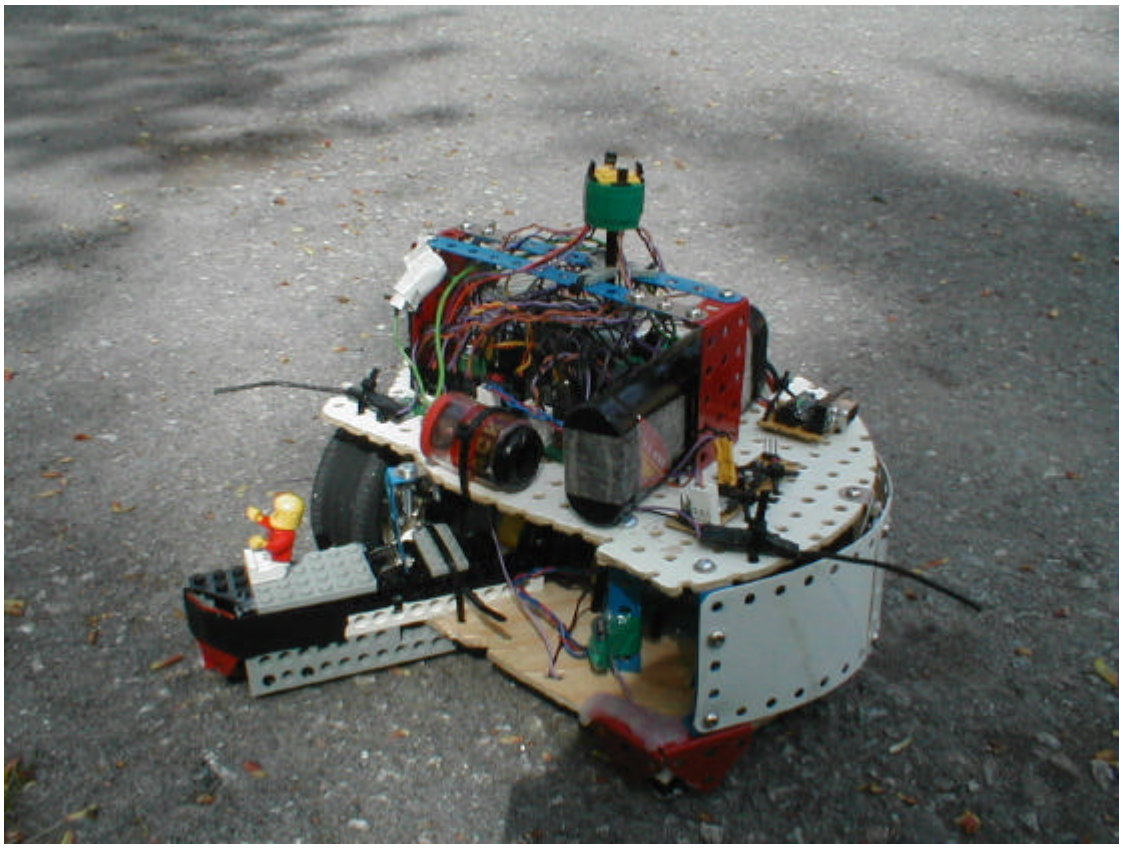
The Scrapheap

An Autonomous Robot

Hugo Cornelius (hugoc@kth.se)
Mattias Johansson (mattej@kth.se)
Mathias Vesterlund (mathiasv@kth.se)
Måns Ullerstam (mans@ullerstam.se)

May 26, 2003



Abstract

This paper is a project report describing the design and construction of the fully autonomous hockey-playing robot The Scrapheap. The Scrapheap was constructed as a major part of the course “Robotics and other Autonomous Systems” held at KTH, Stockholm, Sweden, where the students in groups were required to build a robot. The course ended with a tournament between the different robots. The Scrapheap finished an impressive second in this tournament.

This paper contains all necessary information regarding the main design ideas and software implementation.

Table of Contents

1 Introduction 4
2 Hockey Rules 4
3 Design Process 5
4 Hardware Design 5
4.1 Basic Design 5
4.2 The Stick 6
4.3 Weight Distribution 6
4.4 Motors and Wheels 6
5 Sensing 7
5.1 Floor Sensors 7
5.2 Target Sensors 7
5.3 Puck Sensors 8
5.4 Motor sensors 8
5.5 Antennae 8
6 Locomotion and Navigation 9
6.1 Navigation 9
6.2 Speed 9
6.3 Stuck 9
7 Manipulation 9
7.1 Scoring 10
8 Communication 11
8.1 Debugging 11
8.2 Goal ..... **Error! Bookmark not defined.**
9 Programming Environment 11
10 Electronics 11
10.1 Components 11
10.2 Connections 12
11 Software Design 13
11.1 Goals 13
11.1.1 To score goals fast 13
11.1.2 Low penalty rate 13
11.1.3 Getting unstuck 13
11.1.4 Reliability 13
11.2 Design Choices 13
11.3 Flowchart 13
11.4 Sense 14
11.5 Act 14
12 Results 15
12.1 Qualification Round 15
12.2 Competition 15
12.3 Other 15
13 Conclusions 15
14 References 16
15 Appendix 16
15.1 main.c 17
15.2 main.h 29
15.3 states.c 30
15.4 states.h 36
15.5 globals.h 37

1 Introduction

In the course “Robotics and other Autonomous Systems” held at KTH, Stockholm, Sweden, the students were assigned a project of designing and building a fully autonomous robot. The students were divided into groups of 3-4 students, each group building one robot. Altogether six robots were built. The project robot was meant to play a robot hockey game and at the end of the course a tournament was played. The students were given some essential hardware like a motherboard with a PIC micro controller, two 12 V motors and wheels. In addition, the students had access to a lab with some basic tools and building blocks, such as LEGO pieces, Meccano pieces and soldering iron. The groups had total freedom in constructing their robots as long as they were not violating the rules.

2 Hockey Rules

The rules for the robot competition and the constraints on the robot itself were [3]:

- Matches are three minutes long except for semi-finals (4min) and final (5min). There is only one robot on each team.
- The matches have no breaks whatsoever.
- When the game starts both robots are in their defensive goal zones.
- A goal is scored whenever the puck is partly within the red zone at the goal aluminium plate; the goal zone rule has not been violated (see below). The scoring robot also has to signal the goal by playing its own pre-defined tune on its speaker. After each scored goal the puck will be moved to the centre point.
- A robot may only signal a goal directly after it has tried to score. Other signalling will render the robot 10 seconds off the rink.
- A robot is not allowed to grab the puck, and the edge of the vertical projection of the robot body (parts of the robot body that are high enough above the floor to let the puck pass under them do not count) on the floor may not have any concave parts (except for concavities formed between the driving wheels and the rest of the robot, provided they can not be used to guide the puck). Any protruding whiskers or bumpers must be flexible enough so that they cannot be used to move or guide the puck. The robot must be constructed so that the puck cannot be on top of any part of the robot.
- Each robot may have a stick to guide the puck. It may extend no more than one puck diameter from the robot body. The part of the stick outside the robot body must not be wider than one puck radius, and may not have any joints or concavities.
- When a robot goes past the white line an inch into its own defensive goal zone with any non-flexible part (excluding the stick), it will be taken off the rink for 10 seconds and then put back far from the puck. The goal zones are marked by black paint on the floor, as is the centre line.
- When a robot scores, it may not go past the white line in the goal zone with any non-flexible part (excluding the stick). If it does the scored goal is not counted. If the puck is left in the goal zone without scoring it will be moved to the centre point 3 s after a robot last touched it.

- A robot that is pushed into a goal zone and is constantly working to get out of it will receive no penalty (i.e. the any goal is approved or the robot is not taken of the rink).
- If a robot brakes down during a match, it may be taken off the floor for repair for a minimum of 30 s and then put back far from the puck.
- A robot must fit in a circle with diameter 250 mm (as measured on the vertical projection on the floor). This does not apply to flexible parts or the stick.
- When guiding the puck, the stick (or any whiskers/bumpers) must not occlude the IR LEDs of the puck from any direction.
- No device that can harm the opposing robot, on purpose or not, may be mounted on a robot. Examples include knives, electric guns, water guns, and electromagnetic interference devices, but also a stick that is narrow and powerful enough to cause serious damage, should the opponent come in its way when it shoots out in an attempt to score a goal.

3 Design Process

Our basic approach to this project was trial and error. Since there were too many unknown factors, both hardware and software, a trial and error approach seemed to be the most reasonable. When designing the robot, we only thought about its functionality, and not about its appearance, that led to a not so handsome, but quite well working, robot.

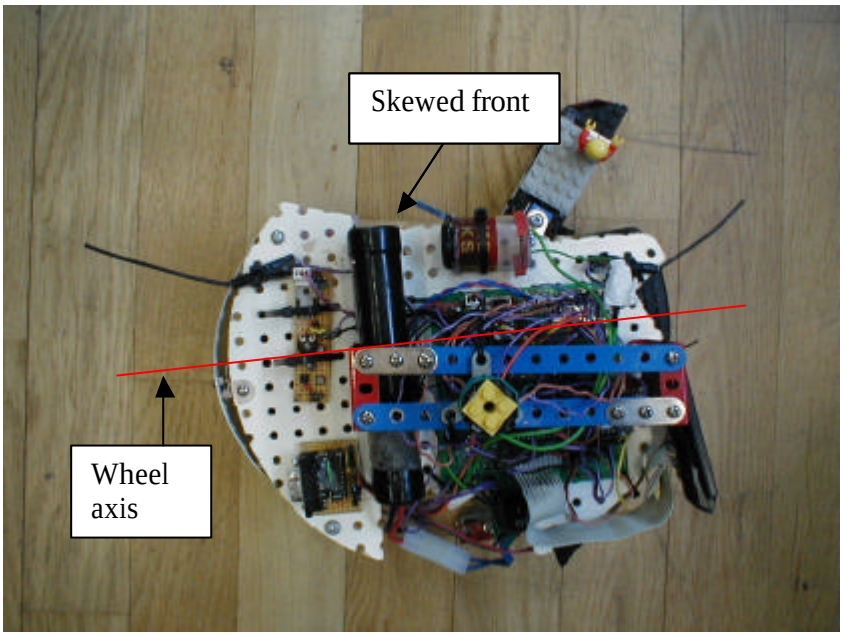
Since the software of the robot was dependent on the hardware and vice versa, we built the robot and designed the software in parallel. We tried to design the robot in such a way that it would be easy to change and rebuild parts of it if something would turn out not to work as we had hoped. For example, we put motherboard and the auxiliary boards used on top of the robot so that they would be easy to access.

When building the robot, we started with the most basic things like motors, wheels, target sensors and an algorithm to go towards a target. When these parts were working properly, we added the stick, more sensors and more complicated behavior one thing at the time. We always tried to get all the parts on the robot to work well before adding another part to it. This was to make it easier to find bugs in the code and also to locate problems in the hardware. Despite our efforts to make troubleshooting easy, we still had lots of problems finding out what was wrong when the robot behaved strangely.

4 Hardware Design

4.1 Basic Design

The basic hardware design took much of its inspiration from JAJ, last year's runner up [1]. The robot was fairly circular. Quite early we realized, from looking at the video from last year's competition, that if the front of the robot was straight or convex the puck had a tendency to slide away from the robot when driving towards the goal. Our solution to this problem was to have a slightly skewed front, see picture 1. This had the effect that whenever the robot drove forward into the puck, it didn't matter where on the front of the robot the impact took place, since it always slid into the wedge between the stick and the skewed front.



Picture 1 - Aerial view of the Scrapheap

4.2 The Stick

The stick was pointing out close to 45 degrees from the skewed front edge to the front right. The width of the stick was half the radius of the puck (which was the maximum allowed [3]). The stick's front was fairly flat and parallel to the wheel axis. This made it possible to guide the puck on the front of the stick while scoring.

4.3 Weight Distribution

The total weight of the robot proved to be quite important due to the weakness of the batteries and motors. The weight distribution between front and rear was also important to lower the friction between the robot and the table. The main source of movable weight was the batteries, which were used to distribute the weight so that the center of gravity was on the wheel axis. The Scrapheap did not use a third wheel for support, due to the fact that the steering became imprecise. Instead two support legs were used, one in the front and one in the rear. Only one of the legs was in contact with the ground at any time. In the beginning we used LEGO pieces as supporting legs, but later taped the soft side of Velcro tape on them resulting in much less friction between the robot and the floor.

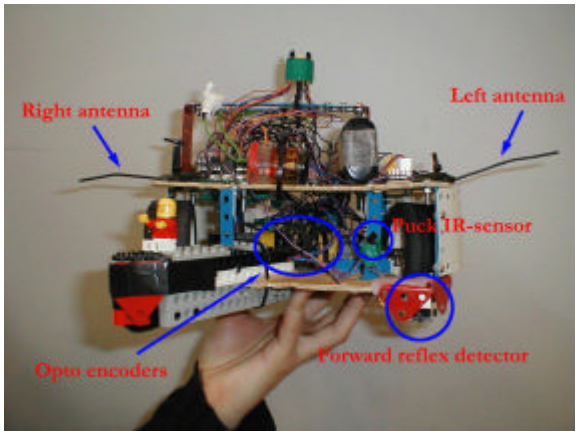
4.4 Motors and Wheels

The wheel spokes were connected to the motor axis via a Meccano piece. The piece was glued to the wheel and screwed to the axis. It was very important that the wheels were leveled in jaw and tilt. Otherwise too much energy was lost in the jitter of the wheels. The fixing and the positioning of the motors to the bottom plate were quite important in the same respect. The slightest skew or loose parts in the locomotion machinery reduced the power to the wheels substantially.

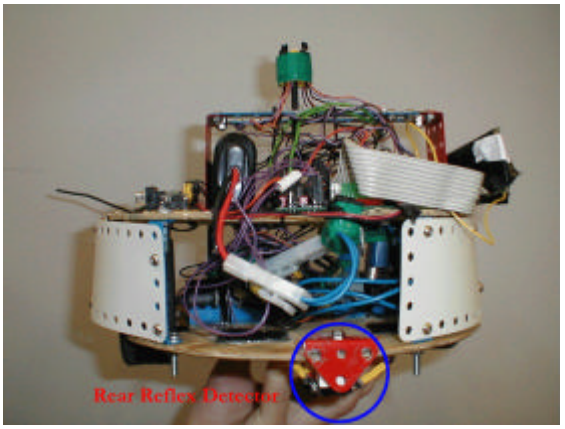
5 Sensing

5.1 Floor Sensors

The robot had two floor sensors, one in the front left corner and one in the center in the rear, see pictures 2 and 3. The main purpose of the floor sensors was to detect when the robot had entered a goal zone or crossed the centerline of the table. The sensors consisted of two optical reflex detectors (see section 10.1) whose output was sampled by the PIC giving a number between 0 and 255 [4]. The highest value represented the darkest input signal. If the sensors were too far from the floor it was interpreted as very dark. This led to high demands on the overall construction of the robot. The sensors had to be very close to the floor at all times, no matter what the robot was doing. In our construction with only two wheels we had to prevent the robot from tipping too much in sudden breaks or accelerations.



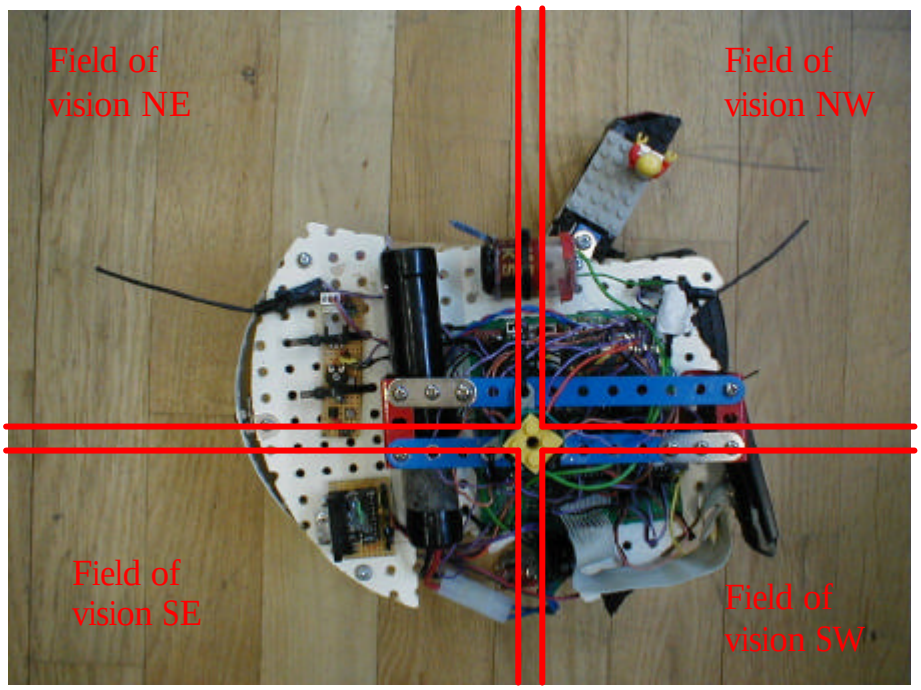
Picture 2 - The different sensor locations



Picture 3 - The rear reflex detectors

5.2 Target Sensors

Since the puck, offensive and defensive goal were all emitting IR light with different modulation, IR sensors (see section 10.1) were used as target sensors. The target sensors were located in the center of the robot on a collar. The directions of the target sensors were NE, NW, SE and SW. This meant that there were two sensors at the front, sensing left and right, as well as two sensors at the back, also sensing left and right well. These sensors were used to detect any target emitting IR-signals. In our case the targets were the opponent goal, the puck or the defensive goal. In navigation we used the fact that we had four different IR sensors, and through choosing the two sensors with the strongest signals we got a direction to the target. For a deeper discussion of the steering algorithm, see section 6.1. We also used the target sensors to decide when the robot was close to one of the goals. Either the sum of all four signals coming from the goal was higher than 220 or the two front sensors both showed a value higher than 17 for the defensive goal or 20 for the offensive goal, and we considered the robot close to respective goal.



Picture 4 - The locations of the IR sensors

5.3 Puck Sensors

To enhance the control of the puck, an extra IR sensor was located in the front of the robot facing the area where the robot held the puck, i.e. the corner between the stick and the skewed front , see picture 2. If the signal from the puck sensor was above 800 the robot was considered close to the puck, and if the signal was higher than 4200 the robot considered itself in possession of the puck.

5.4 Motor sensors

On each motor there were two sensors measuring the rotation of the wheels. The sensors are placed on the inner part of the motor axes, see picture 2. The sensors used were slotted opto detectors (see section 10.1), which have LED's on one side of the slot and light detectors on the other side [3]. A disc with black and transparent fields connected to the motor axis was placed in the slot. When the wheel turned, the disc rotated and the opto detector registered if there was a black or transparent field in the slot. With two sensors on each wheel it was possible to sense the speed and direction of the robot. Our robot used the sensors to recognize if it were stuck and when scoring goals.

5.5 Antennae

Our robot used two antennae, which were sticking out to the front left and right, see picture 2. One antenna consisted of a metal pipe and a metal wire sticking out of the pipe. The wire was connected to 5 V, via a resistor, and the pipe was connected to ground. If the wire was pushed in any direction, it would touch the pipe and a current could be registered. The reason for having an antenna was to see if the robot had hit something on the left or right side when it was stuck.

6 Locomotion and Navigation

6.1 Navigation

The main navigation was based on a routine, which tried to find a specific target. It had two different states, one where the target was behind the robot and one where it was in front.

- If the target was behind the robot, it turned on the spot towards the target by going forward with one wheel and backwards with the other.
- If the target was somewhere ahead of the robot, it moved towards the target and made small left and/or right turns depending on what side the target was located. If the target was in front of the robot to the right (left) a higher speed was applied to the left (right) wheel than to the right (left).

This was the main steering algorithm, but if the robot carried the puck, it never made any sharp left turns. This was due to the fact that our stick was angled towards the right, and thus handled left turns better.

6.2 Speed

The robot did not have any complex speed algorithms. It used the command "motors (speed left, speed right)" for all speed control. This command regulated the duty cycle of the PWM's to the motors. This turned out to be a problem, since the actual ground speed of the robot could differ greatly over time depending on any of the variables below:

- Traction, the traction changes because of table surface and tire surface
- Battery power, when the battery power is high the robot moves quicker
- Other hardware components, e.g. the voltage converter
- Moving with or without puck, different weight and heavy on the right side
- Weight of the robot, this changed during the course of the project

6.3 Stuck

If any of the motor sensors did not register any change during a certain period of time, the robot was considered stuck. If the battery was low and the robot tried to drive at low speed it sometimes stopped without really being stuck. To see if this was the case, the robot first tried to go forward at maximum speed for 50 ms. If the robot was still stuck after that, it tried to get unstuck. To decide on which side the robot was stuck, the antennae on the left and right side was used. If the antenna on the left side was pushed, the robot was considered stuck on the left side. If the robot was stuck on the left side without the puck, it tried to get unstuck by first going a little bit backwards and then turning to the right. If it was not stuck on the left side, it instead of turning to the right, tried to turn to the left.

7 Manipulation

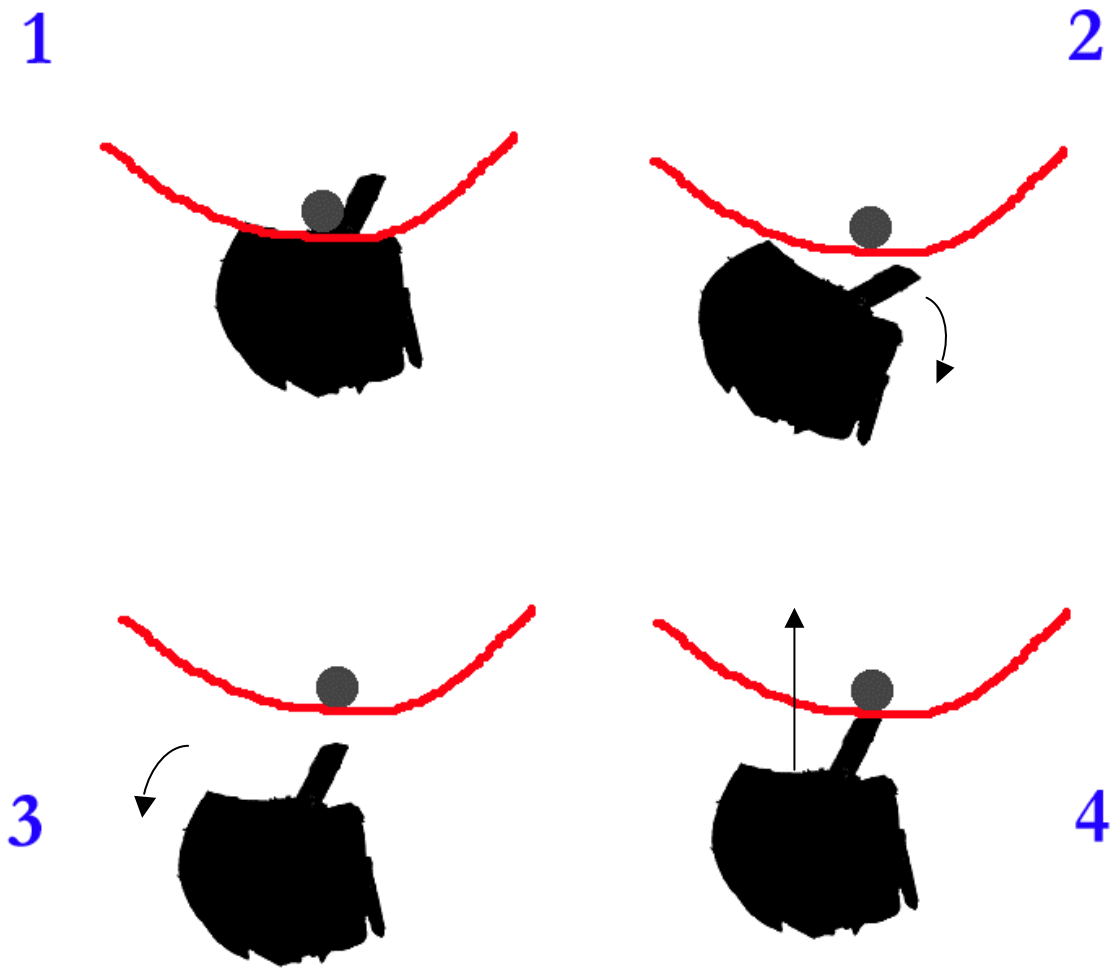
7.1 Extracting the puck

If the robot got stuck and was in possession of the puck it tried to get unstuck without loosing the puck. It always did this by turning to the left after reversing for a short period of time. The reason for this was that the robot had its stick on the right side, and it would run the risk of losing the puck if it would turn to the right.

7.2 Scoring

When the floor was black under the front reflex detector, the IR sensors showed that the robot was close to the offensive goal and it had the puck, the robot tried to score a goal. To be able to stop in front of the goal without going too far into the goal zone, the robot had to slow down when it got close to the offensive goal (see section 5.2) with the puck. This was done by braking and then driving at a slower speed when we got close to the offensive goal with the puck. The braking was only done once when getting close to the offensive goal.

The scoring routine took much of its inspiration from Nätagget, last year's winner [2]. The idea was to push the puck into the goal with the end of the stick (see picture 5). When reaching the offensive goal, the robot first braked by going full speed backwards for a few milliseconds. The robot used the optical encoders on the wheels to go backwards with high accuracy, first with the right wheel and then with the left wheel, until we had the puck in front of the stick. It then drove straight forward with high speed pushing the puck into the goal with the end of the stick.



Picture 5 – The goal routine

8 Communication

Basically the two different communication tools the robot had to the outside world were a speaker and a LCD display [3]. The communication tools were used for debugging and testing purposes and also for telling the world that it had scored a goal. It was possible to use built-in LEDs on the motherboard, but there was no need to use them. The LCD had 2 lines of text with 16 characters each.

8.1 Debugging

The LCD display was used to display sensor values or to display the state the robot was in. The speaker was used to tell when the robot entered a specific state, for example stuck or close to offensive goal. This is a very helpful tool for the trial and error approach.

8.2 Signaling

To get a score counted the robot had to play a melody right after it had scored. Our robot played “Für Elise” by Beethoven.

9 Programming Environment

The MPLAB [5], which is an Integrated Development Environment, was used to manage and compile the robot project source files. MPLAB runs under Windows and has many capabilities such as compilation into hex-code, project management, simulation/debugging with breakpoints and more.

MPLAB is fully integrated with the Hi-Tech PIC C [6] compiler and debug C source, allowing it to compile and link C code (without support for function recursion) into a hex-files. The hex-file is then programmed into the PIC using the PIC bootloader from Hi-Tech. The last 256 bytes of the PIC program memory hosts the bootloader, which is a set of instructions that takes care of the serial communication with the PC while the code is being transferred.

10 Electronics

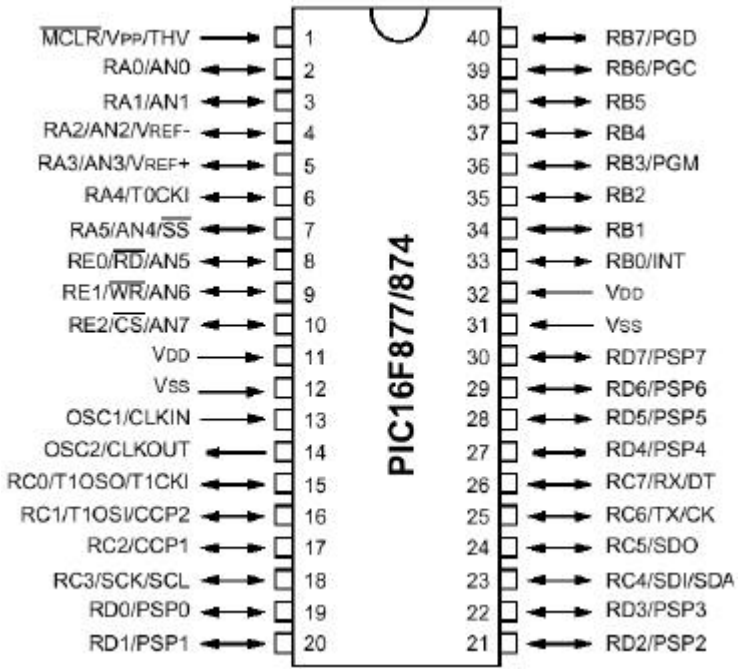
10.1 Components

- 1 PIC (main) PIC16F877 40-pin 8-Bit CMOS FLASH Micro controller.
  Manufacturer: Microchip Technology Inc.
  Part no: PIC16F877
- 1 PIC (for IC) PIC16F876 20-pin 8-Bit CMOS FLASH Micro controller
  Manufacturer: Microchip Technology Inc.
  Part no: PIC16F876
- 1 Motherboard
- 1Boostercard-7.2Vto14.4V
- 1 LED Driver board
- 1 Serial communication board
- 1 LCD Powertip PC1602D A
- 2 12 V motors HL149 12 V 10:1 Motor
- 1 100 O speaker

- 5 IR sensors TSL261 IR Light-to-Voltage Optical Sensor
  Manufacturer: Texas Advanced Optoelectronic Solutions (TAOS)
  Part no: TSL261
- 2 reflex detectors ITR8307 Subminiature High Sensitivity Photo Interrupter
  Manufacturer: Everlight
  Part no: ELITR 8307
- 4 slotted opto detectors ITR 8010 General Purpose Photo Interrupter
  Manufacturer: Everlight
  Part no. ELITR 8010
- 2 antennae
- 4 IR diodes SFH485P GaAlAs Infrared Emitter
  Manufacturer: Siemens
  Part no: SFH485P
- 2 7.2 V NiCd accumulator packs
- 1 signal level converter. RS-232 with DB9 socket

10.2 Connections

The connections to the PIC processor are described below:



Picture 6 – The pins on the PIC processor

RA4, RE0	right opto detectors
RE1, RE2 RA0	left opto detectors front reflex detector
RA1	rear reflex detector
RC2	left motor control pin (enable)
RB0	left motor control pin (direction)
RC1	right motor control pin (enable)
RB1	right motor control pin (direction)
RB4	left antenna (internal pull-up resistors are used)
RB5	right antenna (internal pull-up resistors are used)
RC0	bootloader switch
RD0, RD1, RD2, RD3, RD4, RD5, RD6, RD7	LCD
V_{SS}, V_{DD} RB2	power to RS-232 sound enable
RC6, RC7	communication with RS-232
RB3	sound

11 Software Design

11.1 Goals

When writing the software we had a few specific objectives in mind. Some of the more important ones are described below.

11.1.1 To score goals fast

Since the tournament were based on a qualification round where the robot had to score as many goals as possible in two minutes, it was important that it could score many goals in short time without an opponent. In order to do this, the robot had to get back to the centerline and face the offensive goal quickly after scoring, since the puck was placed in the middle of the rink after every goal.

11.1.2 Low penalty rate

The only way to be forced off the rink during a game was if the robot entered the defensive goal zone. Since most goals were scored when one robot was outside the rink, our robot was designed to be very careful close to the defensive goal.

11.1.3 Getting unstuck

After watching earlier games between robots [7], it was considered to be a major problem that the robot got stuck. Either the robots were stuck together or the robots couldn't get out from the sideboard. Therefore emphasis was put on getting our robot unstuck as fast as possible. There was a need to separate getting unstuck with the puck and without the puck, since there was a possibility of loosing the puck while getting unstuck.

11.1.4 Reliability

If a robot needed repair during a game, the robot had to be taken off the rink for a minimum of 30 seconds. It was therefore very important to build a reliable robot, which would very seldom need a helping hand. For example, when using the optical encoders we also had a time limit so that we wouldn't get stuck next to a sideboard.

11.2 Design Choices

The software design was a sense-act machine. This software design is borrowed from simple insects, which are incapable of planning ahead in time. The robot had an infinite loop doing three things:

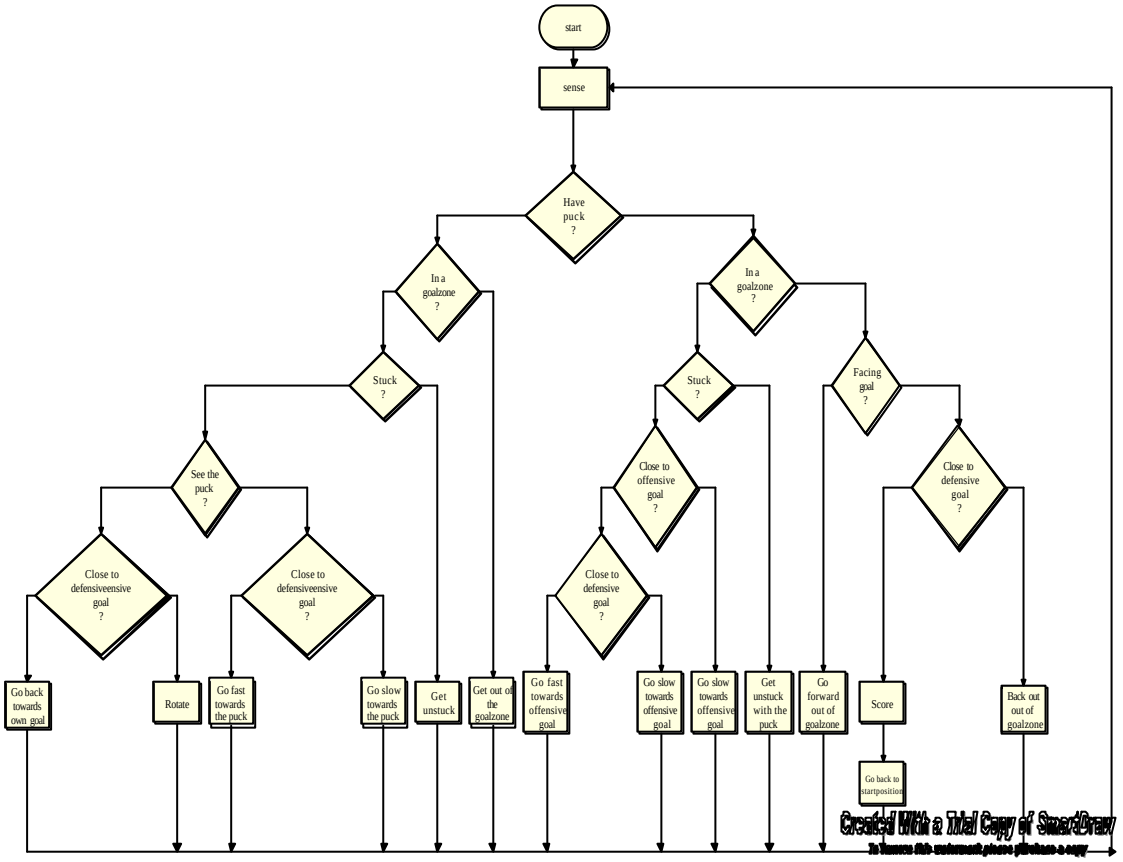
- Reading all sensor values
- Deciding what behavior to choose
- Perform the action related to the behavior

The software was therefore controlled by a flowchart with the behaviors described in the next section.

11.3 Flowchart

The heart of the program was the flowchart. During the design and testing, the flowchart started out being very small. New states were then added when there was a need to expand

the different behaviors in different situations. This was the flowchart of the robot's state decision process and a diagram describing all the different states:



Picture 7 – Software flowchart

11.4 Sense

Reading the sensor values was handled in a function called `sense()`. This function set a large number of global flags depending on the sensor values, for example:

- `close_off` – the robot was close to the offensive goal
- `has_puck` – the robot had control of the puck
- `in_goal` – the robot had entered a black area close to a goal
- `left_stuck` – the left antenna is being pushed, so the robot is touching either the side of the table or the other robot with the left antenna

11.5 Act

There are many specific behaviors depending basically on different sensor values. Some examples of behaviors are:

- `Score_goal()` – the scoring procedure
- `Get_unstuck()` – handles if the robots wheels aren't moving

- `Hunt_puck()` – if the robot does not have the puck, the most important task is to get control of it again

The most interesting behaviors are described in more detail above in sections 6 and 7.

12 Results

12.1 Qualification Round

During the qualification round the robots were allowed to score goals during two minutes without competition. The best robot, FejkOpaque, scored 23 goals. The Scrapheap was the runner-up and was able to score 7 goals. The other robots scored between one and five goals. This meant that The Scrapheap did not have to meet the best robot during the group play.

12.2 Competition

In the competition The Scrapheap was still undefeated before the final versus FejkOpaque. The final was tied at the end of regular time, but unfortunately the robot lost by a golden goal in overtime. The final match ended 2-3.

12.3 Other

The Scrapheap performed well during all games. The robot had to be taken off the rink for rebooting only once, when it got stuck in an undefined loop. There were a few occasions where The Scrapheap lost sight of the puck when it was behind an opponent. All those times The Scrapheap started to go back to its defensive position close to the defensive goal, until it saw the puck again. Overall it performed well against other robots, which was probably due to the fact that many of the situations that occurred during the games had been tested during development.

13 Conclusions

In the development of The Scrapheap, we wanted to take small steps towards a complete robot and try all the subparts out before combining them to a whole. This resulted in a trial and error methodology and a lot of time spent on the rink. The benefit of this was that when it came to building the actual robot, we knew that all the subparts worked and we had relatively few hardware problems. This was also applicable to the software. We also found out that a logging of the code and changes made on it was crucial to avoiding serious problems. The logging became an important part of the development process.

When it comes to the physical part, we wanted a simple design. We do think that we succeeded in our goal, but there are a few things to note. First of all, the skewed front of the robot was designed as a puck-collecting device, and as such it worked well. For the same reason, we also had an angled stick, and this proved not to be the best solution. The angled stick made it difficult to make sharp turns when the robot had the puck, and the ability to turn quickly is crucial in robot hockey. A better physical design would probably have been to have an angled front and a straight stick. Secondly, we built the robot on an old piece of plywood left over from last year's robots. Late in the development process, we noticed that the plywood was relatively soft and changed shape when weights were put on it. This resulted in the fact that some parts came in heavy contact with the ground, which slowed

down our robot and forced us to rethink the placement of some parts, especially the battery. Lessons to be learned from this, is that you should always use fresh wood and that the weight distribution of the robot is important. Thirdly, if we had placed a floor sensor on the front of the stick, many problems we had with the goal zone and the speed close to it could have been avoided. The scoring routine could also have been improved with such a design. Last but not least, on a brighter side, our design was a flexible one and we could fix all those small hardware problems easily.

On the software side, we are fully satisfied with the way the final robot worked. The sense-act design of the brain of the robot worked well. Our routine when we did not see the puck enabled us to defend our own goal in an almost perfect fashion. The stuck routine proved to be very successful. The only thing we feel could have meant an improvement to the robot is a better go_to-routine. Our routine was a little slow, and a few times during the tournament other robots proved to reach the puck much faster. Another smaller improvement could be a PI-regulator controlling the speed of the robot.

All in all, with respect to time and design constraints, we were very pleased with The Scrapheap.

14 References

[1] JAJ - An Autonomous Silver Medalist, John Hallmén, Andreas Hedström, Jonatan Fernstad, June 2002, http://www.nada.kth.se/kurser/kth/2D1426/reports2002/JAJ.pdf

[2] Nätagget - An Autonomous Robot, Per Alexius, Hampus Edvardsson, Linus Ericson, June 2002, http://www.nada.kth.se/kurser/kth/2D1426/reports2002/Natagget.pdf

[3] Course notes V: The Project, http://www.nada.kth.se/kurser/kth/2D1426/L5.pdf

[4] Datasheet for 40-pin 8-Bit CMOS FLASH Micro Controller PIC 16F877. Microchip. http://www.microchip.com/download/lit/pline/picmicro/families/16f87x/30292c.pdf

[5] MPLAB Manual. MPLAB IDE, "Simulator, Editor –User's Guide" Microchip

[6] Hi-Tech PIC C User's Guide. Hi-Tech

[7] Course homepage: Movie from last years competition, http://www.nada.kth.se/kurser/kth/2D1426/RAScompetition2002.avi

15 Appendix

15.1 main.c

```
#include <pic.h>
#include <conio.h>
#include <stdlib.h>
#include <string.h>
#include "defines.h"
#include "lcd.h"
#include "adc.h"
#include "sprint.h"
#include "ir_comm.h"
#include "pwm.h"
#include "intr.h"
#include "serialio.h"
#include "globals.h"
#include "states.h"
#include "main.h"

// Define globals
bank1 int SENSOR_1;
bank1 int SENSOR_2;
bank2 int irval[4][3];
bank1 unsigned int sometime;
bank1 unsigned int stuck_checkpoint;
bank2 int adval[2];
bank1 int irval_stick;
bank1 char enc1,enc2;
bank3 bit temp_enc1,temp_enc2;
bank3 bit verbose;
bit have_puck;
bit in_goal;
bit facing_goal;
bit close_def;
bit close_off;
bit see_puck;
bit off_half;
bit stuck;
bit close_puck;
bit left_stuck;
bit right_stuck;
bit scoring_flag;
char buffer[16];

const char fur_elise[] =
{0x24,20,0x23,20,0x24,20,0x23,20,0x24,20,0x1B,20,0x22,20,0x20,20,0x19,40
,0xFF,20,0x10,20,0x14,20,0x19,20,0x1B,40,0xFF,20,0x14,20,0x18,20,0x1B,20
,0x20,40,0xFF,20,0x14,20,0x24,20,0x23,20,0x24,20,0x23,20,0x24,20,0x1B,20
,0x22,20,0x20,20,0x19,40,0xFF,20,0x10,20,0x14,20,0x19,20,0x1B,40,0xFF,20
,0x12,20,0x20,20,0x1B,20,0x19,80,0,0};

const char blues_notes[] =
{0x3C,20,0x3A,20,0x37,20,0x36,20,0x35,20,0x33,20,0x30,20,0x2A,20,0x27,20
,0x26,20,0x25,20,0x23,20,0x20,20,0x1A,20,0x17,20,0x16,20,0x15,20,0x13,20
,0x10,20,0x0A,20,0x07,20,0x06,20,0x05,20,0x03,20,0x00,80,0,0};
```

```
const char laser[] =
{0x00,2,0x02,2,0x04,2,0x06,2,0x08,2,0x0A,2,0x10,2,0x12,2,0x14,2,0x16,2,0
x18,2,0x1A,2,0x20,2,0x22,2,0x24,2,0x26,2,0x28,2,0x2A,2,0x30,2,0x32,2,0x3
4,2,0x36,2,0,0};

const char klingon[] = {0x10,30, 0x13,20, 0xF0,10, 0x17,40,0x10,20,
0x13,20, 0x17,20,0x10,30, 0x13,20, 0xF0,10, 0x17,40,0x10,20, 0x13,20,
0x17,20,0x15,30, 0x13,20, 0xF0,10, 0x10,40,0x10,20, 0x13,20,
0x10,20,0x17,30, 0xF0,10, 0x17,30, 0xF0,10, 0x15,30, 0xF0,10,
0x17,30,0,0};

// 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);

main()
{
      //Initiate everything we want to initiate
      init();

      // main loop
      for(;;)
      {
            //read all sensors & set appropriate flags
            sense();

            //we have the puck
            if(have_puck)
            {
                  //we are in a goal zone
                  if(in_goal)
                  {
                        //we are facing a goal
                        if(facing_goal)
                        {
                        //we are in defensive goal zone
                              if(close_def)
                              {
                                    get_out_of_goalzone();
                              }
                              //we are in offensive goal zone
                              else
                              {
                                    score();
                                    // go back to start
                                    go_to_start_pos();
                              }
                        }
                        //we are not facing a goal
                        else
                        {
                              get_out_of_goalzone();
                        }
                  }
                  //we are not in a goal zone
                  else
```

```
	{
		//we are stuck
		if(stuck)
		{
			get_unstuck_with_puck();
		}
		//we are close to offensive goal
		else if(close_off)
		{
			close_off_goal_with_puck();
		}
		//we are close to defensive goal
		else if(close_def)
		{
			close_def_goal_with_puck();
		}
		//we are somewhere in the middle of the rink
		else
		{
			goto_off_goal_with_puck();
		}
	}
}
//we haven't got the puck
else
{
	//we are in a goal zone
	if(in_goal)
	{
		get_out_of_goalzone();
	}
	//we are not in a goal zone
	else
	{
		//we are stuck
		if(stuck)
		{
			get_unstuck();
		}
		//we can see the puck
		else if(see_puck)
		{
			//we are close to defensive goal
			if(close_def)
				hunt_the_puck_close_def();
			//we are close to offensive goal
			else
				hunt_the_puck();
		}
		//we are not stuck & we can't see the puck
		else
		{
			//we are close to defensive goal
			if(close_def)
			{
				rotate();
			}
```

```
                              //we are close to offensive goal
                              else
                              {
                                    go_to_own_goal();
                              }
                        }
                  }
            }

      }
}

//init
void init()
{
      /*
      *set appropriate values for A/D channel
      */

      ADCON1=0b00000100;
      TRISA0=TRISA1=1;

      TRISB4=1;
      TRISB5=1;
      RBPU=0;

      ir_init();              //initialize IR co-processor communication
      init_pwm();             //initialize pwm for motor control
      init_uart();                        // initialize serial I/O
      init_soft_tmr();
      init_sound();                 //initialize sound generator
      init_enc();

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

      set_lcd_power(1);       //turn on LCD

      // wait for 100ms
      sometime=soft_time();
      for(;;) if(passed(sometime+100)) break;

      // and then
      init_lcd();

      //assume we start at own half
      off_half=0;

      //initialize stuck flag to 0 (not stuck)
      stuck=0;

      stuck_checkpoint=soft_time();
      enc1=enc1_pos;
      enc2=enc2_pos;

      /*Prints states if verbose=1
```

```
       *otherwise prints sensor values
       *on the LCD-display*/
      verbose=1;
}

//go_to() function
void go_to(int target,int speed)
{
      unsigned int i;
      unsigned int MAX_IR1;
      unsigned int MAX_IR2;

      MAX_IR1=irval[0][target];
      SENSOR_1=0;
      MAX_IR2=0;

      //get the strongest IR-values
      for(i=1; i<4; i++)
      {
            if(MAX_IR1 < irval[i][target])
            {
                  MAX_IR2=MAX_IR1;
                  SENSOR_2=SENSOR_1;
                  MAX_IR1=irval[i][target];
                  SENSOR_1=i;
            }
            else
            {
                  if(MAX_IR2 <= irval[i][target])
                  {
                        MAX_IR2=irval[i][target];
                        SENSOR_2=i;
                  }
            }
      }

      if(SENSOR_1==FL && MAX_IR2<15)
            SENSOR_2 = FR;

      if(SENSOR_1==FR && MAX_IR2<15)
            SENSOR_2 = FL;

      /*
      *if the two sensors in front are the strongest
      */
      if(((SENSOR_1==FL) && (SENSOR_2==FR)) || ((SENSOR_1==FR) &&
(SENSOR_2==FL)))
      {
      /*
      *right sensor is stronger than left sensor
      */
            if(irval[FR][target] > irval[FL][target])
            {
                  if (have_puck)
                        motors(speed,speed-40);
                  else
                        motors(speed,speed-35);
```

```
		}
	/*
	*left sensor is stronger than right sensor
	*/
		else if(irval[FR][target] < irval[FL][target])
		{
			if (have_puck)
				motors(speed-50,speed);
			else
				motors(speed-35,speed);
		}
		/*
		*both sensors are equally strong
		*/
		else
		{
			if (have_puck)
				motors(speed-4,speed);
			else
				motors(speed,speed);
		}
	}
	/*
	*one or both rear sensors is/are the strongest
	*/
	else
	{
		if(have_puck)
		{
			motors(0,MEDIUM_SPEED);
		}
		else
		{
			if(irval[BR][target]>irval[BL][target])
				motors(MEDIUM_SPEED,-MEDIUM_SPEED);
			else
				motors(-MEDIUM_SPEED,MEDIUM_SPEED);
		}
	}
}

/*
*delay function: returns after ms milliseconds
*/
void delay(int ms)
{
	int temp;
	temp=soft_time();
	while(!passed(temp+ms)){}
}

void sense()
{
	//read all ir-sensors
	ir_wait();
	irval[FL][DEF_GOAL]=ir_value(FL,DEF_GOAL);
	irval[FR][DEF_GOAL]=ir_value(FR,DEF_GOAL);
```

```
	irval[BL][DEF_GOAL]=ir_value(BL,DEF_GOAL);
	irval[BR][DEF_GOAL]=ir_value(BR,DEF_GOAL);

	irval[FL][OFF_GOAL]=ir_value(FL,OFF_GOAL);
	irval[FR][OFF_GOAL]=ir_value(FR,OFF_GOAL);
	irval[BL][OFF_GOAL]=ir_value(BL,OFF_GOAL);
	irval[BR][OFF_GOAL]=ir_value(BR,OFF_GOAL);

	irval[FL][PUCK]=ir_value(FL,PUCK);
	irval[FR][PUCK]=ir_value(FR,PUCK);
	irval[BL][PUCK]=ir_value(BL,PUCK);
	irval[BR][PUCK]=ir_value(BR,PUCK);

	irval_stick=ir_value(IR_PUCK,PUCK);


	/*
	*we are close to the puck
	*/
	if(irval_stick > CLOSE_TO_PUCK_THRESHOLD)
	{
		close_puck=1;
		/*
		*we have the puck
		*/
		if( irval_stick > HAVE_PUCK_THRESHOLD)
			have_puck=1;
		else
			have_puck=0;
	}
	/*
	*we are not close to the puck
	*/
	else
	{
		close_puck=0;
		have_puck=0;
	}


	/*
	*robot is on offensive side
	*/
	if(SUM_IR(DEF_GOAL)+50 < SUM_IR(OFF_GOAL))
		off_half=1;
	else
		off_half=0;


	/*
	*we are close to defensive goal
	*/
	if((SUM_IR(DEF_GOAL) > CLOSE_TO_GOAL_THRESHOLD ||
(irval[FL][DEF_GOAL] > 17 && irval[FR][DEF_GOAL] > 17) ||
		(irval[BL][DEF_GOAL] > 17 && irval[BR][DEF_GOAL] > 17))
		&& !off_half)
		close_def=1;
```

```
	else
		close_def=0;


	/*
	*we are close to offensive goal
	*/
	if(SUM_IR(OFF_GOAL) > CLOSE_TO_GOAL_THRESHOLD ||
(irval[FL][OFF_GOAL] > 20 && irval[FR][OFF_GOAL] > 20) ||
		(irval[BL][OFF_GOAL] > 20 && irval[BR][OFF_GOAL] > 20) ||
(have_puck && close_off))
	{
	/*
	*set scoring flag to make sure that we only slow down once
	*/
		if (!close_off)
		{
			scoring_flag=1;
		}
	/*
	*keep close_off set if already set
	*/
		if(off_half)
			close_off=1;
		else
			close_off=0;
	}
	else
		close_off=0;


	/*
	*robot is close to a goal
	*/
	if(close_def || close_off)
	{
		//sample reflexdetectors
		adval[FRONT]=sample_ad_channel(FRONT);
		adval[BACK]=sample_ad_channel(BACK);
		/*
		*robot is in a goalzone
		*/
		if(adval[FRONT] > IN_GOAL_THRESHOLD || adval[BACK] >
IN_GOAL_THRESHOLD)
		{
			in_goal=1;
		/*
		*robot is facing the goal
		*/
			if(adval[FRONT] > adval[BACK])
				facing_goal=1;
			else
				facing_goal=0;
		}
		else
			in_goal=0;
	}
```

```
	/*
	*robot sees the puck
	*/
	if(SUM_IR(PUCK) > SEE_PUCK_THRESHOLD)
		see_puck=1;
	else
		see_puck=0;


	/*
	*robot is stuck
	*/
	if(passed(stuck_checkpoint+STUCK_TIME))
	{
		/*
		 *if motor encoders have not moved enough
		 *we consider the robot as stuck
		*/
		if((abs(enc1-enc1_pos) < STUCK_INTERVAL && abs(enc2-enc2_pos)
< STUCK_INTERVAL))
		{
			stuck=1;
			if (STUCK_LEFT_TRIGGER)
			{
				left_stuck = 1;
			}
			else
			{
				left_stuck = 0;
			}
			if (STUCK_RIGHT_TRIGGER)
			{
				right_stuck = 1;
			}
			else
			{
				right_stuck = 0;
			}
		}
		else
		{
			stuck=0;
		}
		stuck_checkpoint=soft_time();   //reset stuck timer
		//get new encoder values
		enc1=enc1_pos;
		enc2=enc2_pos;
	}


}

// reset clock
void reset_stuck_time()
{
```

```
	stuck_checkpoint=soft_time();
}

//prints IR-, optical encoder-, reflex detector values + close_off flag
void print_sensors(int t)
{
	clear_disp();
	sprint16(buffer,irval[FL][t]);
	lcd_print1(buffer,0);
	lcd_print_at_cursor(" ");
	sprint16(buffer,irval[FR][t]);
	lcd_print_at_cursor(buffer);
	lcd_print_at_cursor(" ");
	sprint16(buffer,irval[BL][t]);
	lcd_print_at_cursor(buffer);
	lcd_print_at_cursor(" ");
	sprint16(buffer,irval[BR][t]);
	lcd_print_at_cursor(buffer);

	buffer[0]=ENC2A?'X':'O';
	buffer[1]=ENC1A?'X':'O';
	buffer[2]=close_off?'X':'O';
	lcd_print2(buffer,0);
	lcd_print_at_cursor(" ");

	sprint16(buffer,adval[FRONT]);
	lcd_print_at_cursor(buffer);
	lcd_print_at_cursor(" ");
	sprint16(buffer,adval[BACK]);
	lcd_print_at_cursor(buffer);
}

//print which state the robot is currently in
void print_state(int state)
{
	clear_disp();
	switch(state)
	{
		case 0:
			strcpy(buffer,"go to start pos");
			lcd_print1(buffer,0);
			break;

		case 1:
			strcpy(buffer,"get out of goal");
			lcd_print1(buffer,0);
			break;

		case 2:
			strcpy(buffer,"score");
			lcd_print1(buffer,0);
			break;

		case 3:
			strcpy(buffer,"cl off w puck");
			lcd_print1(buffer,0);
			break;
```

```
            case 4:
                  strcpy(buffer,"goto off w puck");
                  lcd_print1(buffer,0);
                  break;

            case 5:
                  strcpy(buffer,"cl def w puck");
                  lcd_print1(buffer,0);
                  break;

            case 6:
                  strcpy(buffer,"hunt the puck");
                  lcd_print1(buffer,0);
                  break;

            case 7:
                  strcpy(buffer,"hunt puck def");
                  lcd_print1(buffer,0);
                  break;

            case 8:
                  strcpy(buffer,"rotate");
                  lcd_print1(buffer,0);
                  break;

            case 9:
                  strcpy(buffer,"go to own goal");
                  lcd_print1(buffer,0);
                  break;

            case 10:
                  strcpy(buffer,"g unstuck w p");
                  lcd_print1(buffer,0);
                  break;

            case 11:
                  strcpy(buffer,"get unstuck");
                  lcd_print1(buffer,0);
                  break;

            default:
                  strcpy(buffer,"NO STATE");
                  break;
      }
}

//counts the number of ticks on the optical encoder discs
void opto_delay(int wheel,int length)
{
      int j=0;
      temp_enc2=ENC2A;
      temp_enc1=ENC1A;
      stuck_checkpoint=soft_time();

      /*
      *which wheel?
```

```
	*/
	switch(wheel)
	{
		case LEFT_WHEEL:
			while(j<length)
			{
				if( temp_enc2 != ENC2A )
				{
					j++;
					temp_enc2=ENC2A;
					print_sensors(OFF_GOAL);
					stuck_checkpoint=soft_time();
				}
				//break function if the robot gets stuck
				if(passed(stuck_checkpoint+STUCK_TIME))
					break;
			}
			break;

		case RIGHT_WHEEL:
			while(j<length)
			{
				if( temp_enc1 != ENC1A )
				{
					j++;
					temp_enc1=ENC1A;
					print_sensors(OFF_GOAL);
					stuck_checkpoint=soft_time();
				}
				if(passed(stuck_checkpoint+STUCK_TIME))
					break;
			}
			break;

	}
}
```

15.2 main.h

```
// function declaration
#ifndef MAIN
#define MAIN

void sense();
void go_to(int target,int speed);
void init();
void delay(int ms);
void print_sensors(int t);
void reset_stuck_time();
void print_state(int state);
void opto_delay(int wheel,int length);

#endif
```

15.3 states.c

```
// State Functions
#include <stdlib.h>
#include <pic.h>
#include "states.h"
#include "globals.h"
#include "ir_comm.h"
#include "main.h"
#include "pwm.h"
#include "intr.h"
#include "adc.h"

// functions
void go_to_start_pos()
{
      int temp1;

      if(verbose)
            print_state(0);
      else
            print_sensors(OFF_GOAL);

      reset_stuck_time();
      motors(FAST_SPEED,FAST_SPEED);
      delay(20);

      temp1=soft_time();
      sense();
      /* go back to halfline or until we're close to defensive goal or
       * until 2 secunds has passed,
       * whichever comes first
       */
      while(sample_ad_channel(FRONT)<IN_GOAL_THRESHOLD &&
              sample_ad_channel(BACK)<IN_GOAL_THRESHOLD && !close_def &&
!passed(temp1+2000))
      {
            /*
            *if we get stuck while going back
            */
            if(stuck)
                  get_unstuck();
            else
                  go_to(DEF_GOAL,MEDIUM_SPEED);
            sense();
      }

      //drive a little bit further
      motors(MEDIUM_SPEED,MEDIUM_SPEED);
      delay(400);

      // keep close_off low until reaching the centre line, to avoid
      // the robot from believing that the centre line is the
      // offensive goal
      close_off = 0;
```

```
	reset_stuck_time();
}

void get_out_of_goalzone() {

	if(verbose)
		print_state(1);
	else
		print_sensors(DEF_GOAL);

	if(facing_goal)
		motors(-FAST_SPEED,-FAST_SPEED);
	else
		motors(FAST_SPEED,FAST_SPEED);

	delay(150);
}

void score() {

	if(verbose)
		print_state(2);
	else
		print_sensors(OFF_GOAL);


	//stop the robot
	motors(-FAST_SPEED,-FAST_SPEED);
	delay(110);
	motors(0,0);

	//aim
	motors(0,-MEDIUM_SPEED);
	opto_delay(RIGHT_WHEEL,16);
	motors(-MEDIUM_SPEED,0);
	opto_delay(LEFT_WHEEL,22);
	motors(0,0);

	//shoot
	motors(FAST_SPEED,FAST_SPEED);
	opto_delay(LEFT_WHEEL,20);

	//back out
	motors(-FAST_SPEED,-FAST_SPEED);
	delay(400);

	// play winning song
	play_tune(fur_elise);

	// turn more before heading back
	motors(-FAST_SPEED,SLOW_SPEED);
	delay(100);

	// not close offensive goal
	close_off = 0;

}
```

```
void close_off_goal_with_puck() {

	if(verbose)
		print_state(3);
	else
		print_sensors(OFF_GOAL);

	/*
	*only slow down once when close to goal
	*/
	if (scoring_flag)
	{
		motors(-FAST_SPEED,-FAST_SPEED);
		delay(20);
		scoring_flag = 0;
	}
	go_to(OFF_GOAL,MEDIUM_SPEED-5);
}

void goto_off_goal_with_puck() {

	if(verbose)
		print_state(4);
	else
		print_sensors(OFF_GOAL);

	go_to(OFF_GOAL,MEDIUM_SPEED+5);
}

void close_def_goal_with_puck() {
	if(verbose)
		print_state(5);
	else
		print_sensors(DEF_GOAL);
	go_to(OFF_GOAL,MEDIUM_SPEED);
}

void hunt_the_puck() {

	if(verbose)
		print_state(6);
	else
		print_sensors(PUCK);

	/*
	*go a little bit slower if we're close to the puck
	*/
	if(close_puck)
		go_to(PUCK,MEDIUM_SPEED);
	else
		go_to(PUCK,FAST_SPEED);
}

void hunt_the_puck_close_def() {
```

```
	if(verbose)
		print_state(7);
	else
		print_sensors(OFF_GOAL);

	/*
	*go very slow so we don't push the puck into our own goal zone
	*/
	if(close_puck)
		go_to(PUCK,SLOW_SPEED);
	else
		go_to(PUCK,MEDIUM_SPEED);
}


//rotate on the spot (close to defensive goal) if we don't see the puck
void rotate() {
	if(verbose)
		print_state(8);
	else
		print_sensors(DEF_GOAL);

	motors(MEDIUM_SPEED,-MEDIUM_SPEED);
}

void go_to_own_goal() {
	if(verbose)
		print_state(9);
	else
		print_sensors(DEF_GOAL);
	if (off_half)
		go_to(DEF_GOAL,FAST_SPEED);
	else
		go_to(DEF_GOAL,MEDIUM_SPEED);
}

void get_unstuck_with_puck() {
	if(verbose)
		print_state(10);
	else
		print_sensors(OFF_GOAL);

	/*
	*We first try if we can go forward (if motors are a bit weak and
	*we have the puck the
	*robot sometimes needs a kick start)
	*/
	motors(FAST_SPEED,FAST_SPEED);
	delay(50);
	/*
	*if not, we do the real stuck method
	*
	*if encoders still haven't moved (since last sense-function)
	*enough we're still stuck
	*/
```

```
	if((abs(enc1-enc1_pos) < STUCK_INTERVAL) && (abs(enc2-enc2_pos) <
STUCK_INTERVAL))
	{
		//back out a bit first
		motors(-FAST_SPEED,-FAST_SPEED);
		delay(65);

		/*
		*if stuck on both sides , i.e we're wedged in
		*/
		if (right_stuck && left_stuck)
		{
			motors(-SLOW_SPEED,-SLOW_SPEED);
			delay(100);
		}

		/*
		*turn counter clockwise to keep the puck
		*/
		else
		{
			motors(-SLOW_SPEED,FAST_SPEED);
			delay(300);
		}
	}
	stuck = 0;
}

void get_unstuck() {

	if(verbose)
		print_state(11);
	else
		print_sensors(OFF_GOAL);

	//back out a bit first
	motors(-FAST_SPEED,-FAST_SPEED);
	delay(65);

	/*
	*if stuck on both sides , i.e. we're wedged in
	*/
	if (right_stuck && left_stuck)
	{
		motors(-SLOW_SPEED,-SLOW_SPEED);
		delay(100);
	}

	/*
	*if stuck on right side
	*/
	else if(right_stuck)
	{
		motors(-SLOW_SPEED,FAST_SPEED);
		delay(300);
	}
	/*
```

```
	*if stuck on left side
	*/
	else
	{
		if(SENSOR_1 == FL)
		{
			motors(-SLOW_SPEED,FAST_SPEED);
			delay(300);
		}
		else if(SENSOR_1 == FR)
		{
			motors(FAST_SPEED,-SLOW_SPEED);
			delay(200);
		}

	}

	stuck = 0;
}
```

15.4 states.h

```
// State Functions

// function declaration
#ifndef STATES
#define STATES

void go_to_start_pos();
void get_out_of_goalzone();
void score();
void close_off_goal_with_puck();
void goto_off_goal_with_puck();
void close_def_goal_with_puck();
void hunt_the_puck();
void hunt_the_puck_close_def();
void rotate();
void go_to_own_goal();
void get_unstuck_with_puck();
void get_unstuck();

#endif
```

15.5 globals.h

```
// Global definitions
#if GLOBALS
#else
#define GLOBALS

#define FL 0            //forward left ir-sensor
#define FR 1            //forward right ir-sensor
#define BR 2            //back right ir-sensor
#define BL 3            //back left ir-sensor
#define IR_PUCK 4
#define SUM_IR(x) irval[FL][x]+irval[FR][x]+irval[BL][x]+irval[BR][x]
#define STUCK_TIME 500   //if wheels hasn't moved in STUCK_TIME ms , we
are stuck
#define CLOSE_TO_GOAL_THRESHOLD 220 // old value 170
#define CLOSE_TO_PUCK_THRESHOLD 800
#define IN_GOAL_THRESHOLD 190
#define HAVE_PUCK_THRESHOLD 4200
#define SEE_PUCK_THRESHOLD 40
#define FRONT 0
#define BACK 1
#define STUCK_LEFT_TRIGGER (!bittst(PORTB,4))
#define STUCK_RIGHT_TRIGGER (!bittst(PORTB,5))
#define STUCK_INTERVAL 5
#define RIGHT_WHEEL 1
#define LEFT_WHEEL 2
#define ENC1A RA4
#define ENC2A RE1

// more
extern bank1 int SENSOR_1;
extern bank1 int SENSOR_2;
extern bank2 int irval[4][3];
extern bank1 unsigned int sometime;
extern bank1 unsigned int stuck_checkpoint;
extern bank2 int adval[2];
extern bank1 int irval_stick;
extern bank1 char enc1,enc2;
extern bank3 bit temp_enc1,temp_enc2;
extern bank3 bit verbose;
extern bit have_puck;
extern bit in_goal;
extern bit facing_goal;
extern bit close_def;
extern bit close_off;
extern bit see_puck;
extern bit off_half;
extern bit stuck;
extern bit close_puck;
extern bit left_stuck;
extern bit right_stuck;
extern bit scoring_flag;

// songs
extern const char blues_notes[];
```

```
extern const char laser[];
extern const char klingon[];
extern const char fur_elise[];

//speeds
#define FAST_SPEED 127
#define MEDIUM_SPEED 110
#define SLOW_SPEED 95
#define STOPPED 0

#endif
```