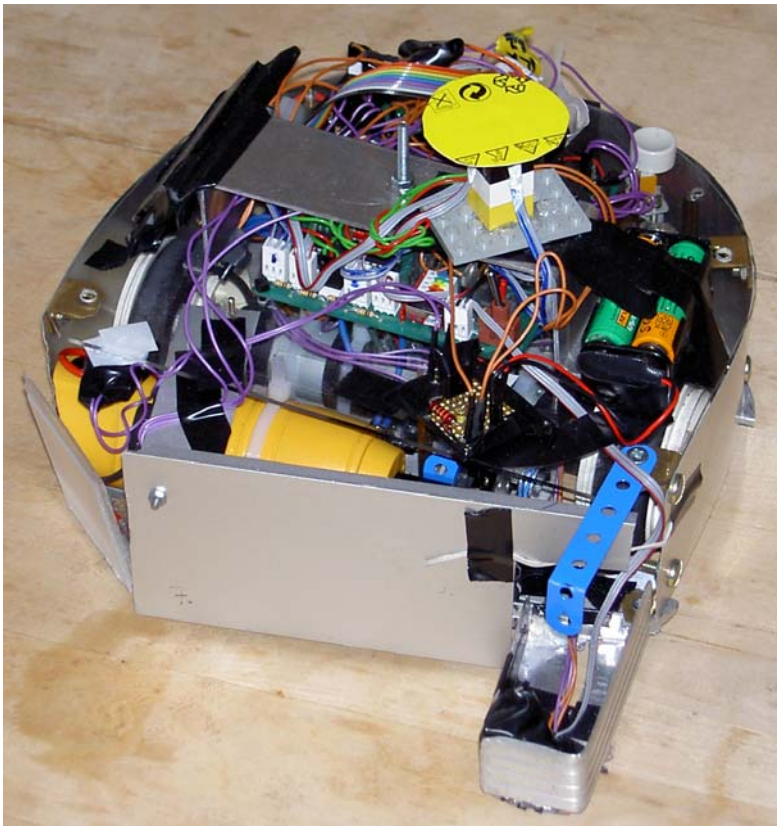
ROBOT PROJECT REPORT

A summary of the construction of the Screwdriver robot



Fredrik Ahlgren
Simon Gustafsson
Daniel Wardzynski

The project report was performed at
NADA, Department of Numerical Analysis and Computer Science
Stockholm, October 2003

NADA
KTH, Royal Institute of Technology
Stockholm, Sweden

2D1426, Robotics and Autonomous Systems
Project report
Fredrik Ahlgren
Simon Gustafsson
Daniel Wardzynski

Abstract

This report is for the course *2D1426 Robotics and Autonomous Systems*, NADA, at *the Royal Institute of Technology*, KTH.

The main focus of this course was to build a small two-wheeled robot that is supposed to play ice hockey on a specially designed ice hockey rink. The actual construction was the most time consuming part, and this report explains in detail the robot design, the background, ideas, implementation and conclusion. We explain the mechanical ideas and the final solution we came to use, as well as the internal circuits and electronics. As we also have attached the C source code, we explain the different behavior algorithms behind our winning strategy.

The final objective of building this “hockey-robot” was competing against other robots in a robot hockey tournament at the end of the course, at the Technical Museum in Stockholm.

We hope that future readers, including you, could continue exploring the “screwdriver”-force and make it better and even stronger. Use as much of our information, including detailed remarks and tips, as possible. Good luck!

2D1426, Robotics and Autonomous Systems
Project report
Fredrik Ahlgren
Simon Gustafsson
Daniel Wardzynski

Table of contents

1 Background

This report is describing the project of constructing the Screwdriver robot, within the course *2D1426 Robotics and autonomous systems* at the *Royal Institute of Technology*, Stockholm. The objective of the construction of this robot is to get an overall understanding of robotics and its basic ideas, as well as overcoming the constraints involved. Robotics is a very interesting multidisciplinary area that involves automation, i.e. electronics and mechanics. As well as computational intelligence, i.e. learning, reasoning, signal processing, sensor fusion, behaviors and neural networks. The main functions that robotics involve are perception, i.e. perceiving (seeing with senses) the environment, and action, i.e. manipulation and locomotion (explained further down the report). Sensors are electrical components that give the robot the ability to perceive the environment. In order to move around or “understand” the newly perceived environment, the robot must use smart algorithms together with important mechanical implementations.

Within this course the main project is constructing a two wheeled autonomous robot that can play ice hockey with special rules, customized to fit the robots involved. This robot is quite simple and should be able to score a goal to pass the requirements for the course. The main parts of the robot is the actual “brain”, the PIC16F877 microcontroller from Microchip containing the actual “intelligence”, i.e. C-code, and the motherboard controlling the communication from and to the microchip. Then we have the motors, R/C servos and five IR-sensors. By using the techniques and methods learned under the lectures, we should implement the technology and create special behaviors for the robot design. In order to do this we need to combine mechanics and software programming.

In the end we must compete with robots from other groups, to test the robots and their abilities against each other. We were seven groups fighting against each other on the specially designed ice hockey rink.

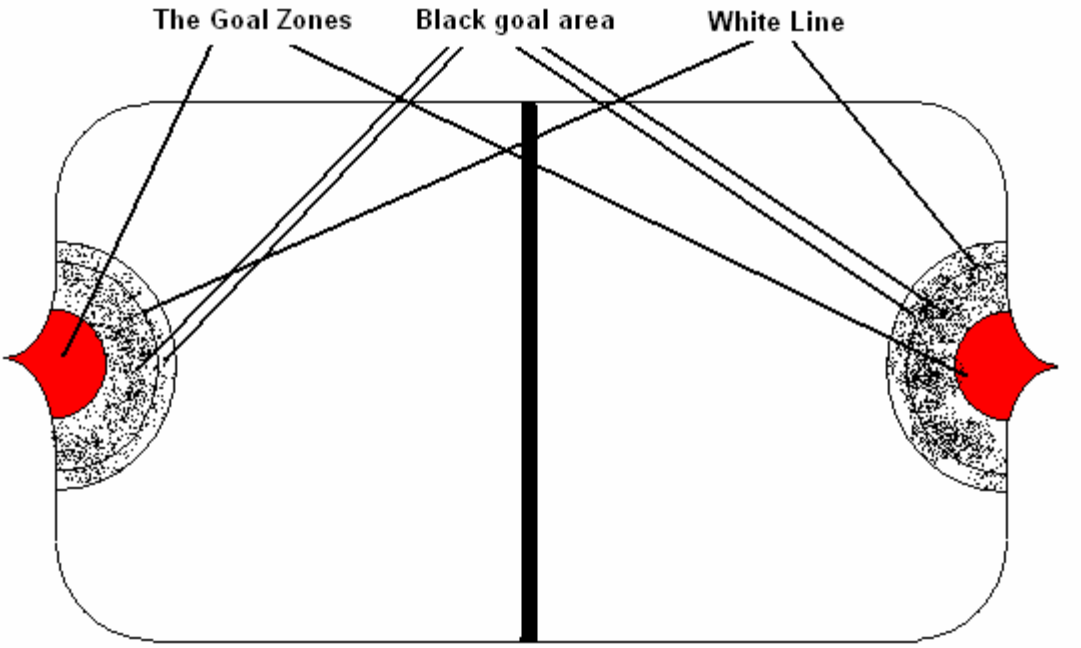
2D1426, Robotics and Autonomous Systems
Project report
Fredrik Ahlgren
Simon Gustafsson
Daniel Wardzynski

1.1 Robot ice hockey

Our so called “robot ice hockey” involves specially designed equipment with therefore specially designed rules.
This means that the robots are not really playing on real ice, but on a wooden floor and with wheels instead of skates.
The goals are made up of IR-diodes that emit pulse modulated signals (each goal sends out a signal with a different pulse frequency). The puck works in the same way yet with a signal with a different frequency. Since the rink is not too big (will explain further down), the robot body must fit in a circle with diameter 250 mm (as measured on the vertical projection on the floor). Look further in the appendix (Robot hockey rules). The vertical projection of the robot body may not include concavities of any kind. Every robot is allowed to have an ice hockey stick to guide the puck, or use it to shoot the puck with. The stick can not extend more than one puck diameter (i.e. 8 cm) from the robot body, and can not be wider then one puck radius. The most important thing, the robots shall not kill, i.e. they can not have any device mounted that could harm another robot.

1.2 Rink

The ice hockey rink is not more then 2.5 meter long and 1.5 meter wide. The wooden floor is white except one black line in the middle, showing the middle of the rink. There are also two black goal yards with a white stripe inside, about 2 cm from the outer bound of the goal yard.
Each goal is marked by a red area closest to the goal IR-emitters. The IR-emitters are placed behind aluminum stripes that protect the IR-diodes from the puck or crazy robots violating the rules. The puck has a diameter of 8 cm and a height of 5 centimeters.
Same things apply to the puck as to the goals. The puck emits signals with a different modulation so the robot can easily distinguish between the puck and the 2 other goals.



The rink and the different areas included.

1.3 The tournament and its rules

The following rules are extracted from M.Bratt´s “Robotics and autonomous systems” [4].
To score a goal the puck must be pushed or shoot into the goal, it means that the puck should enter the red zone closest to the goal diodes. In the same time the robot should not have violated any rules. If this would occur, and the robot would pass the white line in the goal yard, the score is not counted and the puck is released in the middle of the rink. If the robot is pushed into the goal yard the score is counted. If a robot enters his own goal yard voluntary receives a penalty of 10 seconds outside the rink. If the robot stops working there is a possibility to enter “reparation” and then the robot must be removed from the rink for 30 seconds (while repairing). If a robot is stuck against another robot the puck may be removed and released over again in the middle.

The first round of the tournament is called the qualification round, where each robot must score as many goals as possible during 2 minutes. Our robot scored 15 goals during that time, which made him qualify to position number 1. Based on this qualification round, two groups are formed. In group play the games are 3 minute long. The two best robots from each group go into the semi final round, where the games are 4 minutes long. The two looser robots plays for the 3rd and 4th place, while the winning robots from the semi-final round plays in the final. The final game is

played for 5 minutes. The semi-final and final game can not end in a draw, so after regular time the robots enter sudden death where the game is decided.

2 Initial design ideas and goals

The first steps we took before starting to build our robot was to study how other robots behaved during last year's competition [1], and that was very useful to us.

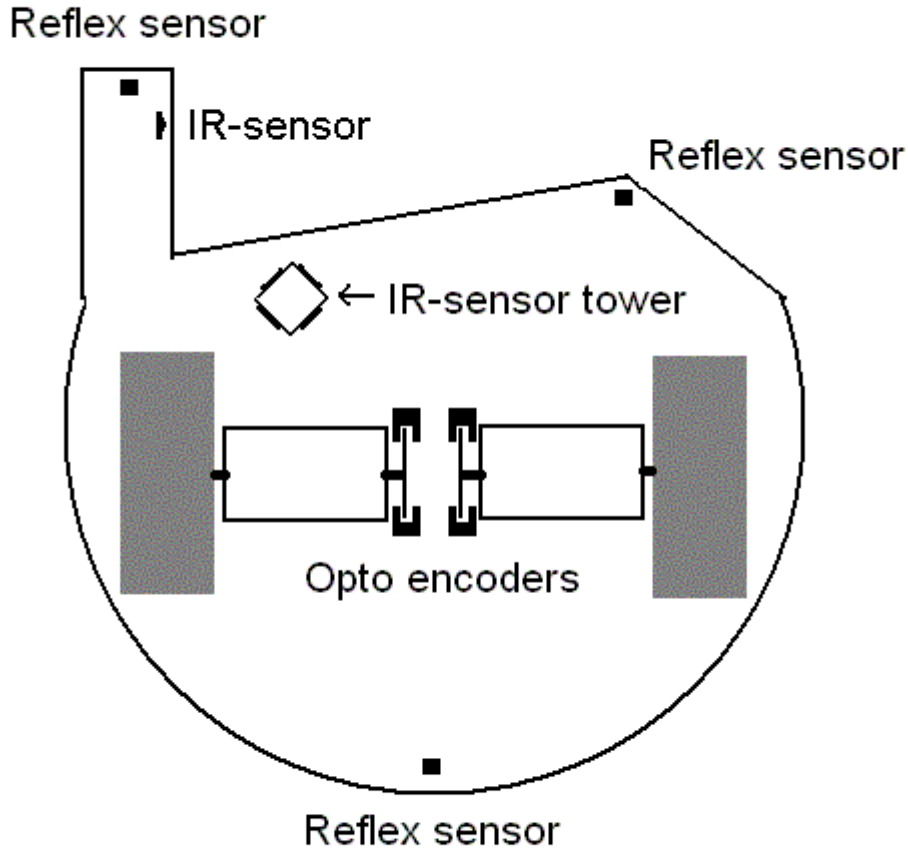
From last years competition we saw that robots with rounder corners didn't have so much problems of getting stuck to the rink, or to other robots. Therefore we gave our robot a smooth rounded surface to prevent it from stucking to other robots or the rink. It turned out to be an advantage. After looking at some of the earlier reports we found that the robot Nätagget [3] seemed to have a good way of getting the puck away from the side of the rink. We decided to use the same kind of behavior with small modifications.

Very early we decided that we wanted to have a shooting mechanism on our robot, since last years winner FejkOpaque [2] impressed us so much. A lot of different ideas only made it to the design stage, and most of them had to be thrown away because they wouldn't fit into the robot body that was build before we knew how much space we needed inside. We already had given the robot the maximum allowable diameter, but that wasn't enough for many ideas. We also had decided the height of the robot before thinking too much on the shooting mechanism. But finally, maybe two weeks before the competition, we decided to cramp one promising idea into the robot to try it out…

The space inside the robot also made it hard to use the wheels we had taken, since they had to be mounted too far away from the motor to sufficiently fit the optical encoder discs where we wanted them. This resulted in a robot whose wheels sometimes started to touch the sheets protecting the wheels of the robot, resulting in much slower motions. Closer to the competition date we made a very different (and space saving) coupling of the wheels and the motor, that gave us larger margins inside the robot.

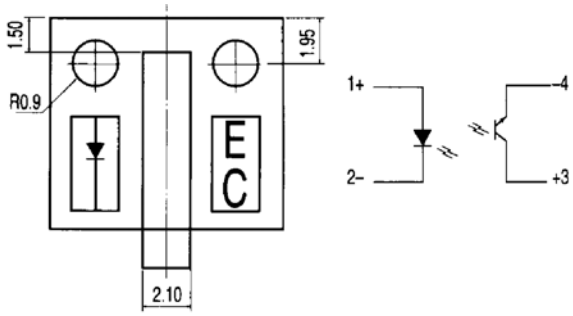
The robot was designed mainly of aluminium and Perspex, to make it rigid and light at the same time. The thickest piece, the bottom of the robot, was drilled to look like a Swiss cheese, to reduce the weight and supply a multitude of mounting options for motors and other devices.

3 Sensing

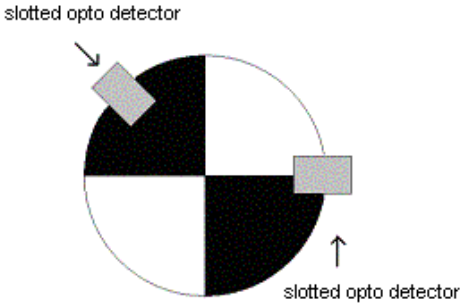


3.1 Optical encoders

A slotted opto detector is in short an IR transmitter and a detector. When nothing obstructs the path of the IR light the detector will get a strong signal. When something blocks the path the detector will get a weak on no signal.

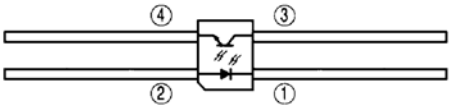


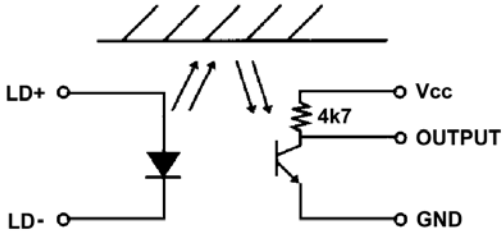
In the case of this robot the optical encoders are used to determine the movement of the wheels. To do this one disc is fastened to the axis of each engine. The disc is divided into transparent and black segments. When the black segment of the disc enters the slotted opto detector, ELITR 8010[9], it obstructs the path of the light which is registered by the detector. By keeping track of the switches in the detector it can be determined if the wheel moves and how fast it moves. By adding one more detector, shifted half a field, the direction of the movement can also be determined [8].
The optical encoders are used in the robot to see if the engines move when they are supposed to. If they do not move for a period of 1000ms the robot will be considered stuck.



3.2 Reflex sensors

The robot is equipped with 3 reflex sensors, ELITR 8307[10], for detection of difference in color on the field. The reflex sensor is an IR-light source and a photo detector for IR-light encapsulated in one package. The light from the IR-source is reflected on the ground and the reflected light is measured.

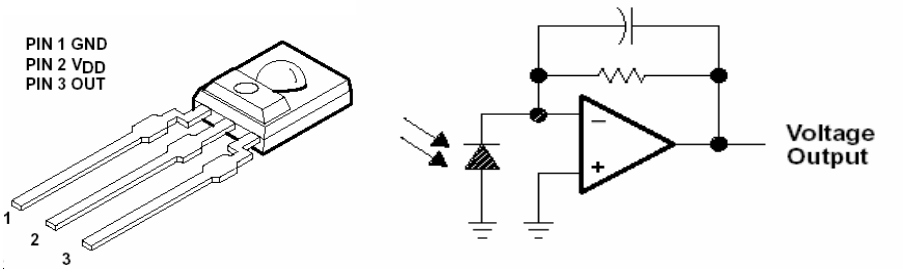




When the robot is positioned on a black surface (like in the goal yard) less light will be reflected than when it is positioned on a bright surface. This makes it possible to determine when the robot enters the goal yard. This is important to make it possible for the robot to keep out of the defensive goal yard to avoid penalty and to stop before entering the offensive goal yard when trying to score.

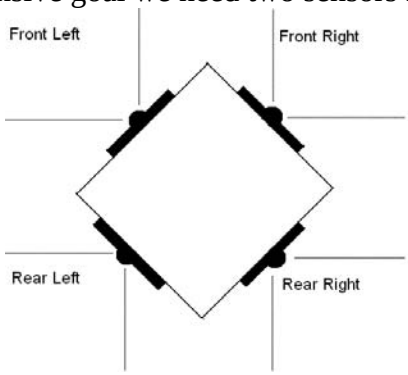
3.3 IR-sensors

An IR-sensor is a device that measures the amount of infrared light in its surroundings. Since there is some IR-light in daylight and in light from electrical light sources the IR-light that should be transmitted and detected can be modulated in a desired frequency. That way it can be distinguished from the unmodulated light. This is used in the light transmitted from the puck and the different goals to make them separable. The main board of the robot is equipped with a pre programmed PIC16F876[7] circuit to process the signals from the IR-sensors.



3.3.1 IR-sensors for steering

To make the “eyes” of the robot there is fore IR-sensors (TSL261[11]). To be able to see in all directions the sensors are mounted on a tower where they split the 360 degree field into four fields of vision. By letting two sensors face forward their difference can be used to steer the robot against a target. If we only would like to drive forward this would be enough, but since we planed to make the robot be able to go backwards against defensive goal we need two sensors facing backwards to.



3.3.2 IR-sensor for puck detection

To easily determine whether the robot got the puck or not without having to rely on switches with a narrow field of detection the robot uses an IR-sensor mounted on the stick. When the puck is close enough to the stick the sensor value will rise above a specified threshold and are considered to have the puck in its possession. This also guarantees that the object in its possession is in fact the puck. Switches weren't used as puck detectors since they can't show whether the robot has the puck, or if something else is in front of the robot.

4 Manipulation

The robot manipulates the puck in two ways. Normally it pushes the puck forward by using the body and the stick as guides. But when the robot perceives that it is very close to the offensive goal, it uses a shooting mechanism to shoot the puck into the goal from some distance.

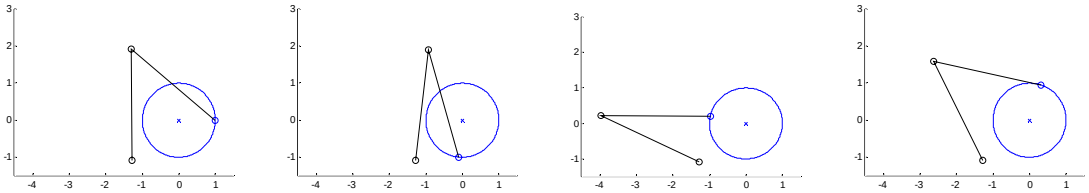
4.1 Shooting

The main goal with the shooting mechanism was to score fast and reliably. The fact that the mechanism also enabled scoring from a distance was seen as a bonus, since it prevents the robot from going too far into the goal zone and thereby making the attempt to score invalid.

When the robot has acquired the puck, it starts to navigate toward the offensive goal in such a way that it always faces that goal zone. If the robot senses that it is very close to the offensive goal, it activates the shooting mechanism that than punches the puck into the goal.

Since it's difficult to determine whether the robot is close enough to the goal or not, the robot also shoots if the stick on the front of the robot passes the black goal area while the robot senses that it's not very far away from the goal. After shooting, the robot always plays a melody to signal that an attempt to score has been made.

The shooting mechanism consists of a screwdriver that rotates a disk, and a system of links attached to generate a rapid and powerful punch. The punch hits a metal piece on hinges that swings out from the front of the robot and hits the puck. Some snapshots from a simulation of the rotating disk and the links are shown in the figure below:



(a) (b) (c) (d)

Figure 1: The initial resting position (a); position (b) after a quarter of a revolution; comparing (b) and (c) shows the magnitude of the punch caused by just another quarter of revolution. The punch is rather fast; the rest of the cycle (d) is used to get back to the initial position

While constructing the mechanism we had to solve some problems: getting the mechanism to return to its initial position after a shot, to reduce the play caused by the loose fit of parts, and fitting of the mechanism inside the robot.

To get the mechanism in its initial position, a slotted opto detector was used. That detector detected an index pulse on a disc attached to the same axis as the rotating disk mentioned previously in the section. To get as close to the resting position as possible, the screwdriver was run at a slow speed (by using PWM modulation), and the motor was stopped as soon as the index pulse was detected. The circuit used to control the screwdriver is further described in section 7.3.

The problem of play caused by loose fit of parts was solved in a rather crude but effective way. Support was added to the screwdrivers axis so it didn't ricochet while shooting. Play caused by the screws holding the links together was solved by making the screws wider by wrapping them with tape. Nevertheless, the screws were tightened periodically between matches, to keep the high performance.

The final problem of fitting the mechanism inside the robot was realized rather late. The length of the punches from the shooting mechanism had to be reduced in order to keep the robot smaller than 25 cm in diameter in its largest state (see rules in appendix, 12.1). The reduced punch had a large effect on the distance the robot could score from, but it fortunately didn't affect the robots ability to score when the robot was instructed to score from a closer distance than before.

5 Locomotion

The robot uses differential steering. That means in short that the robot has two individual driving wheels and two sliding points that only are used to balance the robot body. The driving wheels are connected to two 12 v motors with downshifting gearboxes. Each motor can be set to have a speed from -127 to 127. To make the robot go against a target it sets the motor speeds relative to the readings of the IR-sensors according to the following formula.

$$S = c * \frac{IR\_front\_left - IR\_front\_right}{IR\_front\_left + IR\_front\_right}$$

The result is subtracted from the left motor speed and added to the right. There by creating a difference in speed on the two wheels that makes the robot steer against the target. The steering will become stronger when the target is far to the side but will be small when the target is straight ahead. Thereby the robot will not make any quick movements when the goal is in front of it causing oscillations. The constant c is used to set how strong the robot should be affected by the steering formula. When the robot got the puck it needs to have a bigger difference in motor speed between the motors to compensate for the force created by the friction of the puck. This is achieved by using a bigger value for the constant c in this case.
If the robot gets stronger readings from the rear IR-sensors than from the front IR-sensors, it will instead rotate at full speed in the direction of the desired goal. This is achieved by letting one engine rotate full speed forward and the other full speed backward.

6 Programming environment and computer hardware

6.1 Programming environment

The program for the microcontroller was written in C, to be compiled by the HI-TEC PICC compiler [5]. A free integrated development environment (IDE) from Microchip was used to edit the code, since it was set up to invoke the compiler in an easy way, and since it offered a basic code editor. The free IDE [6] also offered the ability to debug the code, but we didn't find that very useful. That's probably because it was easier to write debugging messages to the LCD attached to the motherboard.

When the code was compiled, the transfer to the robot microcontroller was easy. The microcontroller contained a boot loader, a piece of code executed after each device reset. The boot loader was made to enter a programming mode if one pin of the microcontroller was hold at a low level at reset, otherwise the main program of the microcontroller was executed. In the programming mode, the microcontroller received new code over a serial connection to our computer. The computer therefore had to run a program, invoked by ”LoadRobo.bat”, to transfer the compiled code to the microcontroller.

6.2 Motherboard internals

The motherboard used in this course was customized for small robots[4]. Besides the main microcontroller, the board also contained a preprogrammed microcontroller dedicated to filter incoming IR signals and controlling R/C servos. The motherboard also contained four motor drivers and some voltage regulators.

6.2.1 Main microcontroller

The main microcontroller programmed by us was a PIC16F877 from Microchip. The microcontroller was preprogrammed with a small boot loader that enabled new code to be loaded into the microcontroller over a serial interface. The main microcontroller had the following features [7]:

8K x 14 bits FLASH program memory
368 bytes of RAM registers (SRAM)
20 MHz, 5MIPS RISC
8 level hardware stack
40 pin DIL-capsule

6.2.2 Coprocessor (IR-filtering)

The preprogrammed co-processor of the motherboard, a PIC16F876, was used to filter incoming IR signals, and had the capability to control R/C servos as well. It was connected to the IR sensors and it was used to locate the puck and the goals. No programming of this device was done by us.

6.2.3 Motor and speaker drivers

The motherboard was equipped with two similar motor driver IC:s. The first one was an L293NE with dual H-bridges. It was used to drive the two main motors of the robot. The second motor driver IC was an L293D, who’s only difference to L293NE is that it has internal protection diodes. One H-bridge in the second IC was used to drive the piezo buzzer used by the robot to play tunes.

7 Electronics

This section describes the different electric circuits connected to the robot motherboard. The section is divided into three subsections: **Motherboard externals**, mentioning most peripherals connected to the motherboard; **power wiring**, describing how power is supplied to different parts of the robot; **screwdriver interface**, a separate section in devoted to the circuit controlling the screwdriver.

7.1 Motherboard externals

This section describes the items which were connected to the motherboard. It also describes how they where connected. The screwdriver interface is omitted from this section, and handled separately in section 7.3.
Most connections to the motherboard were made directly to pins of the main microcontroller. Therefore these connections are listed in appendix 12.4. The more complex devices are also describer here.

7.1.1 LCD display

From the beginning the motherboard was supplied with an LCD display, capable of showing two rows of 16 characters each. The display was connected to the motherboard with a piece of flat cable. We experienced a lot of breaks of the solder joints both at the LCD side and the motherboard side of the cable. Hence we resoldered the LCD to a pin header. Thereafter we replaced the solder joints of the cable with connectors fitting the pin headers of the motherboard and the LCD to prevent further problems with the LCD.

7.1.2 RS232 interfacing board

To enable serial communication between a computer and the motherboard, an interfacing circuit had to convert the logic levels of the motherboard into the levels required by the RS232 interface at the computer side. This interface was readymade and was handed out to us at the beginning of the course.

7.2 Power wiring

The power supply of the robot is summarized in figure 3 in the appendix. It’s suggested that figure 3 is viewed to get the main idea before reading the further explanations in this section

7.2.1 Screwdriver batteries

The screwdriver used by the scoring mechanism was powered by a separate pack of NiMH batteries, with a voltage of 4,8 V. The battery pack was connected to the screwdriver interface described in section 7.3.

7.2.2 Main accumulator

The robot was mainly powered by an old 7,2 V NiCd accumulator with a rather low capacity. We experimented with the use of alkalic R6 batteries to get a higher capacity, but it turned out that our heavy robot started to move more sluggishly then. A decision was made to continue to run the robot from the NiCd accumulator to keep its speed up.

7.2.3 Voltage booster card

The 7,2 V main accumulator of the robot was connected to a voltage booster card, which converted the battery voltage to a voltage of 14,5 V. The higher output voltage is then connected to the motherboard and to the LED driver board.

7.2.4 Motherboard power regulators

The motherboard is supplied with 5 V and14,5 V from the voltage booster card. The motherboard supplies two of its four h-bridges directly from 14,5 V from the voltage booster card. Those two h-bridges was used to drive the 12 V locomotion motors. The other two h-bridges, together with any R/C servos was supplied with 8 V from a local voltage regulator. The 5V voltage used to power the logic chips and the microprocessors comes from a second voltage regulator also located on the motherboard.

7.2.5 LED driver board

The LED driver board was used to power the LEDs inside the reflex detectors and slotted opto detectors. In our case the maximum of six LEDs was connected in series to one channel. Since only one of the two channels was working, the additional LEDs required were connected to +8 V with series resistors (See figure 3 in appendix 12.3). The LED driver board was powered directly from the voltage booster card.

7.3 Screwdriver interface

The screwdriver used by the scoring mechanism contained a motor and a gearing system. When the motor was driven by 3.6 Volt, as it was supposed to, the output shaft rotated with 220 rpm. To get a higher rotational speed, the screwdriver was fed by a 4.8 volt battery pack instead. The higher voltage combined with the motors small internal resistance, around 0.6 Ohm, meant that currents greater than 5 Ampere had to be switched to turn the motor on.

A number of interfaces supplying the current to the screwdriver were considered. The motherboard had an unused H-bridge available, but it was rated for a maximum peak current of 1.2 Ampere, way lower than needed, so an alternative solution had to be chosen. Relays were ruled out since we needed to be able to control the speed of the screwdriver. A power transistor (in darlington configuration) was tried, but had a rather large voltage drop, reducing the torque of the motor.

The interface finally chosen was a mosfet power switching transistor, causing virtually no loss of power or motor torque. Schematic of the interface is shown in the figure below. The interface was controlled by a logic signal from the motherboards main processor. The interface circuit also needed +8V, which was supplied by the servo connections on the motherboard.

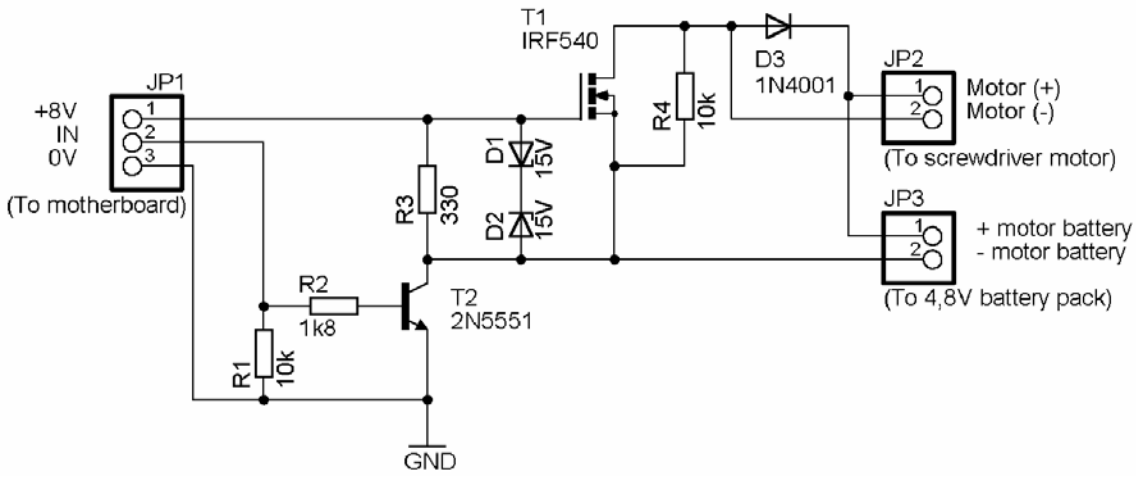


Figure 2: Scheme of the screwdriver interface. Some comments: T2 can be replaced with almost any small signal NPN transistor; R1 prevents the circuit from oscillating when the IN signal isn't connected, and could be omitted; R3 is preferably chosen as a ½ W resistor, the resistance of R3 is low to shorten the switching time (necessary while driving the circuit with PWM-modulated signals); resistor R4 could probably be omitted, just installed on a hunch; the motor battery can't be electrically connected to anything else than this circuit.

8 Software implementation

8.1 Stuck

If the opto-detectors determine that the wheels haven't moved in 100ms the robot is considered stuck. If the robot have the puck in its procession it rotate to the right by backing the right wheel but also drive the left wheel slowly forward so it doesn't lose the puck at the sideboard. If it doesn't have the puck the robot rotates in the direction of the puck on the spot.

8.2 Go to puck

When the robot does not have the puck it is steered against as described in the locomotion section. If the puck happened to be placed far at the side of the robot but still is considered to be in front of it a special solution is activated. The robot will multiply the turning force from the locomotion formula by ten to get a sharper turn.

This makes the robot get to the puck faster and prevent the puck from getting stuck at either side of the robot and cause the robot to be deadlock in a silly dace.

8.3 Go to goal

When the robot needs to steer against a goal it does so in the same way as when it goes for the puck. If the robot have the puck in its possession it puts a little more force on the movement to compensate for the friction caused by the puck.

8.4 Score goal

There is two occasions when the robot will try to score a goal. The first is when it discovers that the stick is over the offensive goal yard. It then try to stop as quick as possible and at the same time shoot the puck. The other is when the robot detects a small difference between the front IR- readings of the goal, but still a big enough sum of the IR-readings. Or in other words when it is facing the goal and is not to far away. It then first shoots the puck and then it stops. The reason why it now stopped after shooting is that it gives better speed for the puck when it gets hit and the puck will have a better and closer position next to the shooting mechanism. After trying to a score the robot in both cases plays the melody and back away for about a second to place itself on the defensive side of the rink.

8.5 Avoid black zones

When any reflex detector see black ice the robot first compare the IR-readings from both goals to decide it is on the middle of the field and the black is the centerline. After that it uses the same IR-readings to decide which side of the rink the robot is on and then drives in the opposite direction of the goal for 500ms. This prevents the robot from entering the black zones around the goals.

8.6 Go home

When the robot does not see the puck it is probably because the puck is behind another robot. The best thing to do then is to run home to the defensive goal to protect it. The steering works in the same way as when going for the puck.

9 Results

The robot performed excellently by itself on the rink. That was proven thoroughly during the qualification round of the competition. Our robot managed to score the most goals of all the robots during these two minutes.

We were aware of problems locating the puck when the robot was far away from the puck. During our testing (alone on the hockey rink) this didn't occur so often, and when it happened, it solved itself when the puck was moved to the middle of the rink, as it is after every goal.

However, during the competition our robot got stuck towards the end of both matches we were playing. The reason seemed to be that when we got stuck closer to our own goal, the opponent was able to block our robots view of the puck. They managed to do that by always returning to a point between our stuck robot and the middle where the puck was dropped after every goal.

The first match this happened, we hoped that the robot would find the puck after the opponent scored a goal. If we would have lifted the robot off the field for 30 seconds, we would have had about 15 extra seconds of useful time at the end of the first match. That might have been enough to make us win that match.

During the rest of the next match, we realized that we had to lift our robot off for 30 seconds, when this behavior occurred, and that gave us some extra useful time on the hockey rink.
To summarize, this was our only large problem during the competition.

One of the reasons that we didn't fix this problem was that we got problems with our gearbox during the last day of coding, and it was getting a lot worse between each test run we made with the robot. We didn't feel that there would be much use of improving our code if the robot would break down before the competition. Therefore we stopped all further development and trial runs early the day before competition. On the competition day we changed the gearbox to a new one made from parts of 2-3 other broken gearboxes, and fortunately it lasted throughout the competition.

To the positive side, we can count our robots high reliability while scoring and its ability not to loose the puck after it got hold of it. Unfortunately this was only enough to win the qualification round.

10 Conclusions

Our robot was very simple from the beginning, built on top of a Lego base, with the motors fastened by steel wires. This helped us become familiar with the hardware and the programming of the microcontroller. Only one IR-sensor was used, so the robot couldn't do much.

Later a base plate was machined out of aluminium, and the robot started to take shape. The angled front of the robot turned out to be a great advantage. Because of that front, we never had problems with loosing the puck while driving to the goal with the puck. Spacers less than 1 dm long were used to fasten the Perspex top of the robot, where most circuits were attached with cable ties, electric tape, and screws. The last thing added to the robot was the scoring mechanism, which enabled us to score fast and reliably. We highly recommend trying to finish the mechanical design rather fast, so more time can be spent on actual coding.

When it comes to coding, we started out creating simple functions mostly to debug the hardware, and that's important to find faults like broken IR-sensors or problems with optical encoders. Most functions was very simple at the beginning of the coding process, but they developed during time, growing as experiments showed us what advantages more elaborate solutions could provide.

This very interesting project have taught us a lot of ”practical robotics”, and given us an insight into the world of experimentally driven design of robots.

11 Suggestions for the future

The two goals have different intensities, so if you have routines dependent on the strength of the goals, you might wish to implement an extra switch to tell the program which goal you want to score in. Otherwise the robot has to be reprogrammed between matches for optimal performance.

Just out of precaution, we also developed a second scoring algorithm, doing the infamous backing up and driving the puck into the goal with the stick. Our way of finding the correct sequence and distance to drive the wheels made this a reliable alternative solution. The robot was equipped with encoder discs at the motors, showing the distance the wheels traveled. To find the distances the robot had to travel, we simply made a program, showing distance of travel on the wheels, and moved the robot by hand in such a way that it backed up, and than hit the puck straight on. The distances read out was then coded into the backup scoring algorithm, and worked flawlessly until we changed the diameter of our wheels… We then had a working system with the screwdriver catapult, so we didn't bother to about remeasure the distances needed by the algorithm.

The paragraph just above also points out that the hardware needs to be built before coding of the software can commence. The software depended on a lot of sensitive threshold levels, so even minor changes in the robot could require a lot of work with the code before it works as good as before.

Make sure that you have motors with new gear boxes, and make sure that there is a stock of them somewhere. I think most groups had problems with them.

Also make sure that you get five working IR-sensors at the first day, and write software to verify it. Two of the IR-sensors our group got were broken, and we luckily were able to get two new ones. This year it seemed to be hard to find new sensors of this kind…

There seems to be a limited supply of computers. This year seven groups had to share six computers, and that was more noticeable towards the end when all groups are coding at the same time. Fortunately one of our group members had a laptop that we developed most of the code on.

Your four new best friends are going to be the Melt adhesive gun, cable ties, electrical tape, and the crimping tool that makes soldering obsolete…

Adhesive melting glue doesn't stick so well on metal. One way around that is to first cover the metal with adhesive electrical tape before applying the melting glue.

Design the robot so it doesn't loose time on the rink. 10 seconds for backing into your own goal yard is bad, but 30 seconds of ”repair time” is worse. It's much easier to score while alone on the field.

Keep in mind that there is no point in trying to score if you don't score. Some other robots seemed to miss more than half of their scoring attempts. That makes the time your robot ends up alone on the rink only worth half of what it could be worth.

The laboratory provided by the course only had the most basic things to offer. To design a robot of something other than Lego or Mechano, or a robot with more specialized circuits, access to a better laboratory like ELAB is highly recommended. One group found a pair of wheels that were only half as wide as the normal wheels found on most robots. They are probably the best choice, so take them if you have the chance.

12 Appendix

12.1 Robot hockey rules

The following rules are extracted from M.Bratt´s “Robotics and autonomous systems” [4].

• Matches are three minutes long except for semi-finals (4 min) 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 signaling 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[1] 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 can not 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. It must be at least 2 cm high.

• 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.

[1] Parts of the robot body that are high enough above the floor to let the puck pass under them do not count

• 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.

• 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.

12.2 Additional robot constraints

• 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[4].

• When guiding the puck, the stick (or any whiskers/bumpers) must not occlude the IR LEDs of the puck from any direction[4].

• 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[4].

12.3 Part lists

Parts for screwdriver interface:

Resistors:	R1 = 10k; R2 = 1k8; R3 = 330; R4 = 10k
Diodes:	D1, D2 = 15 V Zener diodes; D3 = 1N4001
Transistors:	T1 = IRF540; T2 = 2N5551
Other:	JP1 = 3x1 pin header; JP2, JP3 = 2x1 pin headers; 4,8V battery pack

Exchangeable parts of main board

1 PIC16F877 main microcontroller
1 PIC16F876 preprogrammed coprocessor
1 L293NE motor driver
1 L293D motor driver
1 74HCT04N

Sensors

5 IR-sensors TSL261
5 Slotted opto detectors, ELITR 8010
3 Reflex detectors, ELITR 8307

Other devices
1 RS232 interfacing board
1 LED driver board
1 LCD-display Power tip PC1602D A
1 Voltage booster card 7,2V input, 14,5V output
1 Main accumulator 7,2 V, 1700mAh NiCd accumulator pack
1 Screwdriver battery pack (4 pcs AA (R6) NiMH batteries)
1 Power switch
1 Piezoelectric buzzer (with 4k7 potentiometer in series)
1 Screwdriver interface (see figure 2)
1 Screwdriver model Robust; speed 220rpm; rated voltage 3,6V.
2 12V motors with 10:1 reduction, HL149
2 resistors, 330 Ohm

Mechanical parts
2 optical encoder disks, 4 sectors
2 wheels
A lot of aluminium, lexan and screws

12.4 Motherboard connections

Connections made to the main microcontroller

Pin 2 AN0	Reflexdetector at stick (note pull upp used)
pin 3 AN1	Reflexdetector rear end (note pull upp used)
pin 4 AN2	Slotted opto detector, for screwdriver indexing (note pull upp used)
pin 5 AN3	Reflexdetector front (not on stick) (note pull upp used)
pin 6 RA4	Slotted opto detector, ENC1A (note pull upp used) (right)
pin 8 RE0	Slotted opto detector, ENC1B (note pull upp used) (right)
pin 9 RE1	Slotted opto detector, ENC2A (note pull upp used) (left)
pin 10 RE2	Slotted opto detector, ENC2B (note 4k7 pull upp resistor connected to pin 11) (left)
pin 11 VDD	(see comment of pin 10)

pin 15 RC0	Boot loader switch
pin 16 CCP2	third motor control pin (right motor enbl)
pin 17 CCP1	first motor control pin (left motor enbl)
pin 25 UART TX	to TX, RS232 interfacing board
pin 26 UART RX	to RX, RS232 interfacing board
pin 31 VSS	to GND, RS232 interfacing board
pin 32 VDD	to +5V, RS232 interfacing board
pin 33 RB0	second motor control pin (left motor d)
pin 34 RB1	fourth motor control pin (right motor d)
pin 35 RB2	fifth motor control pin
pin 36 RB3	sixth motor control pin
pin 37 RB4	to IN, screwdriver driver board

Connections to motor terminals

motor terminal marked 1	left motor
motor terminal marked 2	right motor
motor terminal marked 3	piezo busser with (4k7 pot in series)

12.5 Schematics

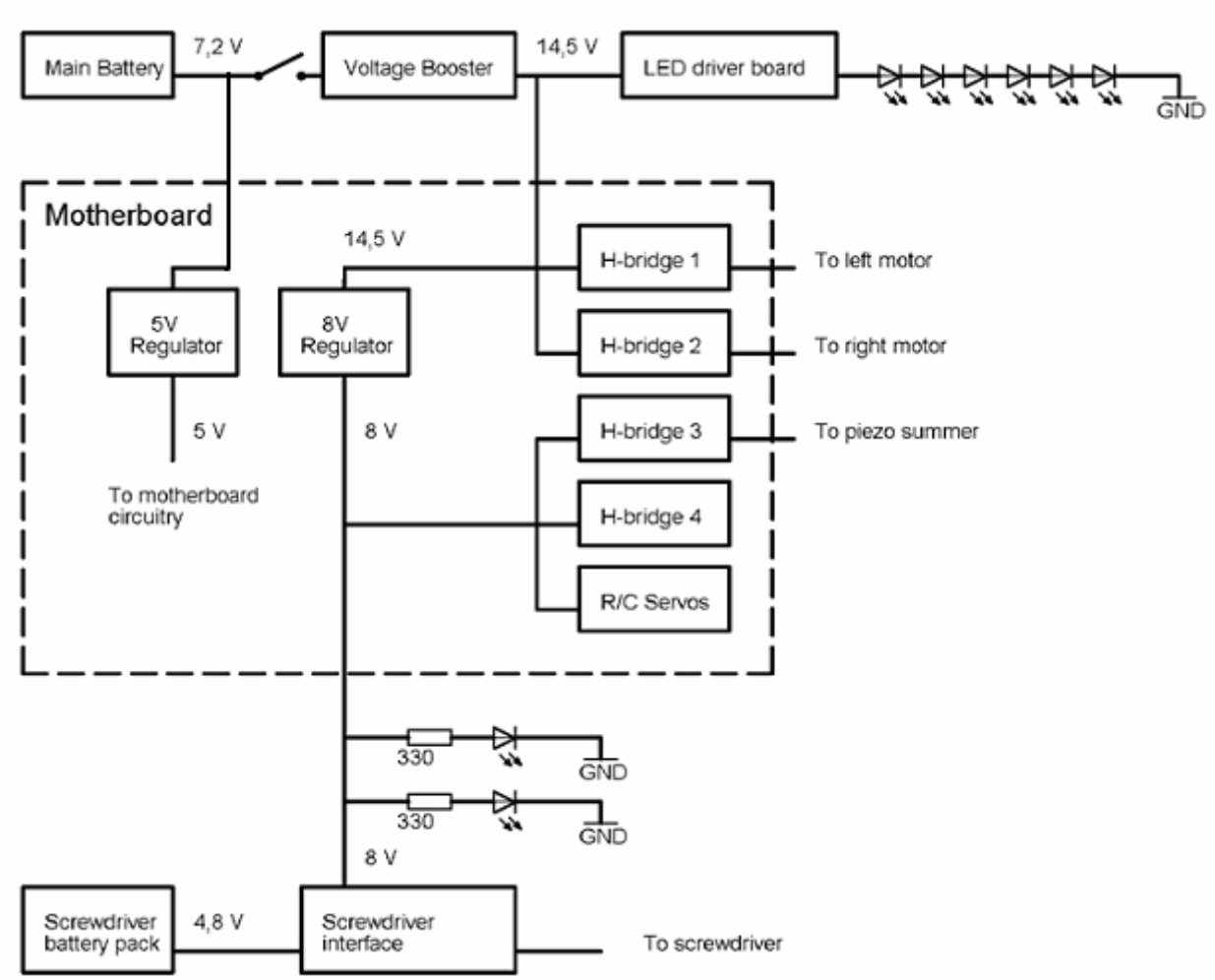


Figure 3: The different blocks involved in supplying power to different parts of the robot. Note that two of the IR-LEDs are powered by 8 V and series resistors, since one channel of the LED driver board was broken.

2D1426, Robotics and Autonomous Systems
Project report
Fredrik Ahlgren
Simon Gustafsson
Daniel Wardzynski

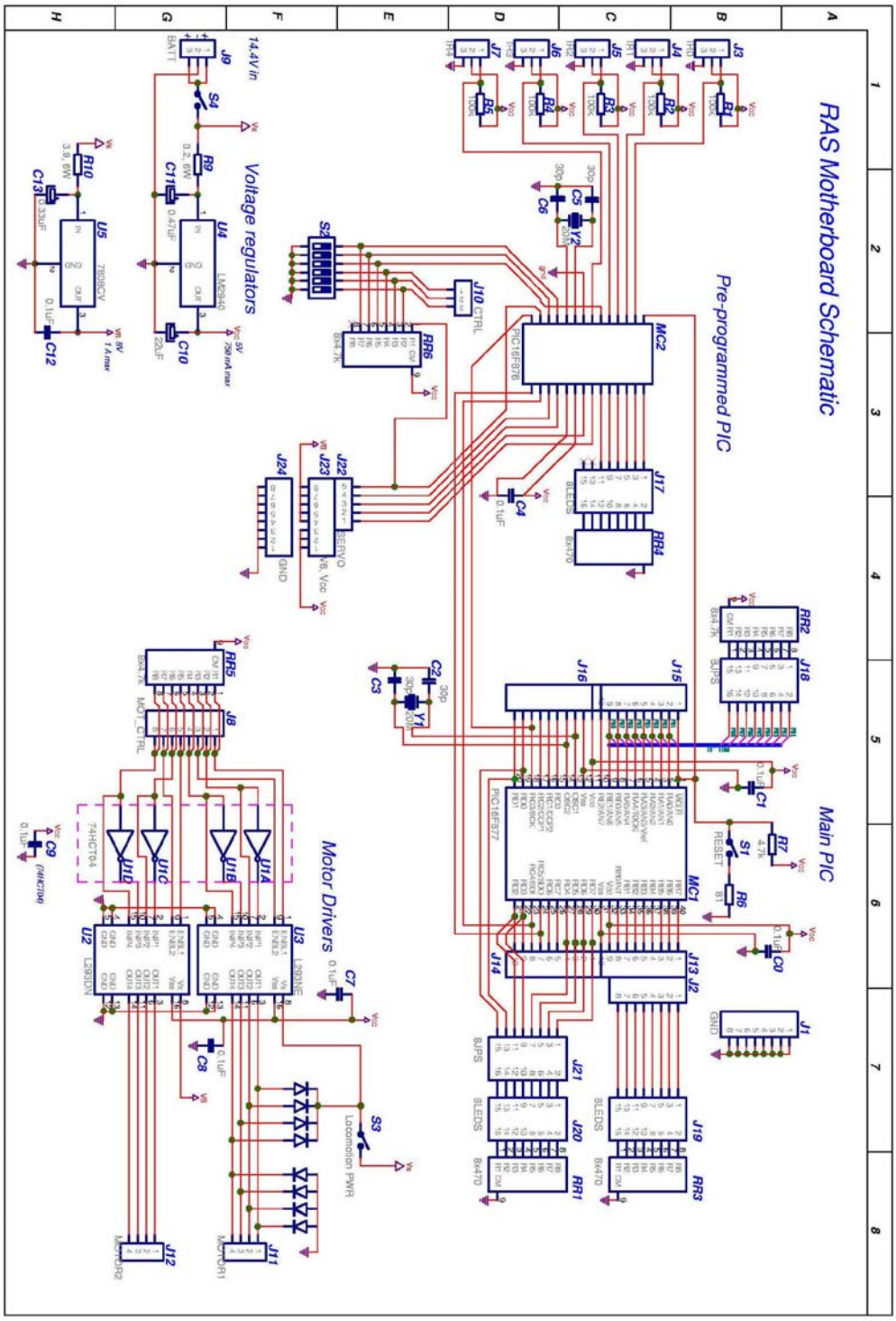
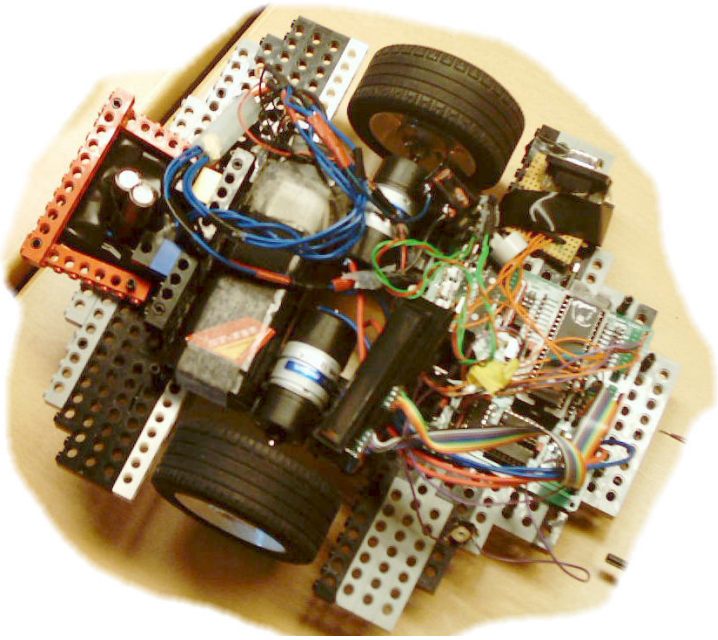
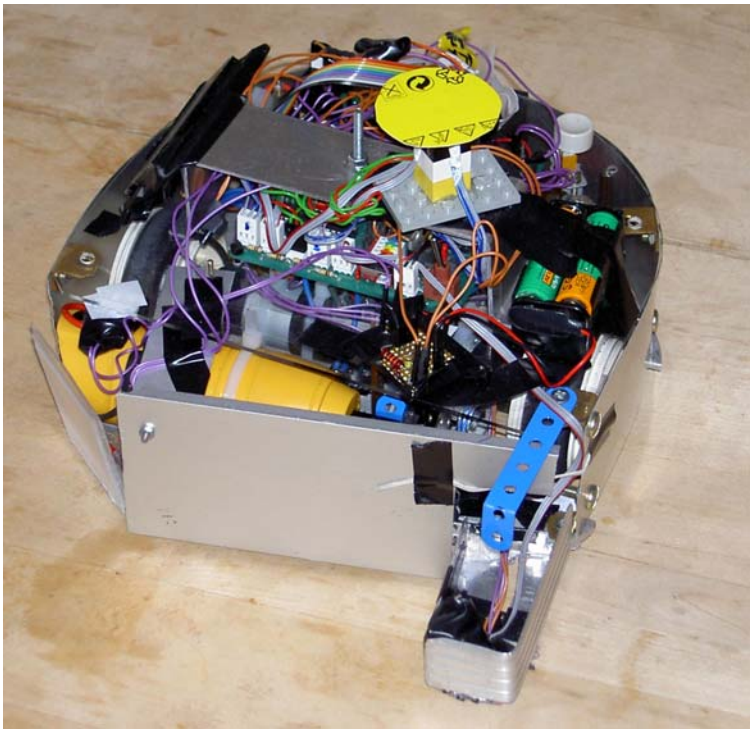


Figure 4: This is the schematic for the motherboard

12.6 Robot Pictures

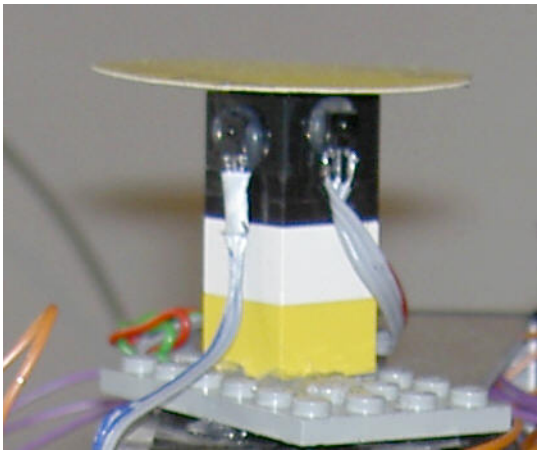


Our early prototype built with LEGO

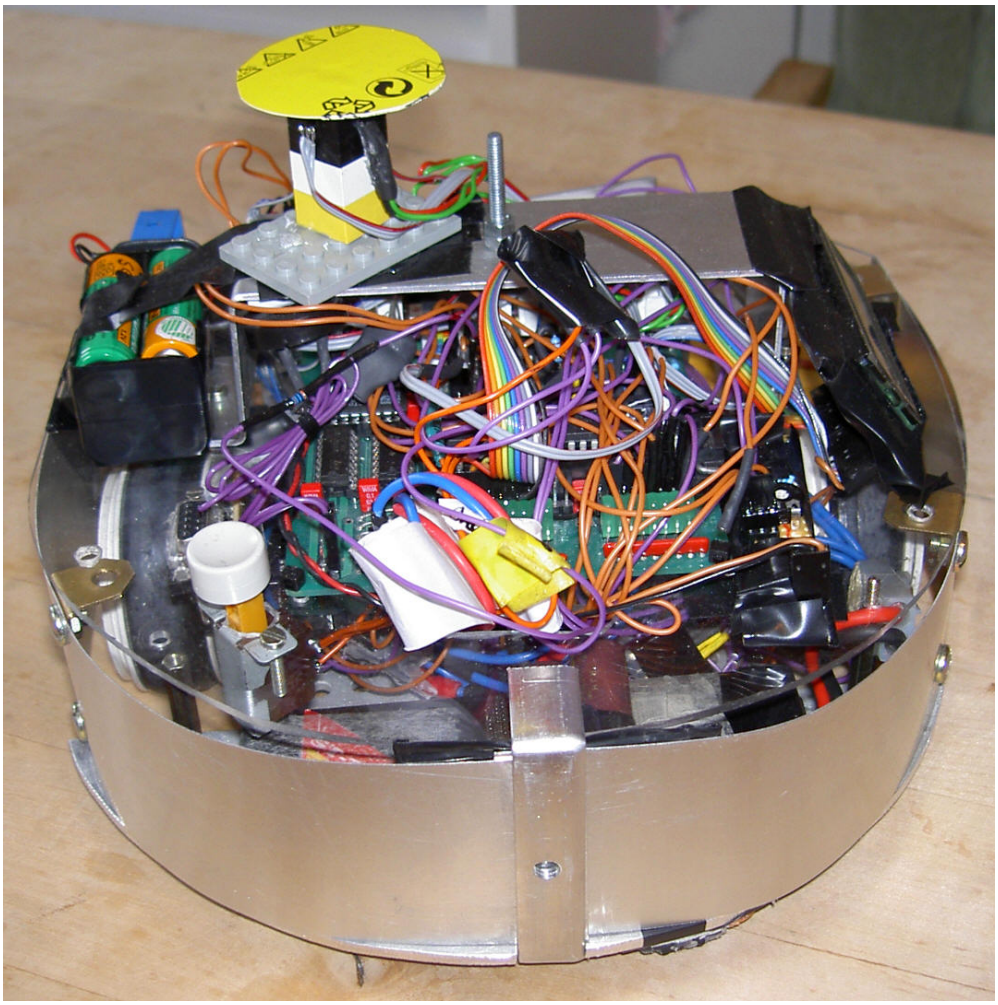


The finished robot

2D1426, Robotics and Autonomous Systems
Project report
Fredrik Ahlgren
Simon Gustafsson
Daniel Wardzynski

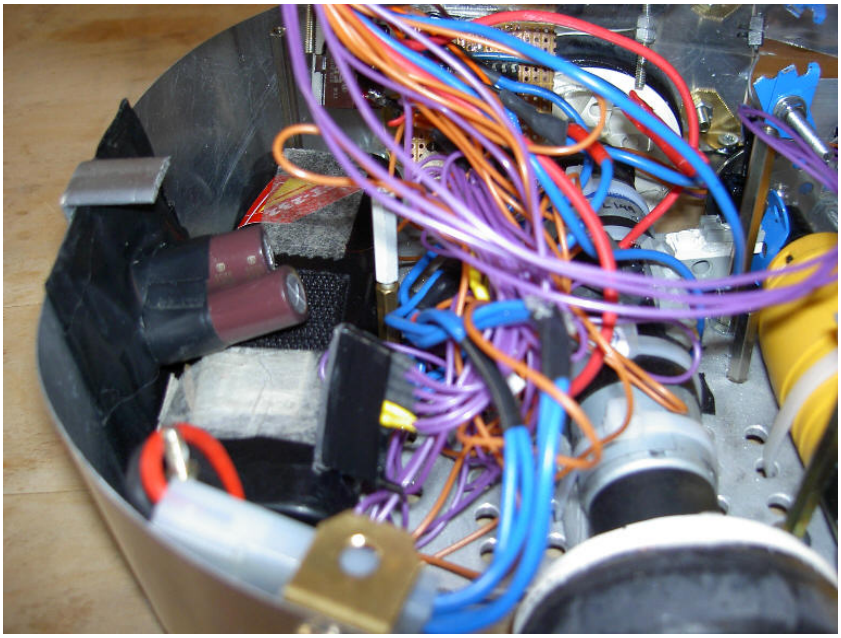


The IR-sensor tower

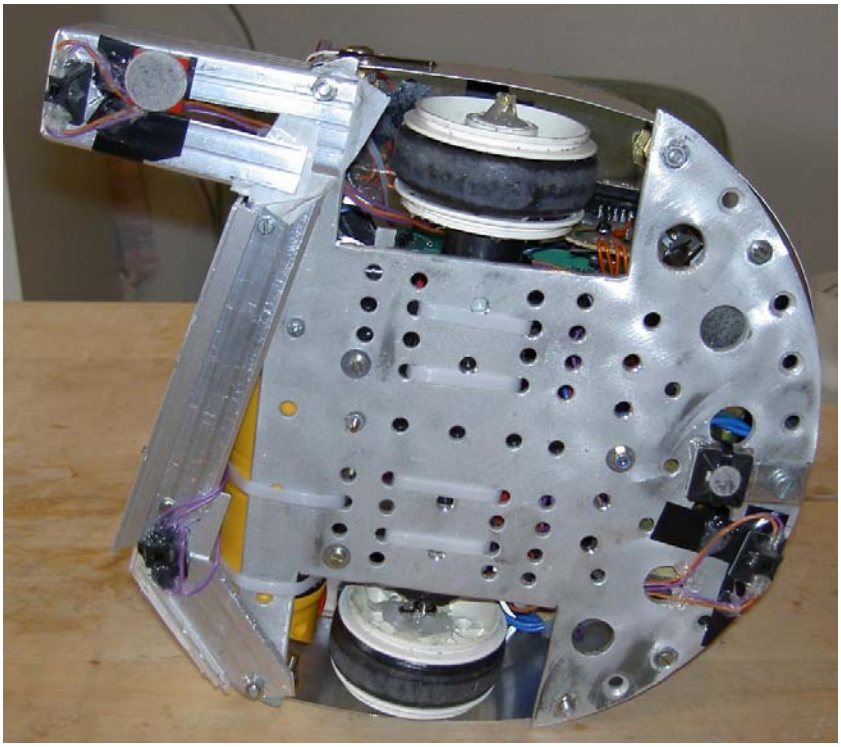


The robot from behind. Observe the excessive use of longer cables.

2D1426, Robotics and Autonomous Systems
Project report
Fredrik Ahlgren
Simon Gustafsson
Daniel Wardzynski

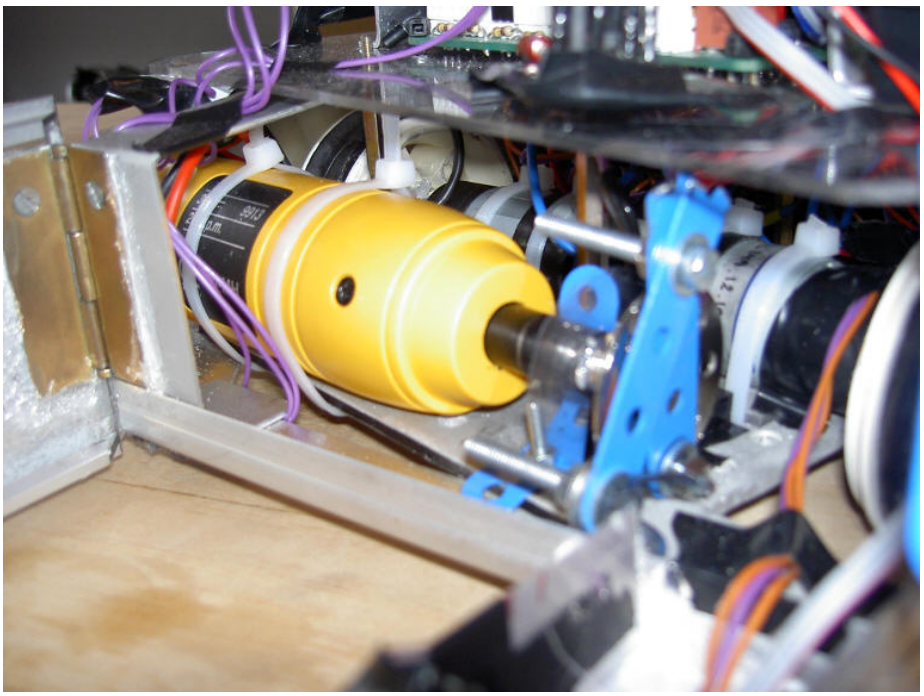


Even more cables were hidden inside the robot.

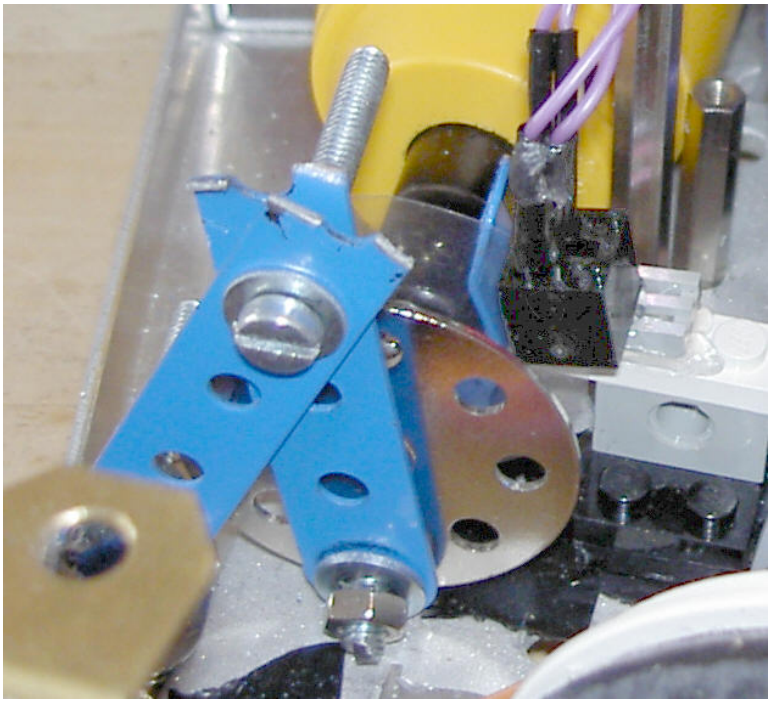


The robot seen from below. The white cable that is seen was used to hold the screwdriver and both motors in place. Note that furniture cloths were used to stabilize the robot without giving to much friction. Also note the three reflex sensors in the picture (things with cables soldered to them).

2D1426, Robotics and Autonomous Systems
Project report
Fredrik Ahlgren
Simon Gustafsson
Daniel Wardzynski



The shooting mechanism: The screwdriver, the encoder disc and the rotating disc, and the linkage mechanism.



The shooting mechanism: Encoder disc and slotted opto detector used to stop the mechanism in its initial position can be seen behind the rotating disc and linkages.

13 References

[1] Movie of last year robot competition
http://www.nada.kth.se/kurser/kth/2D1426/RAScompMay03.avi]

[2] Anders Conradi, Anders Dahl, Per-Johnny Käck, Joel Sjöstrand, *Robotics and Autonomous Systems - Project Report*, May 2003
http://www.nada.kth.se/kurser/kth/2D1426/reports2003/FejkOpaque.pdf

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

[4] Mattias Bratt, *Robotics and autonomous systems - Course notes III, the project,* Computational Vision an Active
Perception Laboratory, KTH, 2003.
http://www.nada.kth.se/kurser/kth/2D1426/L3_2003b.pdf

[5] HI-TECH Software. *HI-TECH PIC C Manual - Features and Runtime Environment*.
Excerpt from the full manual.

[6] Microchip Technology Inc. *MPLAB User's Guide*, 1998.

[7] PIC16F873/4/6/7 Datasheet
http://www.microchip.com/download/lit/pline/picmicro/families/16f87x/30292c.pdf

[8] H.R. Everett, Sensors for mobile robots, theory & application, 1995

[9] ELITR 8010 Datasheet
http://www.elfa.se/pdf/75/07534571.pdf

[10] ELITR 8307 Datasheet
http://www.elfa.se/pdf/75/07534472.pdf

[11] TSL261 Datasheet
http://www.cs.cmu.edu/~RAS/old/DataSheets/tsl260.pdf

14 Source code (roughly all *.h + demo.c)

14.1 DEMO.C

```
#include <pic.h>
#include <conio.h>
#include "defines.h"
#include "lcd.h"
#include "adc.h"
#include "button.h"
#include "sprint.h"
#include "debug.h"
#include "ir_comm.h"
#include "eeprom.h"
#include "pwm.h"
#include "intr.h"
#include "serialio.h"

extern char sound_index;

#define abs(x) ((x)>0?(x):-(x))
#define notnull(x) ((x)==0?(1):(x))
#define GOT_STUCK_LEFT (passed(move_time_left+1000))
#define GOT_STUCK_RIGHT (passed(move_time_right+1000))
#define GOT_STUCK (GOT_STUCK_LEFT&&GOT_STUCK_RIGHT)
#define GOT_PUCK (IR_GOTPUCK_READ > 4800)
#define SEE_PUCK
((IR_FRONT_LEFT_READ+IR_FRONT_RIGHT_READ+IR_REAR_RIGHT_READ
+IR_REAR_LEFT_READ)>5)
#define SUM_OFFGOAL
(IR_FRONT_RIGHT_OFFGOAL_READ+IR_FRONT_LEFT_OFFGOAL_READ+IR_
REAR_RIGHT_OFFGOAL_READ+IR_REAR_LEFT_OFFGOAL_READ)
#define SUM_DEFGOAL
(IR_FRONT_RIGHT_DEFGOAL_READ+IR_FRONT_LEFT_DEFGOAL_READ+IR_
REAR_RIGHT_DEFGOAL_READ+IR_REAR_LEFT_DEFGOAL_READ)

#define REFLEX_FRONT_STICK_BLACK
(sample_ad_channel(REFLEX_FRONT_STICK_CHANNEL) > 240)
```

```
#define REFLEX_FRONT_OTHER_BLACK
(sample_ad_channel(REFLEX_FRONT_STICK_CHANNEL) > 254)
#define REFLEX_REAR_BLACK
(sample_ad_channel(REFLEX_REAR_CHANNEL) > 240)

#define IR_FRONT_LEFT 0
#define IR_FRONT_RIGHT 3
#define IR_REAR_LEFT 1
#define IR_REAR_RIGHT 2

#define REFLEX_FRONT_STICK_CHANNEL 0
#define REFLEX_REAR_CHANNEL 1
#define REFLEX_FRONT_OTHER_CHANNEL 2

//Motor speed constant
#define MOTORSPEED 117


// 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 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, 0xF0,40, 0};

const char wagner[] = {0x02, 20, 0x07, 20, 0xF0, 20, 0x02, 20,
              0x07, 20,0x0B, 40, 0xF0, 20, 0x07, 40,
              0xF0, 20, 0x0B, 20,0xF0, 20, 0x07, 20,
              0x0B, 20, 0x12, 40, 0xF0, 20,0x0B, 40,
              0xF0, 20, 0x12, 20, 0xF0, 20, 0x0B,
            20,0x12, 20, 0x16, 40, 0xF0, 20, 0x06,
              40, 0xF0, 20,0x0B, 20, 0xF0, 20, 0x06,
              20, 0x0B, 20, 0x13, 80,0, 0};
```

```
bank2 static int
IR_FRONT_LEFT_OFFGOAL_READ,IR_FRONT_RIGHT_OFFGOAL_READ;
bank2 static int IR_REAR_LEFT_OFFGOAL_READ,
IR_REAR_RIGHT_OFFGOAL_READ;
bank2 static int
IR_FRONT_LEFT_DEFGOAL_READ,IR_FRONT_RIGHT_DEFGOAL_READ;
bank2 static int IR_REAR_LEFT_DEFGOAL_READ,
IR_REAR_RIGHT_DEFGOAL_READ;
bank2 static int IR_FRONT_LEFT_READ,IR_FRONT_RIGHT_READ;
bank2 static int IR_REAR_LEFT_READ,
IR_REAR_RIGHT_READ,IR_GOTPUCK_READ;
bank2 static int Motor_front,Motor_rear,Front_or_rear;
bank2 static int Motorspeed_left,Motorspeed_right;

//buffer for lcd printing
static char buffer[7];

//time variable for timing needs
bank1 static unsigned int sometime;

void read_ir();
void go_to_puck();
void go_to_offgoal();
void go_to_defgoal();
void get_free();
void test_motors();
void test_ad();
void test_enc();
void test_score();
void score_run();
void score_run_fast();
void test_ir();
void go_back();
void back_home();
void avoid_penalty();

main()
{
// Default bank (if bank keyword is not used) is 0.
// All function params and non-static local vars always go in
// bank 0.
```

```
// Therefore, try to use one of the other banks when possible
// to free memory space in bank 0.
// As an example, the only reason to make the vars declared below
// static is that it makes it possible not to put them in bank 0.
    bank2 static char i;
    bank2 static char sensor; //currently displayed IR sensor
    bank2 static char sample;
    bank2 static char tmp;

    //use five A/D channel (AN0-AN4) (see adc.h)
    ADCON1=0b00000010;TRISA0=TRISA1=TRISA2=TRISA3=TRISA5=1;

    ir_init();              //initialise IR co-processor communication
    init_buttons();   //initialise buttons
    init_pwm();             //initialise pwm for motor control
    init_uart();        //initialise serial I/O
    init_soft_tmr();  //initialise software timer
    init_sound();           //initialise sound generator
   init_enc();              //initialise optical encoders

    RB4=0;TRISB4=0; //Initialise signals for shooting mechanism

  // 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;

    init_lcd();                                         //initialise lcd

    play_tune(wagner);   // play 'wagner' sound at start up

    /*****************************************************
    *Testing functions for simple testing of robot components
    *****************************************************/
    //test_motors();                    //testing engines
    //for(;;) test_ad();    //Testing AD converters with sensors
```

```
//test_enc();                                   //Testing optical encoders
//test_score();
//test_ir();                                    //Testing IR sensors
//test_wll2();
//for(;;)back_home();



/*******************************************************
                                The main loop
*******************************************************/
for(;;)
{
      //synchronise to IR reception
      ir_wait();

      // Read all the IR values
      read_ir();

      //clear display
      lcd_print1("               ",0);
      lcd_print2("               ",0);

      //If Robot got puck
      if (GOT_PUCK)
      {
            //if close to goal and goal straight ahead
            if(((IR_FRONT_RIGHT_OFFGOAL_READ>50)
            &&(IR_FRONT_LEFT_OFFGOAL_READ>50)))
            {
                        //shoot puck;
                        score_run_fast();
                        go_back();
            }
            else
            {
                        //if on right side of ice and over black.
                        if(((IR_FRONT_RIGHT_OFFGOAL_READ

+IR_FRONT_LEFT_OFFGOAL_READ)>80)
```

```
                &&( ((REFLEX_FRONT_STICK_BLACK )
                || (REFLEX_FRONT_OTHER_BLACK))
                    && (abs(SUM_DEFGOAL-SUM_OFFGOAL)>80)))
                                        score_run(); //shoot puck
                    }

                    //got to offgoal to make goal
                    go_to_offgoal();
                    lcd_print1("Going to offgoal!",0);
                }
                else //if don't have puck
                {
                    if(SEE_PUCK)
                    {
                            //go get puck
                            go_to_puck();
                    }
                    else
                    {
                            go_to_defgoal();
                            lcd_print1("Don't see puck!",0);
                    }
                }

                //If stuck
                if (GOT_STUCK)
                {
                    lcd_print2("STUCK!!!",0);
                    get_free();
                }
        //Make sure you are not in black zone
        avoid_penalty();
        }
}



/*******************************************************
Function: go_to_puck
Arguments: void
Return value: void
```

2D1426, Robotics and Autonomous Systems
Project report
Fredrik Ahlgren
Simon Gustafsson
Daniel Wardzynski

```
Description: Makes the robot run to the puck
*******************************************************/
void go_to_puck()
{
	//calculate turning factor if puck is in front of robot
Motor_front=(15*(IR_FRONT_LEFT_READ-IR_FRONT_RIGHT_READ))
/(IR_FRONT_LEFT_READ+IR_FRONT_RIGHT_READ);

	//Special cases to avoid puck getting stuck at robot side
	if(Motor_front>11)
		Motor_front=Motor_front+10;
	if(abs(Motor_front)>10)
		Motor_front=Motor_front*2.5;

	//Turn full speed if puck behind robot
	if((IR_REAR_RIGHT_READ)>(IR_REAR_LEFT_READ) )
		Motor_rear=127;
	else
		Motor_rear=-127;

	//Set correct speed depending on if puck is in front of or behind robot
	if( ((IR_REAR_RIGHT_READ+IR_REAR_LEFT_READ)
		>(IR_FRONT_RIGHT_READ+IR_FRONT_LEFT_READ))

	||(IR_REAR_RIGHT_READ+IR_REAR_LEFT_READ+IR_FRONT_LEFT_RE
AD
		+IR_FRONT_RIGHT_READ)<100)
	{
		Motorspeed_left=Motor_rear;
		Motorspeed_right=(-1*Motor_rear);
	}
	else
	{
		Motorspeed_left=MOTORSPEED-Motor_front;
		Motorspeed_right=MOTORSPEED+Motor_front;
	}

	//For safety make sure not to oveflow data type
	if(Motorspeed_left< -127) Motorspeed_left=-127;
	if(Motorspeed_left>127) Motorspeed_left=127;
	if(Motorspeed_right< -127) Motorspeed_right=-127;
```

```
	if(Motorspeed_right>127) Motorspeed_right=127;

	//Run motors
	motors(Motorspeed_left,Motorspeed_right);
}


/*******************************************************
Function: go_to_offgoal
Arguments: void
Return values: void
Description: Makes the robot go to offensive goal
*******************************************************/
void go_to_offgoal()
{
	//calculate turning factor if puck is in front of robot
	Motor_front=(25*(IR_FRONT_LEFT_OFFGOAL_READ-
IR_FRONT_RIGHT_OFFGOAL_READ))

	/(IR_FRONT_LEFT_OFFGOAL_READ+IR_FRONT_RIGHT_OFFGOAL_REA
D);

	//Turn full speed if puck behind robot
	if((IR_REAR_RIGHT_OFFGOAL_READ)>(IR_REAR_LEFT_OFFGOAL_REA
D) )
		Motor_rear=120;
	else
		Motor_rear=120;


	//Set correct speed depending on if puck is in front of or behind robot
	if((IR_REAR_RIGHT_OFFGOAL_READ+IR_REAR_LEFT_OFFGOAL_READ)

	>(IR_FRONT_RIGHT_OFFGOAL_READ+IR_FRONT_LEFT_OFFGOAL_REA
D) )
	{
		Motorspeed_left=Motor_rear;
		Motorspeed_right=(-1*Motor_rear);
		turnspeed=Motor_rear;

	}
```

```
        else
        {
            Motorspeed_left=127-2*Motor_front;
            Motorspeed_right=127+2*Motor_front;
            turnspeed=-Motor_front;
        }

        //For safety make sure not to oveflow data type
        if(Motorspeed_left< -127) Motorspeed_left=-127;
        if(Motorspeed_left>127) Motorspeed_left=127;
        if(Motorspeed_right< -127) Motorspeed_right=-127;
        if(Motorspeed_right>127) Motorspeed_right=127;

        //Run motors
        motors(Motorspeed_left,Motorspeed_right);
    }

    /*******************************************************
    Function: read_ir
    Arguments: void
    Return values: void
    Description: Reads IR values into variables
    *******************************************************/
    void read_ir()
    {
        //Read IR values for puck
        IR_FRONT_RIGHT_READ=notnull(ir_value(IR_FRONT_RIGHT,PUCK));
        IR_FRONT_LEFT_READ=notnull(ir_value(IR_FRONT_LEFT,PUCK));
        IR_REAR_RIGHT_READ=notnull(ir_value(IR_REAR_RIGHT,PUCK));
        IR_REAR_LEFT_READ=notnull(ir_value(IR_REAR_LEFT,PUCK));

        //Read IR values for offensive goal
        IR_FRONT_RIGHT_OFFGOAL_READ=notnull(ir_value(IR_FRONT_RIGHT
    ,OFF_GOAL));
        IR_FRONT_LEFT_OFFGOAL_READ=notnull(ir_value(IR_FRONT_LEFT,OF
    F_GOAL));
        IR_REAR_RIGHT_OFFGOAL_READ=notnull(ir_value(IR_REAR_RIGHT,OF
    F_GOAL));
        IR_REAR_LEFT_OFFGOAL_READ=notnull(ir_value(IR_REAR_LEFT,OFF_
    GOAL));
```

```
	//Read IR values for defensive goal
	IR_FRONT_RIGHT_DEFGOAL_READ=notnull(ir_value(IR_FRONT_RIGHT
,DEF_GOAL));
	IR_FRONT_LEFT_DEFGOAL_READ=notnull(ir_value(IR_FRONT_LEFT,DE
F_GOAL));
	IR_REAR_RIGHT_DEFGOAL_READ=notnull(ir_value(IR_REAR_RIGHT,DE
F_GOAL));
	IR_REAR_LEFT_DEFGOAL_READ=notnull(ir_value(IR_REAR_LEFT,DEF_
GOAL));

	//Read stick IR sensor
	IR_GOTPUCK_READ=(ir_value(4,PUCK));
}

/*******************************************************
Function: test_score
Arguments: Void
Return values: Void
Description: Test to score without shooting mechanism
*******************************************************/
void test_score()
{
		lcd_print1("Scoring: start",0);
		motors(0,0);
		sometime=soft_time();
		for(;;) if(passed(sometime+700)) break;

		enc1_pos=enc2_pos=156;

		lcd_print1("Score: left rev",0);
		motors(-117,0);
		while (enc2_pos >93) {};

		motors(0,0);

		lcd_print1("Score: right rev",0);
		motors(0,-117);
		while (enc1_pos >93) {};

		motors(0,0);
		lcd_print1("Score: standing",0);
```

```
        sometime=soft_time();
        for(;;) if(passed(sometime+300)) break;

        enc1_pos=enc2_pos=193;
        lcd_print1("Score: forward",0);
        motors(127,127);
        while (enc1_pos > 10) {};

        motors(-117,-117);
        lcd_print1("Score: complete",0);
        play_tune(klingon);

        while(enc1_pos <193) {};
        motors(0,0);
        lcd_print1("Score: returning",0);
}

/*********************************************************
Function test_motors
Arguments: Void
Return values: Void
Description: Runs Booth engines 2 sec then Left engine 2 sec
             then right engine 2 sec.
*********************************************************/
void test_motors()
{
    lcd_print1("Both forward    ",0);
    motors(100,100);
    sometime=soft_time();
    for(;;) if(passed(sometime+2000)) break;

    lcd_print1("left forward   ",0);
    motors(100,0);
    sometime=soft_time();
    for(;;) if(passed(sometime+2000)) break;

    lcd_print1("right forward   ",0);
    motors(0,100);
    sometime=soft_time();
    for(;;) if(passed(sometime+2000)) break;
}
```

2D1426, Robotics and Autonomous Systems
Project report
Fredrik Ahlgren
Simon Gustafsson
Daniel Wardzynski

```
/*********************************************************
Function: get_free
Arguments: Void
Return values: Void
Description: Turn robot to get free if stuck
*********************************************************/
void get_free()
{
	sometime=soft_time();

	//if got puck when stuck
	if(GOT_PUCK)
	{
		motors(50,-127);
		enc1_pos=130;
		while (enc1_pos > 100)
			if(passed(sometime+1000))break; //use timeout to not get
stuck

	}
	else
	{
		if((IR_REAR_LEFT_READ+IR_FRONT_LEFT_READ)
			>(IR_REAR_RIGHT_READ+IR_FRONT_RIGHT_READ))
		{
			motors(-127,0);
			enc2_pos=130;
			while (enc2_pos > 100)
					if(passed(sometime+1000))break; //Use timeout
to not get stuck
		}
		else
		{
			motors(0,-127);
			enc1_pos=130;
			while (enc1_pos > 100)
					if(passed(sometime+1000))break;} //Use timeout
to not get stuck
		}
```

```
        }
        motors(0,0);
}

/*******************************************************
Function: test_enc
Arguments: Void
Return values: Void
Description: Test and display encoder values
*******************************************************/
void test_enc()
{
        sprint16(buffer,enc1_pos);
        lcd_print2(buffer,0);
        lcd_print_at_cursor("        ");// spaces to make sure previous value is
covered

        buffer[1]=RA4?'x':'o';
        buffer[0]=RE0?'x':'o';
        buffer[3]=RE1?'x':'o';
        buffer[2]=RE2?'x':'o';
        buffer[4]=0;

        buffer[0]=enc1_err?'X':'_';
        buffer[1]=' ';
        buffer[2]=enc2_err?'X':'_';
        buffer[3]=0;

        sprint16(buffer,enc2_pos);
        lcd_print2(buffer,8);
        lcd_print_at_cursor("        ");// spaces to make sure previous value is
covered

        sometime=soft_time();
        for(;;) if(passed(sometime+50)) break;
}

/*******************************************************
Function: test_ad
Arguments: Void
Return value: Void
```

```
Description: Test and display a/d values
**********************************************************/
void test_ad()
{
	unsigned char i;
	bank2 static char sample;
	bank1 static unsigned int sometime;

	//use all eight A/D channel (AN0-AN7) (see adc.h)

ADCON1=0b00000000;TRISA0=TRISA1=TRISA2=TRISA3=TRISA5=TRISE0=
TRISE1=TRISE2=1;

	clear_disp(); //clear LCD

	for(i=0; i<8;i++)
	{
		sample=sample_ad_channel(i);

		//convert value to string
		sprint16(buffer,sample);

		if(i*4<=12)
			lcd_print1("",i*4);
		else
			lcd_print2("",(i-4)*4);
		lcd_print_at_cursor(buffer);
	}
}

/*********************************************************
Function: score_run
Arguments: Void
Return values: Void
Description: Make goal when close to goal
**********************************************************/
void score_run()
{
	enc1_pos=120;
	RB4=1;  //start shooting engine
	sometime=soft_time();
```

```
motors(20,-120);
for(;;)
{
    if(passed(sometime+100)) break;
    //Break enough to not enter goal zone
    if(enc1_pos<120)
        motors(0,0);
    else
        motors(20,-100);

    for(;;)
    {
        if(passed(sometime+150)) break;

        //Break enough to not enter goal zone
        if(enc1_pos<124)
                motors(50,50);
        else
                motors(-127,-127);
    }
    for(;;)
    {
        if(passed(sometime+250)) break;

        //Break enough to not enter goal zone
        if(enc1_pos<127)
                motors(127,127);
        else
                motors(-127,-127);
    }

    for(;sample_ad_channel(2)>100;) ;

    //Play goal tune
    play_tune(wagner);

    //Stop shooting engine
    RB4=0;

    back_home();    //Reverse against centre field
```

```
            sometime=soft_time();
            for(;;)
            {
                back_home();
                avoid_penalty();
                if(passed(sometime+500)) break;
            }

            //Bring shooting mechanism to start position
            for(;sample_ad_channel(2)<105;)
            {
                RB4=1;
                sometime=soft_time();
                for(;sample_ad_channel(2)<105;) if(passed(sometime+2))
break;
                RB4=0;
                sometime=soft_time();
                for(;sample_ad_channel(2)<105;) if(passed(sometime+7))
break;
                back_home();            //back at same time
                avoid_penalty();        //don't back into defensive goal

            }
}

/*******************************************************
Function: score_run_fast
Arguments: Void
Return values: Void
Description: Make goal when not close to goal
*******************************************************/
void score_run_fast()
{
    RB4=1;
    sometime=soft_time();
    for(;;)if(passed(sometime+250)) break;
    motors(-127,-127);
    for(;sample_ad_channel(2)>100;) ;

    //Play score tune
    play_tune(wagner);
```

```
        RB4=0;
        back_home(); //go back to centre

        sometime=soft_time();
        for(;;)
        {
            back_home();
            avoid_penalty();
            if(passed(sometime+500)) break;
        }

        //Bring shooting mechanism to start position
        for(;sample_ad_channel(2)<105;)
        {
            RB4=1;
            sometime=soft_time();
            for(;sample_ad_channel(2)<105;) if(passed(sometime+2)) break;
            RB4=0;
            sometime=soft_time();
            for(;sample_ad_channel(2)<105;) if(passed(sometime+7)) break;
            back_home();
            avoid_penalty();
        }
    }

    /*******************************************************
    Function: lab0
    Arguments: Void
    Return values: Void
    Description: The code for Lab0
                (Go forward if light else spin and play)
    *******************************************************/
    void lab0()
    {
        //Lots of unmodulated light
        if(ir_value(sensor,UNMODULATED) > 20)
            motors(70,70);
        else
        {
            motors(100,-100);
```

```
            if(sound_index == 255)
                play_tune(blues_notes);
        }
    }


    /*******************************************************
    Function: test_ir
    Arguments: Void
    Return values: Void
    Description: Display IR values for puck
    *******************************************************/
    void test_ir()
    {
        for(;;)
        {
            read_ir();
            sprint16(buffer,IR_FRONT_LEFT_READ);
            lcd_print1(buffer,0);
            lcd_print_at_cursor("       ");

            sprint16(buffer,IR_FRONT_RIGHT_READ);
            lcd_print1(buffer,8);
            lcd_print_at_cursor("       ");

            sprint16(buffer,IR_REAR_LEFT_READ);
            lcd_print2(buffer,0);
            lcd_print_at_cursor("       ");

            sprint16(buffer,IR_REAR_RIGHT_READ);
            lcd_print2(buffer,8);
            lcd_print_at_cursor("       ");
        }
    }


    /*******************************************************
    Function: avoid_penalty
    Arguments: Void
    Return values: Void
    Description: If detects goal zone, run in other direction
```

```
**********************************************************/
void avoid_penalty()
{
      bank2 static unsigned int avtime;
      avtime=soft_time();

      //Don't care about centre line
      if(abs(SUM_DEFGOAL-SUM_OFFGOAL)>80 )
      {
            if(REFLEX_REAR_BLACK)
            {
                  motors(127,127);
                  while(!passed(avtime+300));
            }

      if((REFLEX_FRONT_STICK_BLACK)||(REFLEX_FRONT_OTHER_BLACK) )
            {
                  motors(-127,-127);
                  while(!passed(avtime+300));
            }
      }
}

/*********************************************************
Function: go_back
Arguments: Void
Return values: Void
Description: Back away for 1 sec
**********************************************************/
void go_back()
{
      sometime=soft_time();
      back_home();
      while(!passed(sometime+1000));
      motors(0,0);
}

/*********************************************************
Function: go_to_defgoal
Arguments: Void
Return values: Void
```

2D1426, Robotics and Autonomous Systems
Project report
Fredrik Ahlgren
Simon Gustafsson
Daniel Wardzynski

```
Description: Makes robot go to defensive goal
**************************************************/
void go_to_defgoal()
{
	//calculate engine values when puck is in front of robot
	Motor_front=(10*(IR_FRONT_LEFT_DEFGOAL_READ-
IR_FRONT_RIGHT_DEFGOAL_READ))

	/(IR_FRONT_LEFT_DEFGOAL_READ+IR_FRONT_RIGHT_DEFGOAL_REA
D);

	//Turn full speed if puck is behind robot
	if((IR_REAR_RIGHT_DEFGOAL_READ)>(IR_REAR_LEFT_DEFGOAL_REA
D) )
		Motor_rear=127;
	else
		Motor_rear=127;
	//Set motor speed pending on where puck is
	if((IR_REAR_RIGHT_DEFGOAL_READ+IR_REAR_LEFT_DEFGOAL_READ)

	>(IR_FRONT_RIGHT_DEFGOAL_READ+IR_FRONT_LEFT_DEFGOAL_REA
D) )
	{
		Motorspeed_left=Motor_rear;
		Motorspeed_right=(-1*Motor_rear);
	}
	else
	{
		Motorspeed_left=MOTORSPEED-Motor_front;
		Motorspeed_right=MOTORSPEED+Motor_front;
	}

	//Protect from data type overflow
	if(Motorspeed_left< -127) Motorspeed_left=-127;
	if(Motorspeed_left>127) Motorspeed_left=127;
	if(Motorspeed_right< -127) Motorspeed_right=-127;
	if(Motorspeed_right>127) Motorspeed_right=127;

	//Run engines
	motors(Motorspeed_left,Motorspeed_right);
}
```

2D1426, Robotics and Autonomous Systems
Project report
Fredrik Ahlgren
Simon Gustafsson
Daniel Wardzynski

```
/*******************************************************
Function: back_home()
Arguments: Void
Return values: Void
Description: Makes robot back against defensive goal
*******************************************************/
void back_home()
{
	read_ir();
	Motor_front=(20*(IR_REAR_LEFT_DEFGOAL_READ-
IR_REAR_RIGHT_DEFGOAL_READ))

	/(IR_REAR_LEFT_DEFGOAL_READ+IR_REAR_RIGHT_DEFGOAL_READ);

	//Set motor speed
	Motorspeed_left=-MOTORSPEED+Motor_front;
	Motorspeed_right=-MOTORSPEED-Motor_front;

	//Protect from data type overflow
	if(Motorspeed_left< -127) Motorspeed_left=-127;
	if(Motorspeed_left>127) Motorspeed_left=127;
	if(Motorspeed_right< -127) Motorspeed_right=-127;
	if(Motorspeed_right>127) Motorspeed_right=127;

	//Run engines
	motors(-127,-127);
}
```

14.2 intr.c

```
#include <pic.h>
#include "ir_comm.h"
#include "ser_intr.h"
#include "defines.h"

//definitions missing from pic.h
//static volatile bank1 bit          TRISE2          @
(unsigned)&TRISE*8+2;
//static volatile bank1 bit          TRISE1          @
(unsigned)&TRISE*8+1;
//static volatile bank1 bit          TRISE0          @
(unsigned)&TRISE*8+0;



//pins for controlling the speaker driver (use 6V motor driver IC)
//connect sound pin to dir and sound enbl pin to enbl
//with these settings connect:
//
// pin35 RB2 <---> fifth motor control pin
// pin36 RB3 <---> sixth motor control pin
//
// Then connect speaker with series resistance to motor terminal 3

#define SOUND_PIN RB3
#define SOUND_PIN_TRIS TRISB3
#define SOUND_ENBL_PIN RB2
#define SOUND_ENBL_PIN_TRIS TRISB2

// pins for reading encoder wheels
// note that the use of port E pins as digital I/O requires ADCON1 to be
properly set
#define ENC1A RA4
#define ENC1B RE0
#define ENC2A RE1
#define ENC2B RE2
```

```
#define ENC1A_TRIS TRISA4
#define ENC1B_TRIS TRISE0
#define ENC2A_TRIS TRISE1
#define ENC2B_TRIS TRISE2

//constant vector to give right frequencies for all notes
const char
wave_per_add[16]={0,14,28,41,53,64,75,85,95,104,112,120,128,135,142,14
8};

const char * notes;
char sound_index=255;
unsigned int note_time;

bit has_passed;
unsigned int soft_tmr;
unsigned int soft_tmr_copy;
unsigned char enc1_pos, enc2_pos;
static unsigned char enc1_state;   //stores current state in bits 0,1  old state
in bits 2,3
static unsigned char enc2_state;   //stores current state in bits 0,1  old state
in bits 2,3
static bit enc_started;
bit enc1_err;
bit enc2_err;
const char enc_vals[16]={0,1,-1,2,  -1,0,2,1,  1,2,0,-1,  2,-1,1,0};

unsigned char last_enc1_pos, last_enc2_pos;
unsigned int move_time_left;
unsigned int move_time_right;

//initialise encoders
void init_enc()
{
 ENC1A_TRIS=1;
 ENC1B_TRIS=1;
 ENC2A_TRIS=1;
 ENC2B_TRIS=1;
 enc1_pos=enc2_pos=0;
 enc_started=0;
}
```

```
//initialise tone generator
void init_sound()
{
 SOUND_PIN_TRIS=0;
 SOUND_ENBL_PIN_TRIS=0;

 //init TIMER0
 T0CS=0;    //internal clock
 PSA=0;     //prescaler assigned to WDT
 T0IF=0;
 T0IE=1;
}

//start playing a tune
//(non-blocking)
void play_tune(const char* tune)
{
 sound_index=0;
 if(tune[sound_index]==0&&tune[sound_index+1]==0)
        sound_index=255;
 else
 {
        notes=tune;
        //copy two lowest bytes of soft_tmr
        note_time=soft_tmr;
 }
}
//Variable def to access both 8-bit regs of TIMER1 together as an int
volatile unsigned int TMR1 @ 0x0E;

// initialise the soft timer
// uses TIMER1
void init_soft_tmr()
{
 TMR1=60536;  //0xFFFF-4999 will overflow (reach beyond 0xFFFF) in 1 ms
 T1CON=0b00000001; //TMR1 on in internal clock mode without prescaler
 TMR1IE=1;
}

void interrupt service_routine(void)
```

```
{
 char tempval;
 char note;
 char prescale;
 //calling functions from the interrupt routine should be
 //avoided, so we include the code we need right here instead.
 //That way we can still use separate files.
 //First some declarations in *.irh files (interrupt routine headers)
 #include "ir_comm.irh"
 #include "serialio.irh"
 //then the code itself in *.irc files (interrupt routine C source)
 #include "ir_comm.irc"
 #include "serialio.irc"

 // update soft timer if appropriate
 if(TMR1IF)
 {
        TMR1IF=0;
        soft_tmr++;
        TMR1ON=0;
        TMR1+=60532;
        TMR1ON=1;

 // code to read two encoder wheels with two slotted opto detectors each
        // uses port E and RA4

        // read state of opto detectors
        if(ENC1A) bitset(enc1_state,0);
        if(ENC1B) bitset(enc1_state,1);
        if(ENC2A) bitset(enc2_state,0);
        if(ENC2B) bitset(enc2_state,1);

 //update encoder positions if this is not first time this code runs
        if(enc_started)
        {
              if((tempval=enc_vals[enc1_state])==2) enc1_err=1;
              else enc1_pos+=tempval;

              if((tempval=enc_vals[enc2_state])==2) enc2_err=1;
              else enc2_pos+=tempval;
        }
```

```
        // If the encoders are moving update movetime
        if(enc1_pos!=last_enc1_pos)
        {
            last_enc1_pos=enc1_pos;
            move_time_right=soft_tmr;
        }
        if(enc2_pos!=last_enc2_pos)
        {
            last_enc2_pos=enc2_pos;
            move_time_left=soft_tmr;
        }

        //prepare for next time
        enc_started=1;
        enc1_state<<=2;
        enc1_state&=0b00001111;
        enc2_state<<=2;
        enc2_state&=0b00001111;
}



// handle sound generation
if(T0IF)
{
        T0IF=0;
        if(sound_index!=255)
        {
            // a tune is playing

            // check if it is time for the next note
           If(((unsigned int)soft_tmr - (note_time+((unsigned Int)
                    notes[sound_index+1]<<3)))<0x8000)
            {
                    //time is up; next note
                    sound_index+=2;
                    //check end of tune
                    if(notes[sound_index]==0&&notes[sound_index+1]==0)
                    {
```

```
                                        sound_index=255;
                                        SOUND_ENBL_PIN=0;
                                        PS0=PS1=PS2=1;   // max
prescaling to avoid irrelevant interr
                        }
                        note_time=soft_tmr;
                }
                if(sound_index!=255)
                {
                        // tune is still playing
                        // continue to play note
                        note=notes[sound_index];
                        // if this is a pause (quiet note)
                        if((note&0xF0)!=0xF0)
                        {
                                        // not a pause
                                        // set prescaler depending on octave
0, 1, 2, or 3
                                        prescale=0b100-((note&0x30)>>4);
                                        OPTION|=prescale;
                                        OPTION&=(prescale|0b11111000);
                                        // set timer 0 to overflow when
sound pin needs
                                        // to be toggled next time (new
TMR0 interrupt then)
                                        TMR0+=wave_per_add[note&0x0F];
                                        // toggle sound pin and make sure
enable pin is high
 SOUND_PIN=!SOUND_PIN;
                                        SOUND_ENBL_PIN=1;
                        }
                        else SOUND_PIN=SOUND_ENBL_PIN=0;   //pause
                }
        }
 }
}
```

14.3 adc.h

```
//sample an A/D channel and return the eight most
//significant bits (the two least significand can be
//fetched from ADRESL as its bit 6 and  7)
//
//This function includes a delay for the required
//acquisition time (enough for source impedances up
//to 10kOhms) and then waits for the converion to
//complete. All in all a call will take about 40 us.
//
//(waiting for the acquisition time is really a waste
//of time if the same channel is sampled again and
//not immediately after the previous conversion)
//
//Do not forget to configure which pins you will use
//for A/D in ADCON1 and TRISA (and maybe TRISE too)
//once before first using this function
//Example (if you want only one A/D channel):
//ADCON1=0b00001110;TRISA0=1;
//Example (if you want 5 A/D channels):
//ADCON1=0b00000010;TRISA0=TRISA1=TRISA2=TRISA3=TRISA5=1;
//See fig 11-2 in data sheet
//
//The pin each channel uses is marked on the front page
//of the data sheet as ANx, wher x is the channel number
extern char sample_ad_channel(char channel);
```

14.4 button.h

```
// button debouncing function
//  generates button events if called
//  in main loop and maintains the
//  debounced state of the button
//  possible events are:
//  PRESSED - generated onxce when button
//    goes down. For up and down buttons
//    also repeatedly after HOLD_DELAY with
//    period REP_DELAY
//  HOLD - generated once when button has been
//    down for HOLD_DELAY
//  QRELEASE - generated once when (and if) a
//    button is released before HOLD_DELAY has
//    passed
//        the events and state of each button are stored
//  as bits in the buttons[button_number] global char

void handle_button(char button);          //button must be = 0..3

//clear all button events
extern void clear_button_events();


//initialisation function (see button.c for changing the I/O pins)
extern void init_buttons();


//bit defs for button bytes (one byte per button)
#define IS_DOWN 0                         // debounced state of button
#define PRESSED 1                         // pressed event
#define HOLD 2                            // hold event (after down for
HOLD_DELAY)
#define QRELEASE 3                        // release event (released before
hold event would have occured)
// about button events:
//   each variable indicates whether the corresponding
//   event has occured since the last call of
//   clear_button_events()
```

```
//index defs for buttons[] array
#define MENUB 0                 // menu button byte
#define DOWNB 1                 // down button byte
#define UPB 2                                   // up button byte
#define CANCELB 3               // cancel button byte


// global vars
extern bank2 char buttons[4];           //holds button events, see defs
above

// example:
// to see if the cancel button has been pushed down since last call of
// clear_button_events() use:
// if(bittst(buttons[CANCELB],PRESSED)) do_something();
```

14.5 sprint.h

```
// print an int to buf in decimal format
// buf has to be in bank 0 or 1 or strange
// 'fixup overflow' errors will occur at
// link time
extern void sprint16(char * buf, int num);


//print a char to buf in hex format
// buf has to be in bank 0 or 1 or strange
// 'fixup overflow' errors will occur at
// link time
extern void sprint8X(char * buf, char num);
```

14.6 debug.h

```
//to use this simple debugging tool you have to know the address of every
//variable to be monitored. it can show the contents of every byte in bank
//0, 1, 2, and 3 and in eeprom. integers etc can only be viewed as separate
//bytes.

//a call to this function is blocking and stops all other program execution
//including the interrupt routine
//it is used to view the contents of the file registers and the EEPROM
//it uses three address ranges (use menu button to cheange between them):
//B01 - addresses 0 to 255 in banks 0 and 1
//B23 - addresses 256 to 511 in banks 2 and 3 (the display will still show 0 to
255)
//EEP - addresses 0 to 255 in the EEPROM of the device
//Program FLASH can not be read
//The address shown is changed with the up and down buttons
//The format of the data can be changed between
//hexadecimal, decimal, binary and ascii character modes
//by pressing the menu button
extern void debug();
```

14.7 defines.h

```
// Macros for bit manipulation and testing
#define bitset(var,bitno) ((var)|=1 << (bitno))
#define bitclr(var,bitno) ((var)&=~(1<<(bitno)))
#define bittst(var,bitno) (((var)&(1<<(bitno)))!=0)

//macros for manipulating high and low byte of an int variable
#define lo_byte(int_var) (*((char *)(&(int_var))))
#define hi_byte(int_var) (*(( (char *)(&(int_var)) )+1  )) /*works only for
bank0 vars*/
#define hi_byte2(int_var) (*(( (bank2 char *)(&(int_var)) )+1  )) /*works
only for bank2 vars*/
```

14.8 eeprom.h

```
//read a byte from EEPROM
extern char eeprom_read(char ee_adr);


// write a byte to EEPROM
// tries MAX_WRITE_TRIES times
// and returns 1 on success, 0 on failure
// each write try takes about 4 ms during which all program
// execution stops (including interrupts)
extern bit write_eebyte (char data,char addr);
```

14.9 intr.h

```
extern unsigned int move_time_left;
extern unsigned int move_time_right;

extern bit has_passed;
extern unsigned int soft_tmr;
extern unsigned int soft_tmr_copy;

// Macro to read the soft timer atomically
// Use only this macro to read the time outside the interrupt routine
// USE passed(time_to_check) MACRO INSTEAD FOR COMPARISONS!!!
#define soft_time() (di(),soft_tmr_copy=soft_tmr,ei(),soft_tmr_copy)

// Macro to check if a time is in the past (or present) atomically
// Use only this macro to compare to current time (outside the
// interrupt routine).
// Returns 1 if time_to_check has been passed. Because the time vars
// are only 16 bits it must then return 0 again after some amount of
// time here chosen to be 1024 ms.
// Maximum time for delays is then 65535ms-1024ms or about 64.5 sec.
#define passed(time_to_check)\
 (di(),has_passed=((unsigned int)(soft_tmr-
(time_to_check))<1024),ei(),has_passed)

// Use above macros like this:
//              sometime=soft_time();
```

```
//              for(;;) {
//                              if(passed(sometime+60000)) break;
//              }
//
// The for loop will break after 60 seconds
// (64.5 seconds max as stated above)
// Of course yhe program would normally do other
// things than just executing a for loop while waiting


// initialise the soft timer
// uses TIMER1
extern void init_soft_tmr();


//initialise tone generator
// uses TIMER0
extern void init_sound();

//start playing a tune
// (non-blocking)
// tune is an array of chars in the following format:
// {note, time, note, time, .... ,0,0}
// i.e. it contains notes and their duartion, and always
// ends with two zeros
// format of note:
// bits 0-3 is the number of half tones from the base tone in each octave
// (so that if C was the base tone it would be 0, D=2, E=4, F=5, G=7,
// A=9, and H=11; cf counting the keys of a piano, including the black ones)
// bits 4-5 is the octave number from 0 to 3
// if bits 4-7 are all 1 the note will be quiet, i.e. a pause
// format of time:
// duration of note in units of 8 ms, so that 1 s is 125 units.
// Two equal notes (not necessarily with the same duration)
// will sound like merged to one longer note. Insert a small pause
// to avoid this (and subtract the time of the pause (one unit?)
// from that of the first of the equal notes
//
// a call will abort any already playing tune and start playing the new tune
// (i.e. playing the empty tune {0,0} will just abort previous tunes)
extern void play_tune(const char* tune);
```

```
// read the encoder positions from these vars
// (can also cleared from main program like "enc1_pos=0;")
extern unsigned char enc1_pos, enc2_pos;

// error bits that are set by interrupt if a pulse has been missed
extern bit enc1_err;
extern bit enc2_err;

// init function to call before using encoders
extern void init_enc();
```

14.10 ir_comm.h

```
// Define 2D array for storing received signal values
// One int per sensor and target
// First index is sensor number (0...4)
// Second index is target number (0-puck, 1-off_goal, 2-def_goal,
// 3-robot, 4-umodulated)
// in bank 0, 5 sensors, 5 16bit readings each
extern bank1 int target_map[5][5]; //don't read data from here; use
ir_value() macro

//used by ir_value macro below
extern unsigned int temp_ir_val;

// Macro to read an IR value without risking interference from the
// interrupt routine. Always use this to read IR values
// First index is sensor number (0...4)
// Second index is target number (0-puck, 1-off_goal, 2-def_goal,
// 3-opponent, 4-umodulated); use defines below
// Use like if(ir_value(0,PUCK)>THRESHOLD) do_something();
// #define ir_value(sens,targ) (di(),
temp_ir_val=target_map[sens][targ],ei(),temp_ir_val)
// Commented away beacause the risk of interrupt routine writing to the int
value
```

```
// being read just between reading the two bytes is very small and the results
not grave.
// Using it with several times in one expression (e.g. comparing to sensors)
also gives
// unexpected results. Therefore we
#define ir_value(sens,targ) target_map[sens][targ]
// instead, just for compatibility with old code
// To be absolutely safe against inteference from interrupts, use di() and ei()
around
// critical code segments (they had better be short though) or
// use wait_ir() in a way that ensures that the interrupt routine will not
modify
// the ir values being read when they are being read.

// target defs
#define PUCK 0
#define OFF_GOAL 1
#define DEF_GOAL 2
#define OPPONENT 3
#define UNMODULATED 4

// set one or more of bits 0..4 to request data for
// those sensors; LOOK AT ir_wait_for() FUNCTION
// BEFORE MODIFYING
extern bank2 char ir_requested;

// each bit indicates if the corresponding sensor has
// been received; used and cleared by ir_wait_for()
extern bank2 char ir_received;

// some useful flags handled by interrrupt routine
// (see below)
extern bank2 char spi_flags;

// bit defs for spi flags
// used by interrupt routine to distinguish two consecutive start bytes
#define GOT_1ST 0
// set on failure to receive one ten bit set of data for one sensor
// or rx overflow, cleared on reception of sensor 0
#define BAD_DATA 1
// set after rx overflow, cleared on reception of sensor 0
```

```
#define OVERFLOW 2
// 1 when receiving set of 12 bytes over SPI else 0
#define RECEIVING 3

//these variables control the servos connected to the co-processor
//usable range is -5500 to +12500 (though much less should be enough on
most servos)
//check to see that your extreme values does not damage the servo (it should
not
//keep buzzing at its end position if unloaded)
//servo0 is to the left on the motherbord (second three-pin header from the
dip switch)
extern int servopos[4]; //don't write to this one (reading is OK)

//use this macro to set servo positions
#define set_servo(num,pos) di(); servopos[num]=pos;ei();
//(read servo positions directly from servopos array)


// init SSP module in SPI mode
// I/O pins used RC3, RC4, RC5 (TRISC bits 3, 4, 5 are handled here)
extern void ir_init();

// call this function in the main loop to keep it synchronised
// to the arrival of new IR measurements from the co-processor
// (if desired).
//
// Waits for a new reception of all sensors to complete
// The argument is the sensor number to wait for, which becomes
// the last one befor the waiting is over and ir_wait returns.
// If all sensors are read (set by ir_requested declared above)
// use e.g. 4 to read sensors 0-4 and then return.
// BE SURE NOT TO WAIT FOR A SENSOR THAT HAS NOT BEEN REQUESTED
// THROUGH ir_requested!
extern void ir_wait_for(char sensor);

//as a default this macro can be used, which waits for sensor 4
//THIS CAN'T BE USED IF SENSOR 4 IS NOT REQUESTED THROUGH
ir_requested!
#define ir_wait() ir_wait_for(4)
```

14.11 lcd.h

```
//macro to power the lcd on and off
#define set_lcd_power(on) (RD4=on)

//get the lcd going after power-up
//the lcd uses all of port D
extern void init_lcd();

//functions to write to the LCD:

//print to display mem line 1 at position pos
//str is a null terminated array of chars
extern void lcd_print1(const char *str,char pos);

//print to display mem line 2 at position pos
//str is a null terminated array of chars
extern void lcd_print2(const char *str,char pos);

//print to display mem at the current cursor position on line 1 or 2
//str is a null terminated array of chars
extern void lcd_print_at_cursor(const char *str);

//clear the display
extern void clear_disp();
```

14.12 pwm.h

```
//initialise PWM
//uses both CCP pins (RC1 and RC2; connect these to enbl pins
//near the motor driver circuit) as well as
//the two motor direction control pins defined in pwm.c (connect
//these to dir pins near the motor driver circuit)
//see also end of this file
extern void init_pwm(void);

// Set speeds of motors (-128...+127). Negative values
// mean backward rotation.
extern void motors(signed char left, signed char right);
```

```
//In the default configuration with
//#define left_dir_pin RB0
//#define left_dir_tris TRISB0
//#define right_dir_pin RB1
//#define right_dir_tris TRISB1
//in pwm.c, make the following connections:
//
// pin17 CCP1  <---> first motor control pin (marked enbl)
// pin33 RB0   <---> second motor control pin (marked d)
// pin16 CCP2  <---> third motor control pin
// pin34  RB1   <---> fourh motor control pin
//
// left motor  <===> motor terminal marked 1 (two pins)
//right motor  <===> motor terminal marked 2 (two pins)
//
// then reverse the polarity of the motors by turning
// the connectors at the motor terminals if necessary
// until the motors run in the direction requested by the
// your program
```

14.13 ser_intr.h

```
//header file to be included only into serialio.c
//and the interrrupt routine source file

//settings for Windows Hyperterminal:
// chose 'connect using' COM2 if this
// is where the RS232 cable is connected
// and click configure
// choose speed defined by SPEED below
// 8 data bits, no parity, 1 stop bit,
// no flow control
// click OK and go to settings tab
// click ASCII setup
// check append line feeds... box
// click OK twice
```

```
// define USE_LEDS if debug LEDs on the pic
// are to be used
// BUT CHANGE THE IO PINS BELOW SO THAT THEY
// DO NOT CONFLICT WITH ANYTHING ELSE
//#define USE_LEDS

#ifdef USE_LEDS
// debug LEDs (only meaningfull if USE_LEDS is defined)
// These pins will be defined as outputs so don't
// ground them or tie them to Vcc.
// The TRIS definitions must match as below
#define RX_FULL RB3
#define RX_FULL_TRIS TRISB3
#define TX_FULL RB2
#define TX_FULL_TRIS TRISB2
#define RX_FERR RB0
#define RX_FERR_TRIS TRISB0
#endif

// set communication speed
// 9600, 38400, or 115200 bps
// (default is 9600)
#define SPEED 115200

// buffer sizes
// the buffers reside in BANK3
// (that has a total of 96 bytes),
// as do 4 bytes of pointers to
// the buffers. ==> max sum of these two is 92:
#define RX_SIZE 16
#define TX_SIZE 32

extern bank3 volatile char tx_buf[TX_SIZE];
extern bank3 volatile char rx_buf[RX_SIZE];
extern bank3 volatile char rx_in;
extern bank3 volatile char rx_out;
extern bank3 volatile char tx_in;
extern bank3 volatile char tx_out;
```

14.14 serialio.h

```
// For Windows HyperTerminal settings see ser_intr.h

// Include this file and conio.h to get access
// (conio.h MUST be included BEFORE serialio.h!!!)
// to putch(), getch, getche(), kbhit() and lookch()
// (see below for a description of the latter). Also
// include serial_intr.h, serialio.irc, serialio.irh
// in your interrupt routine (see instructions in
// those files).
// Before serial I/O will work you must also call
// the init_uart() initialisation routine and
// enable interrupts with PEIE=1 and ei().

// Calls to the above functions are all non-blocking,
// i.e. there is no waiting for something to become
// available for reading in the receive buffer, or
// for a free space in the transmit buffer. Trying to
// read an empty receive buffer will simply return
// zero, while a write to a full send buffer will be
// ignored (optionally it will also turn on a debugging
// LED).

// If the receive buffer is full when a character is
// received from the RS-232 line the interrupt routine
// will echo a '#' back to the host (and optionally
// light a LED).

// initialises serial I/O
// uses the TX and RX pins (RC6 and RC7)
// and, if USE_LEDS is defined in ser_intr.h
// three other pins defined in the same file
extern void init_uart();
// extra function to look at the next
// character to be read but leave it
// in the buffer to be read again
extern char lookch(void);
//print to serial buffer
//str is a null terminated array of chars extern void print(const char *str);
```