

This report details the construction and design of and autonomous table hockey playing robot. The project was a compulsory part of the course 2D1426, Robotics and Autonomous Systems at KTH.

The result of the project was that we were able to build a robot just in time for the final tournament, in which our robot did fairly well.

Table of contents

Table of figures

Introduction

This project report details the construction and operation of the robot The Matris Reengineered (or Matris for short), which was designed to play table hockey. Both the construction of the robot and this report are compulsory parts of the course 2D1426, Robotics and Autonomous Systems at KTH.

Objectives

During the course the participants build autonomous robots in teams of three to four persons. The robot's construction and programming are designed to make the robot capable of playing table hockey using techniques taught in the course lectures. In the end of the course the robots play a tournament against each other.

Table hockey

The game of table hockey is played in table-top rink that is approximately 2.4 x 1.2 meters in size. The objective is to have an autonomous robot guide a puck into the opposing goal zone, and to hinder the opponent from driving the puck into your zone. For a complete description of the rules see [1].

Available hardware

Each group had access to a starting kit, which in most cases consisted of the robots built a year before. An overview of the most basic components is presented below:

- **DC motors:** 2 12V motors with exchangeable gears
- **PIC16F877 microcontroller:** a single chip microcontroller unit (MCU) with a 20 MHz processor, 368 bytes of RAM and 8 k x 14 bits of program memory
- **Motherboard:** a purpose built board with connections for the MCU, motors etc. A coprocessor for IR signal processing and a voltage booster card to drive the motors are coupled to the motherboard
- **various infrared sensors:** different types of infrared (IR) diodes and sensors are used for detecting the objects and markings in the hockey rink
- **Lego and Meccano:** used for structural parts of the robot

Other components, such as various types of tactile sensors, could be built using the hardware provided in the laboratory. It was also possible to use parts not provided by the laboratory, as long as they did not break the rules of the game.

Constraints

Various constraints applied to the construction and programming of the robot. First and foremost was the timing constraint. The competition date dictated the working pace for the whole project and affected the project planning to a great extent. We will return the importance of this constraint in later sections.

The rules of the game also placed more or less strict requirements on the project. The most important constraints are listed below. These constraints are the ones that directly affected the design and construction of the robot, while the other rules rather affected the programming.

- **Robot diameter** must not more than 25 cm, excluding the stick.
- **Form**: the edge of the vertical projection of the robot body may not have any concave parts, excluding concavities formed by the wheels and parts that are above puck level.
- **Stick**: the stick may not extend more than one puck diameter from the body, not be wider than one puck radius and not have any joints or concavities. When guiding the puck, the stick may not occlude the puck from any direction.
- **Scoring**: to qualify into the tournament, the robot must be able to score at least one goal during a two minute unopposed qualifying run. When and only when the robot is scoring or is close to scoring it must play a tune. A goal is ruled out if any non-flexible part of the robot goes into the goal zone.
- **No devices** that could be used to harm the other robot may be mounted on the robot.

For a complete description of the constraints, please see [1].

Robot Structure

Our strategy with the robot was to make it small and fast. We decided to use a plywood board as a base because the material is easy to work with. To get a low centre of gravity we placed the motors in the middle on the top side of the board. We shaped the rear of the board as a circle to prevent the robot from getting stuck when it turned near walls.

At the front of the robot we cut out the stick. Our initial idea was that we should score by pressing the puck in to the goal with the help of the front of the stick. To make it easier for the robot to hit the puck with the front of the stick we made it as wide as the rules allowed. We made the front of the robot sloping so that when we got the puck the forces on the puck would push it towards the stick when driving forward. Below you can see a sketch of our creation (Figure 1).

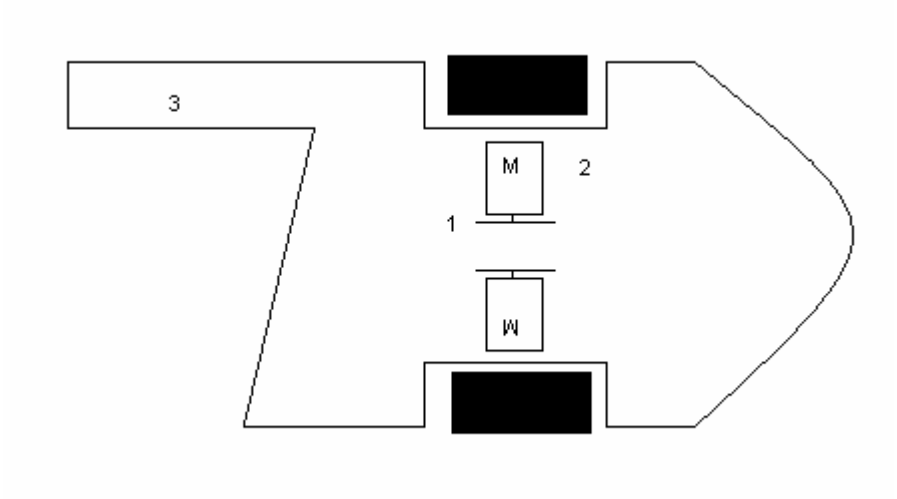


Figure 1. (1) OH-film, (2) Motor, (3) Stick

Motors

We used two 12 volt motors one at each side of the robot. These motors were equipped with 10:1 scale gearboxes. The wheels were connected motor axes with epoxy glue. We used straps to mount the motors, three on each motor, and we glued them to the board to be sure they wouldn't move.

Our motors were very old and had plastic gearboxes. Plastic gearboxes aren't recommended because they break down easily under strain. We had to replace both of our boxes at different times of the construction. Since the motors were old the difference in acceleration and top speed between them was large.

Sensors

IR sensors

The puck and the two goals emit infrared light. The different light sources have differently modulated signals, so that they can be separated from each other. The PIC coprocessor handles the filtering of the signals. An interface is provided to read the IR values of the puck, the defenders goal and the offenders goal.

The IR light is detected using IR sensors (see Parts List) that have a detection range of almost 180° (see [2]), sketched symbolically in Figure 2.This means that using only one sensor one can not tell the difference of a close lying IR emitting object at an angle of, say 45° from an object situated straight ahead, but further from the sensor. To solve this problem we came up with three possible solutions.

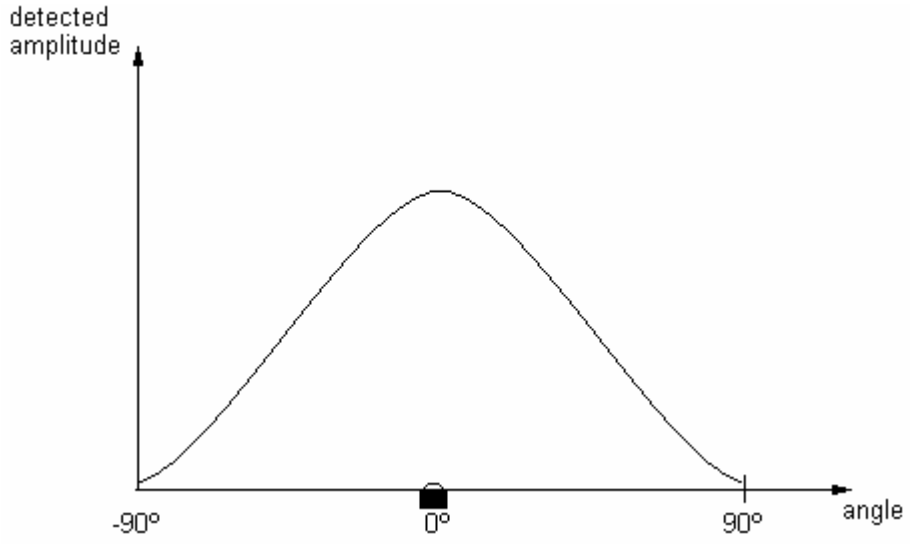


Figure 2. Symbolic graph of IR sensor detection capabilities.

The first one was to just place IR sensors on the robot facing in different directions. The one sensor that detected the highest amplitude for a specific object would decide in which direction the object was. The problem with this solution was that the motherboard could only connect a maximum of five IR sensors, so the accuracy of this option was quite limited.

To increase the accuracy we tried another scheme. We attached an IR sensor to a servo motor that could be periodically rotated by the controller (see Figure 3). This way, we could determine which angle produced the maximum detected IR value, i.e. the correct angle to the IR emitter. Now the angle accuracy was only determined by the servo's accuracy, and we could correctly determine angles up to an accuracy of approximately ten degrees.

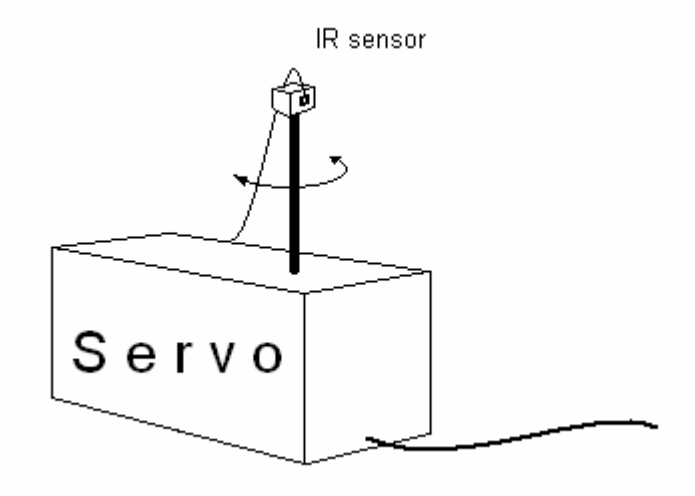


Figure 3. Servo controlled IR sensing.

This approach was an apparent success, but after testing it we were forced to abandon it because of its tardiness. We could only get new readings about every second. Even with optimizations this approach would probably not be fast enough.

We decided to go back to our previous approach and try to modify it to get a better accuracy. If we could interpolate the angles for objects that were located between the IR sensors we would increase the accuracy significantly.

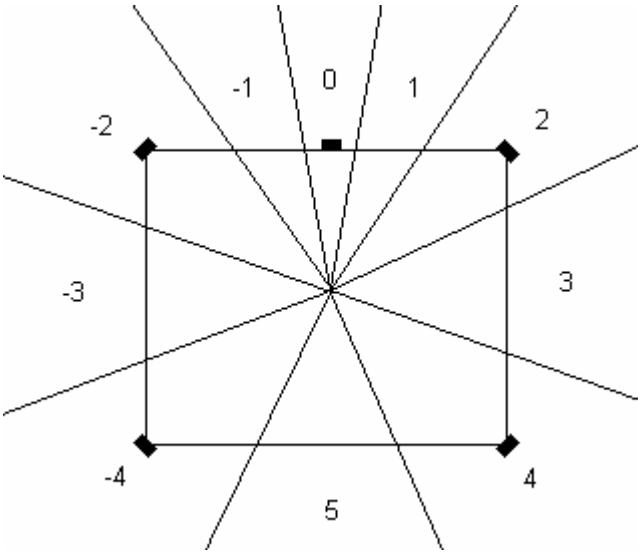


Figure 4. Placing of IR sensors and the detectable angles.

By placing three sensors facing forward at different angles and two facing backwards we could get a higher resolution when detecting objects in front of the robot (see Figure 4).

As mentioned earlier it is trivial to determine which of the sensors has the strongest reading of an object, but to determine if an object is situated between two sensors is not just as easy. We could say that an object lies between two sensors if their respective readings are of the same order of magnitude. This could be determined by dividing one value with the other, but the process of division is very time consuming on the PIC processor.

Our solution to the problem was to take the two largest values and right-shift them one bit at a time until one of them equals zero. Then we look at the other value and see how large it is. If it is smaller than some threshold value, the two values are of the same order of magnitude, and the detected object lies between the two sensors (odd angles in Figure 4). This scheme gave us reasonably accurate readings for the puck and the goals.

Reflex detectors

Reflex detectors where used in order to determine if the robot is located on a white (the "ice") or black (the goal zones or centre line) surface. Three reflex detectors were used; one under the tip of the stick, one under the front of the robot and one under the back (see Parts list). Each detector was glued to a piece of Meccano, fastened underneath the robot. The detectors were placed so that they could "see" the ground through holes in the Meccano pieces. In this way they also got very close to the ground (a few millimetres).

Since the pieces of Meccano were also used to support the robot, a few bits of Velcro were cut out and mounted underneath to reduce the friction. This worked very well, except when dirt got stuck to the Velcro and blocked the reflex detector's visibility.

The IR-diode in the reflex detectors were connected in series to the LED-driver board and the collectors of each reflex detector were connected to an analogue input pin on the PIC-controller and a pull-up resistor. Thus, one needed three A/D conversions to get readings from all detectors. The A/D readings were then thresholded in order to determine if the surface beneath was black or white.

Optic encoders

To be able to detect speed and acceleration we used slotted optodetectors with encoder discs. Our encoder discs were circular pieces of OH-film with interleaved black and transparent fields. These discs were glued on the axis on the motor side before the transmission, which yielded a better resolution. To be able to calculate speed and direction of rotation we had to use two optodetectors on each disc.

First we tried to use a higher resolution OH-film, with 16 black & transparent fields, to get better readings but placing the optodetector correctly was too difficult. Instead, we decided to use the lower resolution OH-film with 4 fields. This gave us 80 pulses for each turn of the wheel.

The slotted optodetectors function similarly to the reflex detectors. The difference is that the LED and detectors are placed on each side of a fork, making them able to detect if something passed between them. The detectors were connected to Schmitt triggers on the PIC's ports so that their output could be digitally read.

The IR diodes in the optodetectors were connected in series to the LED-driver board and the collector of each reflex detector was connected to an analog input pin on the PIC-controller.

Puck contact sensors

The robot must be able to tell if it has the puck or not. This proved to cause a lot of problems. The first solution tried was with one sensor, consisting of a few decimetres of semi-flexible metal wire slipped through a small copper cylinder, with some heat shrink tubing partly isolating the two pieces from each other. When the wire is bent however, there will be contact. The cylinder was connected to ground and the wire was connected to an input pin on the PIC (using a built-in pull-up resistor).

The sensor was mounted on the stick so that the wire would cover most of the robot's front. The idea was that when the robot gets the puck, the puck will push the wire, which then comes in contact with the metal cylinder, sending a signal to the PIC.
The problem with this method was that if the robot got the puck a bit away from the stick, the metal wire wasn't bent enough to touch the cylinder.

Because of this, another sensor was added at the other side of the robot, resulting in another problem: the combined stiffness of the two wires pushed the puck forward when moving the robot.

The final solution was to fasten a piece of metal wire to the front of the robot and have another flexible piece of metal wire in front of the robot body. The puck will then make the flexible wire to come in contact with the fastened wire and thus make a closed circuit and cause a signal to be sent to the PIC.

Locomotion

Speed control

The purpose of speed control is to control rotating speed and the wheels. Our speed control system consists of three parts: speed measurement, PI-control system and control interface; described in the following sections.

Speed measurements

Speed measurements are done by optical encoders mounted at the motor side of the gear. The optical encoders create a pulse train of 4 pulses/revolution. Since we also are interested in the rotation direction we use two encoders on each wheel, the encoders generate phase-shifted pulse trains. The phase-shifted pulse train is read by a routine and translated into a byte that is used as position measurement. The position can be found in the two variables enc1_pos and enc2_pos, the variables are updated by an interrupt routine and they are increased by 80/revolution (one step per pulse, two encoders per wheel and 10:1 gearbox).

The speed is calculated (in the interrupt routine) as the difference in the position variables over a given time interval. After calculation, the speed is stored in the global variables Speed1 and Speed2, which makes the speed for motor1 and motor2 available for all routines.

One problem is to decide the time interval for speed measurements: if we make it too short we will get to low resolution, and if we make it too long we will get a slow response to speed changes. We get around this problem by choosing the following strategy: we calculate the speed each 10 ms but we take the difference over the last 40 ms. We selected these time intervals by testing different intervals and we found out that 10 ms and 40 ms worked out well.

With this speed measurement, the wheels were able to reach speeds between –20 and +20 (without load).

PI-control

The basic way to control the motors is to set the power that will be fed to them. The power can be set by calling the routine *motors(power1, power2)* where *power1* and *power2* are integers between –128 and +127. The problem is that the motors don't respond in the same way for the same input. We solved this problem by programming a forward connected PI-control system.

*power1=wanted_speed1*f1+err1*f2+integrated_err1+err_diff*f4*
*power2=wanted_speed2*f1+err2*f2+integrated_err2-err_diff*f4*

err=wanted_speed-real_speed
err_diff=err1-err2

The first part of the expression is a guess of what power we should use; to set *f1* we made the assumption that the speed is a linear function of the power. This gives us:

*max_Power=max_speed*f1*
*127=20*f1*
f1=6

The linearity-assumption is quite inaccurate and that's the reason for using a PI-control system. The parameter *f2* was selected by testing and we found that 8 was a good choice for the parameter.

One might wonder why we don't use a factor *f3* on the integrated error. That's because we adjust the impact of the integrated part by selecting how often we will sum up the errors. A shorter summing interval is the same as a larger *f3*.

The purpose of the last part of the power-expression is to reduce the difference between the motors. If one of the motors is going faster than the other, the robot will not be able maintain course. If on the other hand both of the motors are going slower, the robot will be able to stay on course.

The control system is partly implemented inside the interrupt routine. The speed calculation and integration of the error (in the PI-controller) is handled in automatically by the interrupt routine. One important part of the control systems must be handled outside the interrupt routine. We must continuously call the function `upd_motors()` to calculate the power and feed the motors with this new power. The reason why this is not done inside the interrupt routine is

that this demands a series of function calls and function calls should be avoided in the interrupt routine because of the limited function call stack on the PIC.

Control system interface

The interface is a set of *routines* that can be called to tell the control system how to behave. The routines are:

```
void set_speed(wanted_speed1,wanted_speeed2)
void turn_pos(pos_diff1, pos_diff2)
void turn(object)
void follow_object(object)
```

`set_speed` sets the `wanted_speed`-parameters in the power-expression for the control system. `set_speed` is the function we use most of the time to control the motors.

We use `turn_pos` when we wish to do precision motion. The function makes it possible to turn the wheels a given angle; more precisely `turn_pos` drives the motors until the wanted difference `pos_diff` is achieved in the position variables `enc1_pos` and `enc2_pos`.

Both of the functions `turn` and `follow_object` are used to get to either the puck, the offensive goal or the defensive goal as fast as possible.

The function `turn` stops the robot and makes a turn on the spot to get aligned with the given object. `follow_object` tries to follow the given object by going in softer arc towards the object. How soft the arcs will be are determined by the angle (that is calculated by the navigation routine), a smaller angle results in a softer arc.

Behaviour

The robot is controlled by a reactive behaviour-based system. The design method used was a hybrid between situated activity design and an experimentally driven design.

While the robot works it switches between a set of states. A finite state acceptor diagram is depicted in Figure 5. The state the robot is in determines how the robot should react on sensory data. E.g. if the robot is in the state STATE_FIND_PUCK, it only attention to IR light from the puck; if the robot is in the state STATE_FIND_GOAL it pays no attention to the IR light from the puck. Each state decides when to set the robot in another state and what the new state will be.

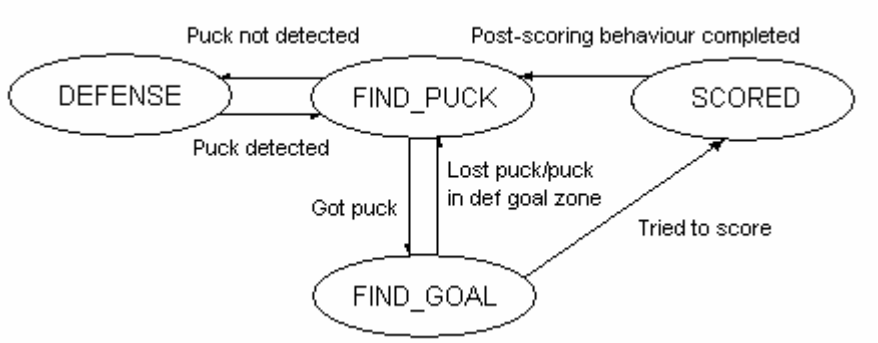


Figure 5. Finite state acceptor diagram.

The robot can be in four different states: STATE_FIND_PUCK, STATE_FIND_GOAL, STATE_SCORED and STATE_DEFENSE.

STATE_FIND_PUCK is the state the robot starts in. In this state the robot follows the puck until it gets it (the bumber sensor in the front gets activeis activated). When the robot gets the puck its state is changed to STATE_FIND_GOAL. If the robot is in STATE_FIND_PUCK but cannot find the puck it will set it self in STATE_ DEFENSE. STATE_FIND_PUCK uses the routines `turn` and `follow_object` to get to the puck.

STATE_FIND_GOAL is the state where we try to score. In this state the robot goes towards the goal in high speed and stops as soon as the club enters the goal zone. When the robot stops the puck will hopefully glide into the goal, and after testing we found out that this was often the case. STATE_FIND_GOAL uses the routines turn and `follow_object` to get to the goal. When the robot believes it has scored it will set itself in STATE_FIND_SCORED.

We experimented with other scoring strategies as well, e.g. a strategy where we stopped by the goal and went back a few centimetres and then pushed the puck into the goal with the stick. We found out that this strategy was not as good as the one we finally chose.

STATE_DEFENSE is the state the robot goes into when it doesn't see any offensive opportunities, i.e. when it can't see the puck. In STATE_DEFENSE the robot goes towards the defending goal and stays there (just outside the goal zone) until it can see the puck, whereupon it sets it self in STATE_FIND_PUCK.

STATE_SCORED is the state the robot is in when it just have made a goal. In this state it plays the scoring melody and goes back towards the middle zone of the rink.

Results and conclusions

The hours before the competition turned out to be very hectic due to lots of hardware failures. First one of the gearboxes broke down, then the other one. The optoencoders failed, the encoder wheels kept falling off the motors and last but not least, the PIC seemed to behave very strangely. Against all odds, we managed to fix the problems and during the qualification round of the competition, our robot scored 6 goals during the first 30 seconds. After that, it unfortunately, got stuck and didn't manage to score any more goals.

In the competition against other robots, however, our robot didn’t do that well. A big part of the problem was lack of robustness. In the first match, a cable to one of the motors came loose and we had to take the robot off the ice for 30 seconds to repair it. During that time our opponent managed to score a few goals. In the second match, our PIC locked up and we had, again, to take the robot off the ice for 30 seconds of repair. One thing, beside the (lack of) robustness we could have improved was the robots ability to move away when it got stuck. Because of lack of time, the code that was intended to detect when the robot got stuck and move away wasn’t efficient enough.

Another thing we could have done differently was to use an IR-sensor to detect if the robot has the puck. Although our detector worked quite well, it seemed to be a better idea to use an IR-sensor for that. Many of the other robots used this strategy quite successfully.

The PI-regulator turned out to be a good thing, but we could have needed some more time to optimize the parameters.

References

[1] Course Notes V,Robotics and Autonomous systems 2003, http://www.nada.kth.se/kurser/kth/2D1426/L5.pdf

[2] TSL261 Data Sheet, http://www.taosinc.com/pdf/tsl260.pdf

Parts List

- 1 PIC PIC16F877 40-pin 8-Bit CMOS FLASH Micro controller. Manufacturer: Microchip Technology Inc.Part no:PIC16F877.
- 1 PIC (for IR) PIC16F876 20-pin 8-Bit CMOS FLASH Micro controller. Manufacturer: Microchip Technology Inc.Part no:PIC16F876.
- 1 Motherboard.
- 1 Boostercard - 7.2V to 14.4V.
- 1 LED Driver board.
- 1 Serial communication board.
- 1 LCD Powertip PC1602D A.
- 2 12V motors HL149 12V 10:1 Motor.
- 1 100 Ohm speaker.
- 5 IR sensors TSL261 IR Light-to-Voltage Optical Sensor.Manufacturer: Texas Advanced Optoelectronic Solutions (TAOS).Part no:TSL261.
- 3 reflex detectors ITR8307 Subminiature High Sensitivity Photo Interrupter. Manufacturer:Everlight.Part no:ITR8307.
- 4 slotted opto detectors ITR 8010 General Purpose Photo Interrupter.
- 2 7.2V NiCd accumulator packs

Appendix A, Source Code

mainfile.C

```
#include <pic.h>
#include <conio.h>
#include "defines.h"
#include "lcd.h"
#include "ir_comm.h"
#include "serialio.h"
#include "intr.h"
#include "sprint.h"
#include "pwm.h"
#include "adc.h"
#include "button.h"
#include "stearing.h"
#include "ir_find.h"
#include "g3_globl.h"

#define VERBOSE 1

// no code protect, no WDT, no BOD, power up timer, HS Xtal
// (has no effect when writing the program using the
// boot loader)
__CONFIG(0x3FB2);


const char
matrix[]={0x06,10,0x16,30,0x06,10,0x16,30,0x06,10,0x16,30,0x06,10,0x16,60,
          0x06,10,0x16,30,0x06,10,0x16,30,0x06,10,0x16,40,
          0x06,10,0x17,30,0x06,10,0x17,30,0x06,10,0x17,30,0x06,10,0x17,60,
          0x06,10,0x17,30,0x06,10,0x17,30,0x06,10,0x17,40,0,0};



main() {


     int last_has_puck_time=0;
     volatile char is_scoring = 0;

     init_robot();


     // main loop
     for(;1;)
     {
          handle_button(0);
          navigate();
          bumper();
          //Kontrollutskrifter
          str_print1("ST:             ", 0);
          int_print1(robot_state, 3);
          if (robot_state==STATE_FIND_PUCK) int_print1(angles[PUCK], 5);
          if (robot_state==STATE_FIND_GOAL) int_print1(angles[OFF_GOAL], 5);
          if (robot_state==STATE_DEFENSE) int_print1(angles[DEF_GOAL], 5);
          int_print1(FRONT_IN_ZONE, 8);
          int_print1(BACK_IN_ZONE, 9);
          int_print1(CLUB_IN_ZONE, 10);
```

```
//decide what to do
switch(robot_state)
{
     case STATE_FIND_PUCK:
          if (HAS_PUCK)
          {
               set_state(STATE_FIND_GOAL);
          }else if (angles[PUCK]==6){
               set_state(STATE_DEFENSE);
          }else if ((angles[PUCK]<-1)||(angles[PUCK]>1)){
               str_print2("TurnP", 0);
               turn(PUCK);
          }else{
               follow_object(PUCK);
               if(FRONT_IN_ZONE || CLUB_IN_ZONE){
                         set_speed(-10,-10);
                         delay(400);
               }
               if(BACK_IN_ZONE){
                         set_speed(10,10);
                         delay(400);
               }

          }
          break;

     case STATE_FIND_GOAL:

          if (HAS_PUCK)
          {
               last_has_puck_time=soft_time();
          }else if (soft_time()-last_has_puck_time>1000){
               set_state(STATE_FIND_PUCK);
               last_has_puck_time=0;
          }

          if (((CLUB_IN_ZONE)||(FRONT_IN_ZONE))&&(is_near_goal!=2))
          {
               set_state(STATE_SCORED);
               last_has_puck_time=0;
               clear_disp();
               set_speed(-10,-10);
               upd_motors();
          }else if(((CLUB_IN_ZONE)||(FRONT_IN_ZONE))&&(is_near_goal==2)){
               turn_pos(-15, -5, -10);
               set_state(STATE_FIND_PUCK);
               last_has_puck_time = 0;
          }else if(angles[OFF_GOAL]==6){
               str_print2("No Goal    ", 0);
               set_speed(-5,5);
          }else if ((angles[OFF_GOAL]<-1)||(angles[OFF_GOAL]>1)){
               str_print2("T_g     ", 0);
               turn(OFF_GOAL);
          }else{
               follow_object(OFF_GOAL);
               str_print2("F_g      ", 0);
          }
          break;
     case STATE_DEFENSE:
          if ((CLUB_IN_ZONE)||(FRONT_IN_ZONE))
          {
               turn_pos(-40,-40, -10);
          }else if(BACK_IN_ZONE){
               turn_pos(20,20, 10);
          }else if(angles[PUCK]!=6){
               set_state(STATE_FIND_PUCK);
          }else{
               follow_object(DEF_GOAL);
```

```
                    }
                    str_print2("!     ", 0);
                    int_print2(Speed1_wanted, 1);
                    int_print2(power1, 4);
                    int_print2(Speed2_wanted, 9);
                    int_print2(power1, 12);
                    break;
               case STATE_SCORED:
                    if (last_has_puck_time==0)
                    {
                         play_tune(matrix);
                         set_speed(-12,-12);
                         upd_motors();

                         delay(700);

                         //turn_pos(-100, -100, -10);
                         //turn_pos(0, -100, -10);
                         last_has_puck_time=soft_time();
                         if (last_has_puck_time==0) last_has_puck_time=1;

                    }else if (!passed(last_has_puck_time+1500)){
                         str_print2("GO HOME", 0);

                         if(angles[DEF_GOAL]==6){
                                   str_print2("No Goal    ", 0);
                                   set_speed(5,-5);
                         }else if ((angles[OFF_GOAL]<-1)
                                                  ||(angles[OFF_GOAL]>1)){
                                   str_print2("T_dg     ", 0);
                                   turn(DEF_GOAL);
                         }else{
                                   follow_object(DEF_GOAL);
                         }


                    }else{
                         last_has_puck_time=0;
                         set_state(STATE_FIND_PUCK);
                    }
                    break;
               default:
                         str_print2("default",0);
                    break;
          }
          upd_motors();
     }
     clear_disp();

     str_print1("Exit",0);
     return;
}
```

g3_globl.h

```
#define STATE_FIND_PUCK 0
#define STATE_DEFENSE 1
#define STATE_FIND_GOAL 2
#define STATE_SCORED 3
#define STATE_MAKE_SCORE 4

#define set_state(new_state) if(robot_state!=new_state) (robot_state=new_state)

#define IS_STATE_FIND_PUCK (robot_state==STATE_FIND_PUCK)
#define IS_STATE_DEFENSE (robot_state==STATE_DEFENSE)
#define IS_STATE_FIND_GOAL (robot_state==STATE_FIND_GOAL)
#define IS_STATE_SCORED (robot_state==STATE_SCORED)

#define HAS_PUCK bittst(buttons[0],IS_DOWN)


// wait for s ms
#define delay(s) for(sometime=soft_time();!passed(sometime+s);upd_motors())


#define str_print1(my_str, position) if(VERBOSE) lcd_print1(my_str, position)
#define str_print2(my_str, position) if(VERBOSE) lcd_print2(my_str, position)
#define str_print_at_cursor(my_str) if(VERBOSE) lcd_print_at_cursor(my_str)

#define int_print1(my_int, position) if(VERBOSE) (sprint16(buffer, my_int),
lcd_print1(buffer, position))
#define int_print2(my_int, position) if(VERBOSE) (sprint16(buffer, my_int),
lcd_print2(buffer, position))
#define int_print_at_cursor(my_int) if(VERBOSE) (sprint16(buffer, my_int),
lcd_print_at_cursor(buffer))


extern bank2 volatile unsigned char robot_state;
extern bank2 unsigned int sometime;
extern bank1 char buffer[7];
extern const char state_switch_tune[];
extern bank2 unsigned char sample;

void init_robot(void);
```

g3_globl.c

```
#include <pic.h>
#include <conio.h>
#include "defines.h"
#include "lcd.h"
#include "ir_comm.h"
#include "intr.h"
#include "sprint.h"
#include "pwm.h"
#include "adc.h"
#include "button.h"
#include "stearing.h"
#include "ir_find.h"
#include "g3_globl.h"


#define VERBOSE 1

bank2 volatile unsigned char robot_state;
bank2 unsigned int sometime;
bank1 char buffer[7];
bank2 unsigned char sample;

const char state_switch_tune[]={0x16, 20,0x00, 10, 0,0};

void init_robot(void)
{
     // IR sensors 0-2
     //ir_requested = 0b111;
     init_soft_tmr();
     init_pwm();         //initialise pwm for motor control
     ir_init();          //initialise IR co-processor communication



     // 3 pinnar A/D (RA3, RA1, RA0)
     ADCON1=0b00000100; TRISA0=1011;


     // enable interrrupts
     PEIE=1;             // peripheral interrupts enable
     ei();               // global interrupt enable

     set_lcd_power(1);              //turn on lcd
     for(sometime=soft_time();!passed(sometime+100););      // wait for 100ms

     // and then
     init_lcd();
     init_enc();
     init_buttons();
     init_sound();

     robot_state=STATE_FIND_PUCK;
     return;

}
```

ir_find.h

```
#define FRONT_IN_ZONE (is_near_goal && reflex[FRONT_REFLEX]>REFLEX_THRESHOLD)
#define BACK_IN_ZONE (is_near_goal && reflex[BACK_REFLEX]>REFLEX_THRESHOLD)
#define CLUB_IN_ZONE (is_near_goal && reflex[CLUB_REFLEX]>REFLEX_THRESHOLD)

#define REFLEX_THRESHOLD 170
#define FRONT_REFLEX 0
#define BACK_REFLEX 1
#define CLUB_REFLEX 2

extern bank2 signed char angles[];
extern bank2 unsigned char reflex[], is_near_goal;

void navigate();
```

ir_find.c

```
#include <pic.h>
#include "defines.h"
#include "ir_comm.h"
#include "adc.h"
#include "ir_find.h"
#include "g3_globl.h"
#include "sprint.h"
#include "lcd.h"

#define VERBOSE 0

// Threshold for IR-detectors to determine if we're near goal
#define NEAR_GOAL_THRESHOLD 200

// Puck/Off goal/Def goal
bank2 signed char angles[] = {0,0,0};
bank2 const signed char angle[] = {0, 2, -2, 4, -4};
bank2 unsigned char reflex[] = {0,0,0}, is_near_goal;

// "filter" to avoid cross-talk
void filter_ir(void)
{
     char sens, targ;
     unsigned int tmp;

     for (sens=0; sens<=4;sens++)   //loopar alla sensorer
     {
          for (targ=0;targ<=2; targ++) //loopar puck/goals
          {
               if (targ==0)
               {
                    tmp=ir_value(sens, 1)+ir_value(sens, 2);
               }else if (targ==1){
                    tmp=ir_value(sens, 0)+ir_value(sens, 2);
               }else{
                    tmp=ir_value(sens, 0)+ir_value(sens, 1);
               }
               tmp=tmp>>6;
               if ((signed int)(target_map[sens][targ]-tmp)<=0)
               {
                    target_map[sens][targ]=0;
               }else{
                    target_map[sens][targ]=target_map[sens][targ]-tmp;
               }
          }
     }
     return;
}

// Update puck/goals angles and reflex detectors
void navigate() {
     unsigned int signal_ampl[] = {0,0,0};
     int value, ir_max, ir_max2, temp1, temp2;
     unsigned char i, j, k, ir_max_sensor, ir_max_sensor2;
     unsigned int sample;

     ir_wait_for(4);
     filter_ir();

     // loop through puck/goals
     for(j = 0; j < 3; j++) {

          // Tröskelvärden
          ir_max = 10;
```

```
          ir_max2 = 10;

          // which IR-sensor gives highest/almost highest readings?
          for(i = 0; i<5; i++){
               value = ir_value(i, j);
               if(value > ir_max) {
                    ir_max2 = ir_max;
                    ir_max_sensor2 = ir_max_sensor;
                    ir_max = value;
                    ir_max_sensor = i;
               } else if(value > ir_max2) {
                    ir_max2 = value;
                    ir_max_sensor2 = i;
               }

          }
          // Snäva vinklar frammåt
          if(ir_max_sensor == 0) k = 1;
          else k = 4;

          // Någon över tröskelvärdet?
          if(ir_max == 10 && ir_max2 == 10) angles[j] = 6;
          else {
               // Stor skillnad mellan dem?
               for(i=0;i<14;i++) {
                    temp1 = ir_max>>i;
                    temp2 = ir_max2>>i;

                    if(temp1 == 0 || temp2 == 0) {
                         if(temp1 > k) {
                                   // temp1 bestämmer riktningen
                                   angles[j] = angle[ir_max_sensor];
                                   signal_ampl[j] = ir_max;

                         } else if(temp2 > k) {
                                   // temp2 -"-
                                   angles[j] = angle[ir_max_sensor2];
                                   signal_ampl[j] = ir_max2;

                         } else {
                                   // mittemellan - bak
                                   if((ir_max_sensor == 3 && ir_max_sensor2 ==
                           4) || (ir_max_sensor == 4 && ir_max_sensor2 == 3)) {
                                             angles[j] = 5;
                                             signal_ampl[j] = (ir_max +
                                                       ir_max2)<<1;
                                   }
                                   // mittemellan - annan
                                   else {
                                             angles[j] = (angle[ir_max_sensor] +
                                                       angle[ir_max_sensor2]);

                                             if(angles[j] >= 0) {
                                                       angles[j]=angles[j]>>1;

                                             } else {
                                                       angles[j]=(-
angles[j])>>1;
                                                       angles[j] = -angles[j];
                                             }
                                             signal_ampl[j] = (ir_max +
ir_max2);
                                   }
                         }
                         break;
                    }
               }
          }
```

```
	}

	// sample from reflex-detectors
	reflex[FRONT_REFLEX] = sample_ad_channel(0);
	reflex[BACK_REFLEX] = sample_ad_channel(1);
	reflex[CLUB_REFLEX] = sample_ad_channel(3);

	// are we near any goal?
	if(signal_ampl[OFF_GOAL]>NEAR_GOAL_THRESHOLD ||
					signal_ampl[DEF_GOAL]>NEAR_GOAL_THRESHOLD){
		if(signal_ampl[DEF_GOAL]>signal_ampl[OFF_GOAL])
			is_near_goal = 2;
		else
			is_near_goal = 1;
	}
	else{
		is_near_goal = 0;
	}

}
```

stearing.h

```
//macro to set wanted speed.
#define set_speed(s1, s2) (Speed1_wanted=s1, Speed2_wanted=s2)
#define turn_left(s) set_speed(0, s) //motors( -s, s)
#define turn_right(s) set_speed(s, 0) //motors( s, -s)
#define high_turn_speed 30                                                    //speed to
turn with when adjusting for puck or goal.
#define low_turn_speed 15

#define re_speed 15      //speed to go rewers if stuck with a wall
#define re_time 5000     //time to go rewerse
#define max_speed 10

//global variables for speed measurements
extern signed char Speed1, Speed2;

//global variables for speed control
extern bank2 signed char Speed1_wanted, Speed2_wanted;
extern bank2 signed int speed_control_ipart1, speed_control_ipart2;
extern bank2 signed int power1, power2;
extern bank2 signed char err1, err2, err_diff;
extern bank2 unsigned char spdcont_clock;
extern bank2 int time;
//function prototypes
//void set_speed(signed char s1, signed char s2);
void upd_motors(void);
void turn(char object);
void follow_object(char object);
void turn_pos(signed char pos1_diff, signed char pos2_diff, signed char speed);
void bumper();
```

stearing.c

```
<pic.h>
#include <conio.h>
#include "defines.h"
#include "lcd.h"
#include "ir_comm.h"
#include "serialio.h"
#include "intr.h"
#include "sprint.h"
#include "pwm.h"
#include "adc.h"
#include "button.h"
#include "stearing.h"
#include "ir_find.h"
#include "g3_globl.h"


#define BUMPER_DELAY 1000

#define VERBOSE 1        //Set this to zero to remove error msgs.

//global variables for speed measurements
signed char Speed1, Speed2;

//global variables for speed control
bank2 signed char Speed1_wanted=0, Speed2_wanted=0;
bank2 signed int speed_control_ipart1=0, speed_control_ipart2=0;
bank2 signed int power1=0, power2=0;
bank2 signed char err1, err2;
bank2 unsigned char spdcont_clock=0;
bank2 int time;

void upd_motors(void)
{
     signed char err_diff;

     err1=Speed1_wanted-Speed1;
     err2=Speed2_wanted-Speed2;


     if ((Speed1_wanted>2)&&(Speed1_wanted>2))
     {
          //err_diff only when forward motion
          err_diff=err1-err2;
     }else{
          err_diff=0;
     }

     power1=Speed1_wanted*7+err1*8+speed_control_ipart1+err_diff*2;
     if (power1>127) power1=127;
     if (power1<-128) power1=-128;

     power2=Speed2_wanted*7+err2*8+speed_control_ipar-2*err_diff*2;
     if (power2>127) power2=127;
     if (power2<-128) power2=-128;

     motors(power1, power2);
     return;
}
```

```
void turn(char object)
{
     //Tabellerna visar hastigheten i respektive fall foer angles -4 till 5, -4
faar index 0.
     const char turn_speed_find_puck[10]={-13, -13, -10, -8, 0, 8, 10, 13, 13, 13};

     const char turn_speed_find_goal[10]={-8, -8,  -6, -6, 0, -8, -8, -8, -8, -8};

     if ((object==PUCK)||(object==DEF_GOAL))
          set_speed(turn_speed_find_puck[angles[object]+4], -
turn_speed_find_puck[angles[object]+4]);
     if (object==OFF_GOAL)
          turn_left(-turn_speed_find_goal[angles[object]+4]);

     /*str_print1("Turn ang=", 0);
     int_print_at_cursor(angles[object]);
     str_print_at_cursor("   ");*/
     return;
}


void follow_object(char object)
{

     char my_speed=0;
     if (object==PUCK) my_speed=10;
     if (object==DEF_GOAL) my_speed=10;
     if (object==OFF_GOAL) my_speed=12;


     if (angles[object] == 0)
     {
          set_speed(my_speed, my_speed);
          if (object==OFF_GOAL) set_speed(my_speed, my_speed);
     }

     if(angles[object] < 0)
     {
          set_speed(my_speed+(angles[object]*5),my_speed);
          if (object==OFF_GOAL) set_speed(4, 10);
     }

     if(angles[object] > 0)
     {
          set_speed(my_speed, my_speed-(angles[object]*5));
          if (object==OFF_GOAL) set_speed(10, 4);
     }
     upd_motors();
     return;
}
```

```
void turn_pos(signed char pos1_diff, signed char pos2_diff, signed char speed)
{
     unsigned char enc1_org, enc2_org;
     signed char my_speed1, my_speed2;
     int i;
     enc1_org = enc1_pos;
     enc2_org = enc2_pos;

     for (i=0;(i<3000)&&((my_speed1!=0)||(my_speed2!=0)); i++){
          my_speed1=0;
          my_speed2=0;
          if ((pos1_diff!=0) && (((signed char)(enc1_pos-enc1_org)) > pos1_diff))
my_speed1=speed;
          if ((pos2_diff!=0) && (((signed char)(enc2_pos-enc2_org)) > pos2_diff))
my_speed2=speed;


          if(CLUB_IN_ZONE||BACK_IN_ZONE||FRONT_IN_ZONE)
               break;

          set_speed(my_speed1, my_speed2);
          upd_motors();
     }

     set_speed(0,0);
     upd_motors();
}

void bumper() {


     if((Speed1_wanted > 0) || (Speed2_wanted > 0)) {
          if((Speed1!=0) && (Speed2!=0)) {
               time = soft_time();
          } else if(soft_time() > (time+BUMPER_DELAY)) {
               //Haer bryter vi oss loss
               if(HAS_PUCK)
               {
                    lcd_print2("Fastnat P",0);
                    turn_pos(-15, -15, -5);
                    set_speed(-5, 6);
                    for(sometime=soft_time();!passed(sometime+800);)
                    {
                         upd_motors();
                    }
               }else{
                    lcd_print2("Fastnat  ",0);
                    turn_pos(-20, -60,-14);
               }
               time = soft_time();
          }
     }
     else if((Speed1_wanted < 0) || (Speed2_wanted < 0)){
          if((Speed1!=0) && (Speed2!=0)) {
               time = soft_time();
          } else if(soft_time() > (time+BUMPER_DELAY)) {
               lcd_print2("Fastnat  ",0);
               set_speed(5, 5);

          }
     }
}
```

stearing.irh

```
#define spdcont_intervall 10
#define speedm_intervall 10

//Last update: 2003-05-08 /Alexander


//Local variables for the interupt-routine, include this in the top of the
interupt-routine.
unsigned static char pos_store1;
unsigned static char pos_store2;
bank1 signed static char Speed1_store[3];
bank2 signed static char Speed2_store[3];


unsigned static char speedm_clock;
```

stearing.irc

```
//include this in the interrupt-routine inside the right interrupt-block (timer1-
//flag)
//Last update: 2003-05-08 /Alexander


//Code to calculate speed
if (speedm_clock==speedm_intervall)
{

     speedm_clock=0;
     Speed1=(enc1_pos-pos_store1)+Speed1_store[0]+Speed1_store[1]+Speed1_store[2];
     Speed2=(enc2_pos-pos_store2)+Speed2_store[0]+Speed2_store[1]+Speed2_store[2];

     Speed1_store[2]=Speed1_store[1];
     Speed1_store[1]=Speed1_store[0];
     Speed1_store[0]=(enc1_pos-pos_store1);

     Speed2_store[2]=Speed2_store[1];
     Speed2_store[1]=Speed2_store[0];
     Speed2_store[0]=(enc2_pos-pos_store2);

     pos_store1=enc1_pos;
     pos_store2=enc2_pos;
}else{
     speedm_clock++;
}


//This code updates the integrated part of the power-levell for each motor, the
//power can not be set here since
//function-calls should be avoided in interupt-routine: use upd_motors()!
//spdcont_clock++;

if (spdcont_clock==spdcont_intervall)
{
       //integration should not be done every interupt, we use spdcont_clock to slow
       //integration.

     err1=Speed1_wanted-Speed1;
     err2=Speed2_wanted-Speed2;

     speed_control_ipart1=speed_control_ipart1+(err1);
     speed_control_ipart2=speed_control_ipart2+(err2);

     if (speed_control_ipart1>127) speed_control_ipart1=127;
     if (speed_control_ipart1<-128) speed_control_ipart1=-128;
     if (speed_control_ipart2>127) speed_control_ipart2=127;
     if (speed_control_ipart2<-128) speed_control_ipart2=-128;
     spdcont_clock==0;
}
```