Robotics and Autonomous Systems
– Project Report

Figure 1: The robot FejkOpaque

Anders Conradi, d99-aco@nada.kth.se
Anders Dahl, d99-ada@nada.kth.se
Per-Johnny Kääck, d99-pka@nada.kth.se
Joel Sjöstrand, d99-jgs@nada.kth.se

May 26, 2003
Abstract

This report summarizes the development and characteristics of a small two-wheeled robot as part of the course Robotics and Autonomous Systems, 2D1426 held at the Royal Institute of Technology, KTH during the spring of 2003. The objective of the robot project was the participation in a robot floor hockey tournament at the end of the course. We describe the mechanical solutions that were chosen for the design, as well as the internal electronics. Moreover, the strategy and behaviour is addressed in detail, with C source code attached in an appendix. Future robot builders will undoubtedly find use of the concluding remarks and tips also to be found at the end of the document.
7.2.3 Co-processor (PIC16F876) ........................................ 17
7.2.4 Motor driver circuits (H-bridges) ............................... 17
7.2.5 Set of dip-switches ............................................. 17

8 Electronics .......................................................... 18
8.1 Electronic components and devices off the main board ........ 18
  8.1.1 Auxiliary electronic devices .................................. 18
  8.1.2 Environment manipulation devices .......................... 19
  8.1.3 Accessories .................................................. 20
8.2 Electricity supply ................................................ 20
  8.2.1 7.2 V NiCd accumulators ..................................... 20
  8.2.2 Voltage booster board ....................................... 21
  8.2.3 LED driver board ........................................... 21

9 Software implementation ........................................ 22
9.1 Program states .................................................. 22
  9.1.1 Goto Puck .................................................. 23
  9.1.2 Goto Goal .................................................. 23
  9.1.3 Fire ......................................................... 23
  9.1.4 Return to Center ........................................... 23
  9.1.5 Avoiding the defensive goal zone ......................... 23
  9.1.6 Stuck behaviours ......................................... 24
  9.1.7 Go Home .................................................. 24

10 Discussion .......................................................... 26
10.1 Results ........................................................ 26
10.2 Conclusions .................................................... 27
10.3 Tips for the future ........................................... 28

A Circuit diagrams and connections ................................ 29

B Pictures of the robot ............................................... 33

C Parts list .......................................................... 37
  C.1 Parts on main board .......................................... 37
  C.2 Electronic parts off main board .............................. 37
  C.3 Components for electricity supply ........................... 37
  C.4 Body parts .................................................... 38

D Source code .......................................................... 39
  D.1 adc.h .......................................................... 39
D.2 debug.h .................................................. 39
D.3 defines.h .................................................. 40
D.4 demo.c ..................................................... 40
D.5 intr.h ...................................................... 55
D.6 ir_comm.h ............................................... 57
D.7 lcd.h ...................................................... 59
D.8 pwm.h ...................................................... 59
D.9 sprint.h .................................................. 60

Bibliography .................................................. 61
List of Figures

1 The robot *FejkOpaque* ........................................... 1
1.1 An overview of the rink. ........................................... 6
3.1 The location of the various sensors of the robot. ............... 10
3.2 An example of a simple transparent encoder disc. .............. 12
9.1 The major states used in the program. .......................... 25
A.1 A circuit diagram of the motherboard used in the course. ...... 29
A.2 Signal circuit diagram ............................................. 31
A.3 Power supply diagram ............................................. 32
B.1 The vicious grin of the robot front. .............................. 33
B.2 The back of the robot. ............................................. 34
B.3 The robot seen from above. ...................................... 34
B.4 A study of the shooting mechanism. .............................. 35
B.5 An isometric view of the robot. ................................. 35
B.6 The lower of the two robot modules. ............................. 36
B.7 The two robot modules side by side. ............................ 36
Chapter 1

Introduction

1.1 Background

This project report is part of the undergraduate course in *Robotics and Autonomous Systems*, at the *Royal Institute of Technology*. The objective of this course is to teach the basics of robotics, and give an understanding for the complications involved in autonomous systems.

In the course the students are given the task to construct a quite simple autonomous robot being able to play floor hockey. The main parts of the robot are a PIC 16F877 microcontroller mounted on a motherboard, different kind of IR-sensors and two 12V motors. These parts as well as other material such as LEGO and Meccano is provided by the course leader. Concerning the actual construction of the robot the students are given a lot of freedom, as long as no floor hockey rules are violated.

There were a total of six project groups, with three to four group members in each group. Each group constructing their own robot, to compete with in the final tournament held in the end of the course.

More information about the course, the floor hockey rules and the tournament can be found at the course home page at:

http://www.nada.kth.se/kurser/kth/2D1426/.

1.2 Robot floor hockey

1.2.1 The rink and the puck

The games are performed on a floor hockey rink about 2.5 m long, and 1.5 m wide. The floor of the rink is white, except for a black midline and two black goalyards.

About an inch in from the edge of the goalyard there is a white line, that an offensive robot may not pass. In each goalyard there is also an red arc, about 2 cm wide, marking the goal. Behind some aluminium plates at these red arcs
1.2. ROBOT FLOOR HOCKEY

midlinered goal zone
black goalyard
white penalty line

Figure 1.1: An overview of the rink.

few IR-emitters are placed. The infrared light emitted from the two goals has
different modulation to make it possible for the robots to distinguish them.
The puck has a diameter of about 8 cm and a height of about 4.5 cm. Also in
the puck some IR-emitters are placed, and the emitted light from these has yet
another modulation.

1.2.2 Robot constraints

A vertical projection of the robot body must fit within a circle of 25 cm in diam-
eter, apart from the stick and very flexible parts. Also this vertical projection
may not include concavities of any kind, to make sure that the robot body can
not be used to guide the puck [3].

A robot is allowed to have one stick to guide the puck. The stick must not
extend more than one puck diameter from the robot body, and must not be
wider than one puck radius. The stick must also not have any concavities or
joints [3].

A robot must not have any device mounted that, on purpose or not, can harm
another robot [3].

1.2.3 Rules

A goal is scored only if the puck partly enters the red goal zone at the aluminium
plates and no other rule has been violated. If the scoring robot passes the white
line in the goalyard, the score is not counted, but if the robot is pushed into the
goalyard the score may be counted [3].

A robot that voluntary enters its own goalyard receives a penalty and is taken
off the rink for ten seconds [3].

1.2.4 Tournament

First there is a qualification round, where each robot, one at a time, tries to score as many times as possible during two minutes [3]. Based on the result of the qualification round, two groups are formed with three robots in each group. In the groups the games are three minutes long [3]. The two best robots from each group reaches the semi-finals. These games are four minutes long. The losers than play a bronze game, while the winners play the final. The final game is played for five minutes. If any game in the finals results in a draw, the game continues until one robot scores (known as sudden death) [3].
Chapter 2

Strategy and design

We put a lot of effort in the initial phase of the project trying to get a grip of the task and to choose a winning design. The question to answer before anything else could be done was which scoring mechanism to use. We wanted our scoring mechanism to be fast, and thus quite independent of the other robots attempts to prevent us from scoring. Based on this we figured out a very fast shooting mechanism based on a coil and a iron core.

Another goal was to design the robot body in a way that would allow the robot to make moves with the puck in all directions, without losing it. Since the robot body was not allowed to have any concavities and only one stick, we realized that this goal was impossible to achieve. We had to concentrate on finding a solution as good as possible, and ended up with the idea of a front sloping towards the stick.

2.1 Design choices

When the overall strategy was chosen there were still a lot of other design choices to make.

The first big choice was about which material to use to build the robot body. LEGO and Meccano were supplied by the course leader but we wanted something more robust, and decided to use Lexan, a plastic material that can be bent without any heating.

The sensors on the robot were used to implement the predicates the program used to make decisions. The sensors that saw the heaviest use were the IR-sensors and the timers. The IR-sensors were often used to measure distances to the goals instead of relying on the reflex detectors to stop. This was done since the robot was quite heavy and required a long breaking distance. The reflex detectors was mostly used as a backup if the robot got too close to the goals.

In all behaviours that involved travelling some limited distance we chose to use timers instead of encoders to determine when the robot had travelled far enough. This solution was simpler to implement and worked good enough.
Chapter 3

Sensing

One of the most vital parts in robot design is naturally the issue of sensing. The robot’s ability to adapt to its environment is completely dependent of its knowledge of the surroundings. Therefore the choice of sensors and their location was eagerly discussed as soon as the project started. Initially we considered four different kinds of sensors:

- **IR-sensors** to detect the infrared light emitted by the goals and the puck.
- **Reflex detectors** to detect different parts of the rink floor (painted in black and white).
- **Slotted opto detectors** to sense wheel movement.
- **Bumper switches** (and similar) to detect collisions.

Early on, we decided not to opt for any bumper switches, should it not turn out to be a necessity. It never did, hence the other sensors will be addressed in detail below and in chapter 8.

3.1 IR-sensors

There are three sources of infrared light on the rink – the two goals and the puck. In a previous version of the course there were experiments with equipping the robots with IR-emitters as well, but it turned out to be of little or no use because of the limited time span of the project, etc. The light sources are omni-directional (albeit weaker when distant), so that one can find them anywhere on the field. They are distinguished by the fact that they are of different modulation. To calibrate the IR-sensors a small test program was used. See appendix C for info on the sensors.

3.1.1 IR-sensing for navigation

Our robot utilizes five IR-sensors. The need for four of these is pretty obvious; one wants to have the ability to look in any direction in the plane of the rink.
The easiest way to achieve this is of course to place four IR-sensors at the four cardinal points, thus making every sensor cover roughly one quadrant of the field. These sensors were placed together at a piece of LEGO in order to be handled easily. However, a more intriguing issue remains: how should the sensors be oriented on the robot — or more precisely: what offset angle (in the plane of the rink) should one choose between the sensors and the robot? Two straightforward alternatives exists: to let a sensor face forward (corresponding to an 0 degree offset), or to let two sensors face equally much forward (corresponding to an 45 degree offset). We chose the latter, since this enables improved steering when one wants to head approximately straight ahead, which is the case in the crucial scoring procedure. Why? Because both of the sensors will have pretty strong input and one can compare their difference, as opposed to just having one strong sensor and two weak on the sides as information. This strategy seems to have been successful on older robots as well [1, 4]. The navigation sensor positioning is illustrated in figure 3.1. They were mounted rather high to improve the field-of-vision.

**3.1.2 IR-sensing for puck detection**

The remaining IR-sensor was used to know if the robot had the puck or not. A more naive approach would suggest the use of bumper switches instead, but
3.2 REFLEX DETECTORS

this has the disadvantage of not being able to separate between a puck collision and a collision with the rink border or an opposing robot. We considered using a combination of IR-sensors and bumper switches, but fortunately we could discard the latter altogether. The mentioned IR-sensor was placed downward looking (figure 3.1). Note that it should have a limited field-of-view, preventing the robot from incorrectly believing it has control of the puck.

The readings from the IR-sensors are preprocessed by the co-processor on the mainboard. The main PIC is then given strength values for the different IR-sensors and the different targets. The four sensors mounted on the top of the robot are used to determine the general direction of different targets and to measure distances. The directions measured are divided into broad sectors around the robot. The sectors used are

- the hemispheres to the left, right, back and front of the robot,
- the quadrant in front of the robot and
- a narrow cone straight ahead of the robot.

The distances are measured with the sum of the IR-sensors. Since the sum varies both with distance and with the direction to the target this measurement is very inaccurate. Therefore we only use the measurements with a liberal safety margin.

3.2 Reflex detectors

The floor of the rink is painted white, while the rink border, goalyards and midline are painted black. With the use of small cheap reflex detectors one can tell these white and black areas apart – the constraint is that just a few millimetres must separate a detector from the surface of recognition. We had no intention of sensing the midline (or the sideboards either for that matter), but we did want the robot to know when it had entered the two goalyards. This is important – if the robot was to enter the opposing goalyard, passing the white penalty line, when trying to score, the goal would not count. Similarly, if the robot would enter its own goalyard (passing the white line), it would receive a penalty, which is even worse! To be able to actually distinguish the white areas from the black ones we created a small test program to print the A/D-converted values from the reflex detectors on the LCD-display. From these results we could quite easily set proper thresholds for the different detectors. See appendix C for more info on the detectors.

3.2.1 Front reflex detectors

Initially we tried to put one sensor on both sides of the front. This to be able to steer the robot to face the goal in a precise way. It proved to be superfluous (the IR-sensors are enough). Far more important is the ability to stop quickly enough when entering the goalyard. Thus, bearing this in mind, we placed the sensors as far to the front as possible, moving one sensor to the end of the stick, see picture 3.1.
3.2.2 Other reflex detectors

Apart from the detectors at the front, we only had one extra detector resulting in a total of three reflex detectors. The last one was placed at the rear of the robot as illustrated by figure 3.1, and was used to avoid backing into goalyards. One could have used a multitude of reflex detectors around the edge of the robot body, but this would also result in far more complex behaviours.

Since the reflex detectors are mounted at different heights from the ground, the program uses different thresholds for the different sensors. An since digital input pins all have the threshold at the same voltage, the reflex detectors are connected to analog input pins.

3.3 Slotted opto detectors

Slotted opto detectors work in a manner similar to reflex detectors. The only major difference is that the emitter and receiver are placed opposite each other, i.e. they sense when the gap in between is opaque and not. This construction is ideally suited to be used in conjunction with encoder discs, see figure 3.2, which can be mounted on the wheels (figure 3.1), thus enabling registration of wheel rotation. Using two of these on each wheel, it is possible to determine the direction of the rotation. However we settled with just one per wheel and we glued them to the wheel suspension. The detectors were used to determine when the robot did not move (was stuck), and when the speed was high in our highly successful scoring procedure. More info on the detectors can be found in appendix C.

![Figure 3.2: An example of a simple transparent encoder disc.](image)

The two slotted opto detectors are used to measure the approximate speed of the robot by sampling the number of sectors of the encoder wheel that passed the detector in the last 300 ms. The encoders are also used to detect if the robot is stuck. This is done by testing if there has been any sector transitions on any encoder wheel in the last half second. If there has not been any transitions and if the robot is trying to move the program considers the robot stuck.
Chapter 4

Locomotion

The robot uses two 12 V motors for locomotion. The motors are equipped with gearboxes that downshifts the rotating motion from the motors ten times. To steer the robot towards a target we use measurements from the two front IR-sensors. The correction steering factor $d$ is calculated with the following formula:

$$d = c \cdot \frac{IR_{\text{target front left}} - IR_{\text{target front right}}}{IR_{\text{front left}} + IR_{\text{front right}}}$$

where $c$ is a turning constant. The result of the calculation is subtracted from the speed of the left motor and added to the speed of the right motor. Moreover, the speed is varied based on the distance from the target and the state the robot is currently in. If the robot is moving to the puck, the speed is reduced when approaching the puck to avoid the puck from bouncing off the robot. The speed is also reduced when approaching the offensive goal with the puck, to avoid crossing the white penalty line, thus invalidating the goal.

Similarly, whenever the robot is getting too close to its own goal it reduces its speed to avoid entering the goal zone and being taken off the ice. This is done globally to make sure that the program never mistakenly runs into the goal at full speed. The speed is only reduced when the robot is heading towards the goal, since there is no use in reducing the speed otherwise.
Chapter 5

Shooting

We thought that the best way to score a goal would be some kind of shooting mechanism. For that purpose we used a coil with a movable iron core that moves linearly along the moving direction of the robot. Before a shot the iron core is positioned with a bit of one end inside the coil. When the current through the coil is turned on the magnetic field will accelerate the iron core through the coil, which will eventually hit the front plate of the robot, thus giving the puck the extra speed needed for it to slide into goal. This strategy is especially successful when the robot (and hence the puck) is moving, since the puck friction will be lower than in the static case. This implies that shots made on the fly are longer.

One should mention that to come up with a working solution we needed to test quite a lot. This testing started early on – the mechanism being so important to us. The testing is described more thoroughly in chapter 8.

One interesting remark is that when the coil and the iron core was mounted to the chassis of the robot, we needed something to pull the iron core back to its starting position after a shot. For that purpose we attached a thin rubber band at the back end of the iron core – a simple construction which proved to work superbly.
Chapter 6

Operator interface

To ease the development of the program running on the robot, various visual and aural indicators could be used. We were provided with an LCD display capable of displaying two lines of text with 16 characters per line. The main board also had an assembly of LEDs that could be used for debugging purposes. There was also a speaker that could be used to play simple melodies. We only used the LCD display and the speaker and did not bother with the LEDs. The reason for ignoring the LEDs was that they were hard to read and that they didn’t provide us with enough information.

The LCD was used extensively, especially in the various test programs that were used to test parts of the robot. We used such programs for all the major systems of the robot: the motors, the sensors and the shooting mechanism. When we ran the regular program we usually chose the LCD to display IR readings for various targets and the current state in the program.

The speaker was also used, albeit moderately, to play sounds at different state transitions. The reason we did not use the speaker more was that the melodies played were far too annoying and got on everyone’s nerves (the sounds from other robots were more than enough...).
Chapter 7

Programming environment and computer hardware

7.1 Programming environment

The program is written in HI-TECH PIC C, a C language with some limitations and some special features compared to standard C [6]. For all programming in the project we have used Emacs as editor. For compiling and building our code we created a project in MPLAB an IDE from Microchip. MPLAB also has some extra features such as an interface for debugging, but we did not use any of them [7]. When building our project MPLAB created a hex file, that we could transfer to the PIC by the bootloader program NTLOAD.EXE [3].

7.2 Main board and on board devices

All components addressed below can be found in appendix C.

7.2.1 Main processor (PIC16F877)

The main processor PIC16F877 from HiTech was used to control the behaviour of the robot. When programming this processor some limitations has to be considered. The most important limitations of the processor are stated in the specifications below:

- 8K x 14 bits FLASH program memory [2]
- 368 bytes of RAM registers (SRAM) [2]
- 256 bytes of EEPROM non-volatile data memory [2]
- 20 MHz, 5MIPS RISC [2]
- 40-pin capsule [2]
7.2. MAIN BOARD AND ON BOARD DEVICES

- Hardware stack which supports a maximum number of 8 levels of function calls [3]

7.2.2 Signal converter for RS-232

The compiled and assembled program in HEX-format is transferred to the EEPROM of the PIC via a RS-232 interface. A small board for signal conversion must be used to convert the 5 V levels of the PIC, to 12 V levels of a COM-port of a host PC.

7.2.3 Co-processor (PIC16F876)

The task of the co-processor **PIC16F876** was to repeately feed the main processor with information. In our application it just filtered the signals from the IR-sensors and sent the processed data to the main processor.

Physically the co-processor is of the same type as the main processor but it is contained in a smaller 28-pin capsule.

7.2.4 Motor driver circuits (H-bridges)

A H-bridge is a circuit consisting of four transistors that can be pairwise switched on, to drive an electric motor in both directions. There is two IC-capsules on the main board containing two H-bridges each.

To be able to regulate the average voltage to the outputs of the H-bridges, pulse width modulation (PWM) is used. PWM also prevents the transistors of the H-bridges to generate too much heat.

One of the two H-bridge capsules was assigned to drive the locomotion motors. The other one was used to generate sounds for the speaker.

7.2.5 Set of dip-switches

The on board dip-switches were used to control the bootloader program and to choose which side of the rink to start the games on.

Dip-switch 6 was used for the bootloader and was connected to the input pin RC0 on the PIC. Dip-switch 2 was predefined and used by the co-processor to name the two goals as the offensive or defensive goal.
Chapter 8

Electronics

In this chapter the electric and electronic components will be discussed briefly. Each component is presented with an explanation of its use and how it was included in the robots circuits.

8.1 Electronic components and devices off the main board

8.1.1 Auxiliary electronic devices

IR-sensors

Five IR-sensors were used. Four used for navigation, one for each quadrant around the robot to recieve the signals from the goals and the puck. The fifth one was used to detect if the robot has the puck. The IR-sensors are use in a complex way and a lot of calibration of thresholds had to be done. For that reason we let the values from the IR-sensors to be printed to the LCD-display during the rest of the development of the robot.

Reflex detectors

The reflex detectors consists of an infrared LED and a photo transistor in the same capsule. The photo transistor receives infrared radiation if the infrared light from the LED is reflected in any surface. How much infrared radiation that is reflected back to the photo transistor will mainly depend on the distance to, and the colour of, the reflecting surface. When a bright surface is close to the reflex detector much of the infrared radiation is reflected back to the photo transistor. The current through the transistor will then rise which leads to a lower potential over the photo transistor. This potential is measured by the A/D-converter of the PIC.

The infrared LED’s of the reflex detectors are supplied by electric current from the LED driver board, read more about the LED driver board below.
8.1. ELECTRONIC COMPONENTS AND DEVICES OFF THE MAIN BOARD

Slotted opto detectors

The slotted opto detectors are electrically equivalent to the reflex detectors. The only difference compared to reflex detectors is the arrangement of the components. The slotted opto detectors were used together with encoder discs, consisting of alternating black and transparent pie slices. The encoder discs were cut out of overhead film. One good thing about the opto detectors is that they do not need to be calibrated. The reasons for this are the constant distance between the LED and the photo transistor, and the big difference of the potential that arises as the transparent and the black areas of the encoder discs alternates in the slot.

As for the reflex detectors the infrared LED’s are supplied by current from the LED driver board. The output potential from the transistor is connected to a Schmitt-triggered input pin of the PIC instead of any A/D-converter pin.

8.1.2 Environment manipulation devices

12 V locomotion motors

Two 12 V motors are used for the locomotion. They are supplied by pulse width modulated voltage from two H-bridges inside one of the H-bridge capsules on the main board.

100 Ω speaker

One of the H-bridges inside the second H-bridge capsule are connected to the 100 Ω speaker. It is important to use a high impedance speaker due to the fact that the voltage from the second H-bridge is 8 V. A variable resistor is used to adjust the speaker’s volume.

Goal shot coil with movable iron core

<table>
<thead>
<tr>
<th>Specifications of the coil</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length</td>
</tr>
<tr>
<td>Width x height</td>
</tr>
<tr>
<td>Inductance</td>
</tr>
<tr>
<td>Inductance with the iron core</td>
</tr>
<tr>
<td>Resistance</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Specifications of the iron core</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length</td>
</tr>
<tr>
<td>Width x height</td>
</tr>
<tr>
<td>Weight</td>
</tr>
</tbody>
</table>

Table 8.1: Specifications of the coil and core

First we tried to connect one NiCd accumulator pack to the coil. This was not enough to score a goal. After testing we decided to use two accumulator packs
in series which gave the coil a voltage of 14.4 volt. This made it possible to let the iron core start further out of the coil with a longer acceleration distance. We got more than twice as good performance and it was enough to score a goal.

To make it possible for the PIC to activate the shot coil we needed some kind of relay. We used a regular car relay.

To investigate which activation time that was best we attached the coil to a small board of wood and let the PIC turn on the current through the coil for different durations with a test program. The activation time was also printed to the LCD-display.

We ended up with an activation time of 50 ms as a good choice. If the activation time is too long the iron core will instead be de-accelerated before it hits the front plate.

To avoid the induced voltage from the coil a diod was used to short circuit this spike of voltage.

8.1.3 Accessories

Car relay

A relay is used to switch on and off the current through the goal shot coil. This current is large and it is therefore important that the relay can handle currents above 10 A. We chose a simple closing relay commonly used in cars, capable of handling 30 A.

To drive the relay a transistor (T1) was used whose base was connected to pin 34 (RB4) on the PIC through resistor R1.

Circuit wiring

The wiring for the power supply was built from cables that were soldered in split points and bunched together with plastic stripes.

The cables between the lower and the upper part of the chassis were made easily disconnectable and connectable. This makes it easy to lift off the upper part for maintenance work inside the robot.

The signal leads were connected to the main board pins with pin sockets attached to the wires. These leads were collected to a pin list and via a flat cable led to the lower part of the robot. It was arranged like this to make it easy to disconnect and connect the flat cable for maintenance.

The cables to the motors were connected to a separate socket instead of being led through the flat cable. This was because of the higher current to the motors.

8.2 Electricity supply

8.2.1 7.2 V NiCd accumulators

Two rechargeable NiCd accumulator packs (E1 and E2) were used. One dedicated for the goal shot coil and the other one for the rest of the components
including the main board and auxiliary units.

The current through the coil is delivered by both the NiCd accumulators in series.

Alternatively, alkaline batteries could have been used, but the high current consumption when shooting for goals made alkaline batteries unusable due to too high inner resistance. Alkaline cells have about eleven times higher inner resistance than NiCd cells.

8.2.2 Voltage booster board

The voltage booster board was practically unnecessary to use as we used two NiCd accumulator packs that together give 14.4 V.

To follow the rules and to not get any advantages over the other robots, we had to use the voltage booster board as the electricity source to the main board and the motors.

8.2.3 LED driver board

Reflex detectors and slotted opto detectors need an electric current to drive their LEDs. This current was taken from the LED driver board. We used three reflex detectors and two slotted opto detectors. All of these five LEDs were serially connected. The current from the LED driver board was turned to its maximum which was needed for the slotted opto detectors to work properly with the Schmitt-triggers in the PIC.
Chapter 9

Software implementation

The program controlling the robot is designed as a state machine. There is one major state for each basic behaviour of the robot. The major states are further subdivided into smaller substates to handle different parts of the behaviours.

To handle the state transitions a great number of predicates are used. The predicates truth values are based on sensor values or timers. In the states some predicates are tested to determine if the robot should switch into another state. Some transitions however are more critical and are handled globally before the current state gets a chance to run its behaviour. Because of the limited stack depth of the PIC the predicates are implemented with macros, or manually inlined in the code, instead of being functions.

9.1 Program states

In figure 9.1 the major states and state transitions are depicted. The entire source code for the robot is also included in appendix D. The main behaviour of the robot is to:

- get the puck,
- drive to the goal,
- shoot and score,
- and finally return to the center,
- and do it all over again.

There is also a number of states to handle exceptional circumstances. These states are not really exceptional, but they do not fit into the basic behaviour loop. If any of the conditions for these states becomes true a state transition will occur no matter what the robot is doing. The only exception is the state *Fire* that can not be interrupted, but on the other hand, it is only active for 50 ms. The exceptional states are interanlly prioritarized, more important states override other states. The most important state is to avoid running into the
defensive goal, since that would result in the robot being taken of the rink as a penalty. Then next important exceptional states is the stuck states, and last the state handling the case when the puck can not be detected.

9.1.1 Goto Puck

The robot starts in the state *Goto Puck* which tries to move the robot towards the puck. The state is divided into two substates. If the robot is headed in the general direction of the puck it tries to move towards the puck and applies corrective steering along the way. However, if the robot is not headed towards the puck it rotates on the spot until it is. Once the robot has reached the puck the program a transition into the state *Goto Goal* is made.

9.1.2 Goto Goal

The state *Goto Goal* tries to move the robot towards the offensive goal without losing the puck. It is also divided into two substates. If the robot is not headed towards the goal it tries to turn towards the goal. The difference from the behaviour when the robot looks for the puck is that this behaviour can only turn to the left, to avoid losing the puck. This state also only applies power to the right engine to perform this turn, which makes the turn wider and decreases the chance of loosing the puck. When the robot is headed towards the goal it starts driving towards it, applying corrective steering in the same way as when it is moving towards the puck. When the robot has a high enough speed, is headed straight towards the goal and is close enough to the goal it enters the *Fire* state. If the stick of the robot enters the offensive goal zone the robot immediately stops the motors to avoid overstepping into the goal zone. Then it also enters the *Fire* state.

9.1.3 Fire

The *Fire* state is the state that handles shooting the puck into the goal. It activates the fire coil for 50 ms and plays the scoring tune. It then switches to the *Return to Center* state.

9.1.4 Return to Center

The *Return to Center* state merely reverses the engines for one second and then transitions into the *Goto Puck* state.

9.1.5 Avoiding the defensive goal zone

If the robot gets to close to its own goal the program backs away from the goal. This is handled by two different states depending on the if the front or the back is closest to the goal.
9.1. PROGRAM STATES

9.1.6 Stuck behaviours

There are two states handling the case of the robot getting stuck. One, that is called *Stuck With Puck*, handles the case where the robot was in possession of the puck when it got stuck and the other, *Stuck Without Puck*, handles the case where the robot was not in possession of the puck.

**Stuck With Puck**

*Stuck With Puck* tries to unstuck the robot by running through two phases a couple of times. The first phase tries to turn the robot to the left on the spot and the other phase tries to put the left motor into reverse. After alternating back and forth between those two behaviours for half a second the program enters the *Goto Puck* behaviour. The *Stuck With Puck* behaviour is designed to get the robot unstuck without losing the puck.

**Stuck Without Puck**

*Stuck Without Puck* tries to get the robot unstuck and to avoid the situation where the robot runs into the wall, backs away and immediately runs into the wall at the same spot. To avoid this the program backs away, turns on the spot and drives forward for a while and then enters the *Goto Puck* behaviour.

9.1.7 Go Home

If the puck is not visible anywhere on the field the robot moves towards its own goal and stops a bit in front of it. It then tries to turn toward the offensive goal expecting the puck to be placed in the center of the rink. When the puck is detected again there is a transition to the *Goto Puck* state.
Figure 9.1: The major states used in the program.
The transition arrows not originating from any state can take place at any time.
Chapter 10

Discussion

10.1 Results

Our robot was able to navigate on the rink in a satisfactory way. It always located the offensive goal good enough for scoring, and always avoided going to fast towards its own goalyard. When our robot was alone on the rink the only problem we could observe concerning navigation, was if the puck was at the very far end of the other side of the rink. In this case the robot has difficulties locating the puck. This problem fortunately never occurred in the tournament because the puck was always placed on the midline after a score was made, meaning that the puck was always near enough for our robot to locate it.

Concerning locomotion our robot behaved pretty much as expected, but that does not imply that it behaved well. Since the robot had problems locating the puck on far distances, it consequently had problems moving towards it. In other cases our robot was very good at moving towards the puck. The biggest problem occurred when the robot was guiding the puck while moving towards the offensive goal. In these cases it often made an unnecessary 360 degrees turn to the left, before continuing towards the goal. A good indication that our locomotion behaviour was good, is that it in most cases reached the puck before the other robots, even though our robot seemed to have a lower maximum speed than the other robots before the tournament.

Our robot was excellent at scoring. During our tests it only missed scoring a goal a few times, and each time was after a fast turn to the left. In these cases it sometimes shot the puck so that it hit the rink to the left of the goal. During the competition our robot reached an scoring efficiency of 100% (computed as the quote between scored goals and scoring attempts).

The behaviour to get the robot unstuck worked well. The robot was usually able to collect the puck near the rink border, a case where other robots often got stuck without getting the puck. A little bit surprisingly the behaviour of our robot worked quite well even when trying to get the puck when it was close to another robot instead of close to the border. This was probably because the robot didn’t back away when getting stuck with the puck, thus seldom yielding in a duel.
One very bad thing about our robot was that parts of other robots often got squeezed between our wheels and the robot body, resulting in our wheels getting locked. In these cases our robot could not move at all, and it could not get loose until the referee moved the puck to a new place in a totally different direction.

10.2 Conclusions

We were very satisfied when watching our robot’s excellent scoring efficiency, since our choice of scoring mechanism put some limitations on the rest of our construction. Our robot did for example need an extra battery pack, to supply the coil with enough current. Our biggest problem was to fit the whole robot within a radius of 25 cm, since we had to make room not only for the motors and wheels but also for two battery packs and our coil and iron core. Our solution to place the coil in the middle between the wheels, made it very difficult for us to encapsulate our wheels within the robot body, so we chose not to. But as expected this made it possible for other robots to come in contact with our wheels and lock them. The next problem was the extra weight that the coil, the iron core and extra battery added to our robot. This extra weight made our robot very slow, and we were quite concerned during our tests before the tournament. Therefore we were surprised to notice that our robot many times reached the puck before the opponent. The most likely reasons is that we had smaller wheels than most of the other robots and that our program performed well. The smaller wheels meant that our acceleration was improved.

During the construction and testing of our robot our left motor appeared to be much stronger than our right motor. This was a very annoying problem that we had to solve. We ran some tests to come up with a linear model describing the difference in strength at different speeds. Then we used this model to lower the speed of our left motor, but this solution was unsatisfactory since we could not use the full strength anymore. After these results we decided to try replacing the right motor, but nothing changed. Then we tried to switch side of the two motors, and now it turned out that the previously stronger motor appeared to be weaker. We realized that all our work with finding a linear model, and trying to replacing motors, was a waste of time. Now we simply moved the battery packs a bit more to the left on the robot, until the robot appeared to move straight forward, when setting the same speed on both motors.

To avoid penalties for entering our own goalyard the robot lowered its speed when it came close to its own goal. Then it tried to reverse both motors if any reflex detector sensed the black area. But our robot still sometimes passed the white penalty line in the goalyard. Therefore we made the robot stop if it received strong enough IR light from the goal. This worked well during the tournament, because we did not get any penalties at all, but it was a little bit unsatisfactory when our robot once did not reach the puck even though it was clearly outside our own goalyard.

One problem that occurred late in our test phase was that our program sometimes passed the maximum stack level of eight levels. In these cases the behaviour of our robot became very weird. Unfortunately we had to rewrite parts of our program, and did not have much time, so the resulting program is pretty ugly.
10.3 Tips for the future

- Finnish your mechanical construction as early as possible, and then write your code. Writing code for a robot that is going to change form, weight, etc. will be of little use, because parameters will need to be tested and changed.

- Try to make the surface of your robot as smooth as possible, to avoid getting stuck against other robots.

- Write very simple test programs in the beginning to get familiar with the system at your hands, and to make sure that all sensors and components really are functioning.

- Place the wheels within the robot body so that they are totally covered, and cannot come in contact with anything else than the rink floor.

- If one motor appears to be stronger, try to put more of the robots' weight on the same side as this motor.

- Try to come up with a very fast scoring procedure.

- Avoid competition penalties.

- Keep your software solutions simple. They almost always perform as good as much more complex solutions (as long as the simple solution is not stupid).

- Backup your program often, so that you can compare your new improved program with the old one. Then if the behaviour of the robot did not improve, maybe you should go back to your old program solution.
Appendix A

Circuit diagrams and connections

Figure A.1: A circuit diagram of the motherboard used in the course.
<table>
<thead>
<tr>
<th>Pin</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Reset</td>
</tr>
<tr>
<td>2</td>
<td>Reflex detector, front, AN0, with pull-up resistor</td>
</tr>
<tr>
<td>3</td>
<td>Reflex detector, back, AN1, with pull-up resistor</td>
</tr>
<tr>
<td>4</td>
<td>Unused</td>
</tr>
<tr>
<td>5</td>
<td>Reflex detector, stick, AN3, with pull-up resistor</td>
</tr>
<tr>
<td>6</td>
<td>Wheel encoder, left, RA4, with pull-up resistor</td>
</tr>
<tr>
<td>7</td>
<td>Unused</td>
</tr>
<tr>
<td>8</td>
<td>Wheel encoder, right, RE0, with pull-up resistor</td>
</tr>
<tr>
<td>9-14</td>
<td>Unused</td>
</tr>
<tr>
<td>15</td>
<td>Boot loader / program chooser, RC0</td>
</tr>
<tr>
<td>16</td>
<td>Motor enable, left motor, CCP2</td>
</tr>
<tr>
<td>17</td>
<td>Motor enable, right motor, CCP1</td>
</tr>
<tr>
<td>18-24</td>
<td>Unused</td>
</tr>
<tr>
<td>25</td>
<td>RS-232, TX</td>
</tr>
<tr>
<td>26</td>
<td>RS-232, RX</td>
</tr>
<tr>
<td>27-30</td>
<td>Unused</td>
</tr>
<tr>
<td>31</td>
<td>RS-232 level converter, VDD</td>
</tr>
<tr>
<td>32</td>
<td>RS-232 level converter, voltage supply, VSS</td>
</tr>
<tr>
<td>33</td>
<td>Motor direction, left motor, RB0</td>
</tr>
<tr>
<td>34</td>
<td>Motor direction, right motor, RB1</td>
</tr>
<tr>
<td>35</td>
<td>Enable sound RB2</td>
</tr>
<tr>
<td>36</td>
<td>Polarity sound RB3</td>
</tr>
<tr>
<td>37</td>
<td>Goal shot coil, RB4</td>
</tr>
<tr>
<td>38-40</td>
<td>Unused</td>
</tr>
</tbody>
</table>

Table A.1: Used pins of the main processor (PIC)
Figure A.2: Signal circuit diagram

R. 2.5 kΩ resistor
T. Transistor BC-137, Philips
A. Reflex detector, front
B. Slotted opto detector, left encoder
C. Reflex detector, back
D. Slotted opto detector, right encoder
E. Reflex detector, stick

The PIC-pins for the detectors A-F have
4.7 kΩ pull-up resistors.
Figure A.3: Power supply diagram

R. 2.5 kΩ resistor
T. Transistor BC-337, Philips
L. 0.8 mH coil with movable iron core
D. Diode 1N4004, Philips
S. Main switch
S. Car relay, BilTema
Appendix B

Pictures of the robot

Figure B.1: The vicious grin of the robot front.
Figure B.2: The back of the robot.

Figure B.3: The robot seen from above.
Figure B.4: A study of the shooting mechanism.

Figure B.5: An isometric view of the robot.
Figure B.6: The lower of the two robot modules.

Figure B.7: The two robot modules side by side.
# Appendix C

## Parts list

### C.1 Parts on main board

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Device</th>
<th>Model</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PIC (main processor)</td>
<td>PIC16F877 40-pin</td>
<td>Microchip Technology Inc. [1]</td>
</tr>
<tr>
<td>1</td>
<td>PIC (for IR)</td>
<td>PIC16F876 28-pin</td>
<td>Microchip Technology Inc. [1]</td>
</tr>
<tr>
<td>1</td>
<td>Signal converter for RS-232</td>
<td>[3]</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Motor driver circuit</td>
<td>L293NE</td>
<td>SGS Thomson [5]</td>
</tr>
</tbody>
</table>

### C.2 Electronic parts off main board

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Device</th>
<th>Model</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Reflex detector</td>
<td>ITR8307</td>
<td>Everlight [5]</td>
</tr>
<tr>
<td>5</td>
<td>IR sensors</td>
<td>TSL261</td>
<td>Texas instr. [5]</td>
</tr>
<tr>
<td>2</td>
<td>Slotted opto detector</td>
<td>ITR8010</td>
<td>Everlight [5]</td>
</tr>
<tr>
<td>2</td>
<td>12 V motor with 10:1 gear</td>
<td>HL149</td>
<td>Micromotor [5]</td>
</tr>
<tr>
<td>1</td>
<td>100 Ω speaker</td>
<td>20SC100G [5]</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.8 mH coil with movable iron core</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Car relay</td>
<td>42-302</td>
<td>Bitlema</td>
</tr>
<tr>
<td>1</td>
<td>Transistor</td>
<td>BC-337</td>
<td>Philips</td>
</tr>
<tr>
<td>1</td>
<td>Resistor 2.5 kΩ</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>LCD</td>
<td>PC1602D A</td>
<td>Powertip [5]</td>
</tr>
<tr>
<td>1</td>
<td>Diode</td>
<td>1N4004</td>
<td>Philips</td>
</tr>
</tbody>
</table>

### C.3 Components for electricity supply

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>7.2 V NiCd accumulators [5]</td>
</tr>
<tr>
<td>1</td>
<td>Voltage booster board, 7.2V to 14.5 V</td>
</tr>
<tr>
<td>1</td>
<td>LED driver board [5]</td>
</tr>
</tbody>
</table>
## C.4 Body parts

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Wheel</td>
</tr>
<tr>
<td>2</td>
<td>Encoder disc, 16 sectors</td>
</tr>
<tr>
<td>Approx 500 cm(^2)</td>
<td>Plastic boards, Lexan</td>
</tr>
</tbody>
</table>
Appendix D

Source code

D.1 adc.h

```c
// ***************************** adc.h

//sample an A/D channel and return the eight most
//significant bits (the two least significant 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 conversion 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)

//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, where x is the channel number

extern char sample_ad_channel(char channel);
```

D.2 debug.h

```c
// ***************************** 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
```
D.3. DEFINES.H

// 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 change 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)
// EEPROM - 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();

D.3 defines.h

// ***************************** defines.h

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

D.4 demo.c

#include "defines.h"
#include "adc.h"
#include "sprint.h"
#include "debug.h"
#include "ir_comm.h"
#include "pwm.h"
#include "intr.h"
#include "button.h"
#include "eeprom.h"
#include "serialio.h"

#define ENTER_AVOID_FRONT_GOAL { 
  avoid_time = soft_time(); 
  set_motors(-MAX_BACKWARD_SPEED, -MAX_BACKWARD_SPEED);} 
#define ENTER_AVOID_BACK_GOAL { 

40
D.4. DEMO.C

```c
20    avoid_time = soft_time(); \
21    set_motors(MIN_SPEED, MAX_FORWARD_SPEED);}
22
23 // relay and fire mechanism constants
24 #define RELAY_ENBL_PIN RB4
25 #define RELAY_ENBL_PIN_TRIS TRISB4
26
27 // ir sensing constants
28 #define IR_FL 2 // front left sensor, using IR port 2
29 #define IR_FR 1 // front right sensor, using IR port 1
30 #define IR_BL 3 // back left sensor, using IR port 3
31 #define IR_BR 0 // back right sensor, using IR port 0
32 #define IR_S 4 // stick sensor, using IR port 4
33
34 // reflex detectors constants
35 #define REFLEX_F 0 // front detector, using A/D channel 0
36 #define REFLEX_S 1 // stick detector, using A/D channel 1
37 #define REFLEX_B 3 // back detector, using A/D channel 3
38
39 // encoder constants (from intr.h)
40 // The left encoder is connected to RA4 and the right one on RE0
41 #define ENCODER_LEFT_POS (encoder_left_pos)
42 #define ENCODER_RIGHT_POS (encoder_right_pos)
43
44 // timing constants
45 #define FIRE_DURATION 50
46 #define STUCK_WITH_PUCK_DURATION 400
47 #define STUCK_WITH_PUCK_PHASE_DURATION 50
48 #define STUCK_WITH_PUCK_BACKWARD_DURATION 5
49 #define STUCK_WITHOUT_PUCK_TURN_RIGHT_DURATION 300
50 #define STUCK_WITHOUT_PUCK_BACKWARD_DURATION 250
51 #define STUCK_WITHOUT_PUCK_FORWARD_DURATION 350
52 #define RETURN_TO_CENTER_DURATION 1300
53 #define AVOID_FRONT_DURATION 500
54 #define AVOID_BACK_DURATION 400
55
56 // How often to sample the speed
57 #define SPEED_SAMPLE_INTERVAL 300
58
59 // Speeds for different kinds of movement
60 #define LOW_SPEED 113 // when close to target
61 /* max speed allowed in set_motors() */
62 #define MAX_FORWARD_SPEED max_forward_speed()
63 #define MAX_BACKWARD_SPEED max_backward_speed()
64
65 #define TURN_SPEED 127 // when turning without puck
66 #define PUCK_TURN_SPEED 119 // when turning with puck
67 #define PUCK_ROTATE_SPEED 110 // when rotating with puck, only at goal yard
68 #define MIN_SPEED 110 // min speed allowed in set_motors()
69 #define HAS_HIGH_SPEED (current_speed > 11)
70
71 // ir sensing macros
72 #define IR_STICK(x) (ir_value(IR_S, x))
73 #define IR_FRONT(x) (ir_value(IR_FL, x) + ir_value(IR_FR, x))
74 #define IR_BACK(x) (ir_value(IR_BL, x) + ir_value(IR_BR, x))
75 #define IR_LEFT(x) (ir_value(IR_FL, x) + ir_value(IR_BL, x))
76 #define IR_RIGHT(x) (ir_value(IR_FR, x) + ir_value(IR_BR, x))
77 #define IR_SUM(x) (IR_FRONT(x) + IR_BACK(x))
78 #define IR_FRONT_LEFT(x) (ir_value(IR_FL, x))
79 #define IR_FRONT_RIGHT(x) (ir_value(IR_FR, x))
80 #define IR_BACK_LEFT(x) (ir_value(IR_BL, x))
81 #define IR_BACK_RIGHT(x) (ir_value(IR_BR, x))
```

41
```c
#define IR_BACK_RIGHT(x) (ir_value(IR_BR, x))

// Predicates using the IR values
#define TARGET_IN_FRONT_QUADRANT(t) (IR_FRONT(t) > IR_BACK(t) \\
    && IR_FRONT(t) > IR_LEFT(t) \\
    && IR_FRONT(t) > IR_RIGHT(t))
#define TARGET_TO_THE_LEFT(t) (IR_LEFT(t) >= IR_RIGHT(t))
#define TARGET_TO_THE_RIGHT(t) (IR_LEFT(t) < IR_RIGHT(t))
#define TARGET_CLOSE(x) (IR_SUM(x) > 200)
#define GOAL_CLOSE_TO_FRONT(x) (IR_FRONT(x) > 230)
#define GOAL_CLOSE_TO_BACK(x) (IR_BACK(x) > 230)
#define ANY_GOAL_CLOSE_TO_FRONT (GOAL_CLOSE_TO_FRONT(OFF_GOAL) \\
    || GOAL_CLOSE_TO_FRONT(DEF_GOAL))
#define ANY_GOAL_CLOSE_TO_BACK (GOAL_CLOSE_TO_BACK(OFF_GOAL) \\
    || GOAL_CLOSE_TO_BACK(DEF_GOAL))
#define PUCK_CLOSE_TO_FRONT (IR_FRONT(PUCK) > 800)
#define GOAL_CLOSE_FOR_SHOT (IR_FRONT(OFF_GOAL) > 150) // shot
#define HAS_PUCK (IR_STICK(PUCK) > 1000)
#define GOAL_STRAIGHT_AHEAD (((30 * ABS(IR_FRONT_LEFT(OFF_GOAL) \\
    - IR_FRONT(OFF_GOAL))) \\
    / IR_FRONT(OFF_GOAL) < 22)
#define FRONT_OVER_GOALYARD ((REFLEX_FRONT_IS_BLACK || REFLEX_STICK_IS_BLACK) \\
    && ANY_GOAL_CLOSE_TO_FRONT)
#define BACK_OVER_GOALYARD (REFLEX_BACK_IS_BLACK \\
    && ANY_GOAL_CLOSE_TO_FRONT)
#define TARGET_VISIBLE(x) ((IR_SUM(x) + IR_STICK(x)) > 50)

// reflex detecting definitions
#define REFLEX_FRONT (sample_ad_channel(REFLEX_F))
#define REFLEX_STICK (sample_ad_channel(REFLEX_S))
#define REFLEX_BACK (sample_ad_channel(REFLEX_B))

// Predicates for the reflex detectors
#define REFLEX_FRONT_IS_BLACK (REFLEX_FRONT > 235)
#define REFLEX_STICK_IS_BLACK (REFLEX_STICK > 240)
#define REFLEX_BACK_IS_BLACK (REFLEX_BACK > 230)

// Predicates combining the IR values and the reflex detectors
#define FRONT_OVER_GOALYARD ((REFLEX_FRONT_IS_BLACK || REFLEX_STICK_IS_BLACK) \\
    && ANY_GOAL_CLOSE_TO_FRONT)
#define BACK_OVER_GOALYARD (REFLEX_BACK_IS_BLACK \\
    && ANY_GOAL_CLOSE_TO_FRONT)

// Predicates for internal state
#define IS_STUCK (passed(latest_move_time + 700) && \\
    (motor_left != 0 || motor_right != 0))
#define HANDLING_STUCK (S == STUCK_SPIN_FREE || S == STUCK_BACKUP)

// math macros
#define MIN(x,y) (x < y ? x : y)
#define MAX(x,y) (x > y ? x : y)
#define ABS(x) (((x) < 0) ? (-x) : (x))

// behavior state macros
#define GOTO_PUCK 0
#define TURN_LEFT_TO_PUCK 1
#define TURN_RIGHT_TO_PUCK 2
#define MOVE_TO_PUCK 3
#define GOTO_GOAL 10
#define TURN_TO_GOAL 11
#define MOVE_TO_GOAL 12
```
```
#define STUCK_WITH_PUCK 20
#define STUCK_WITHOUT_PUCK 21
#define RETURN_TO_CENTER 25
#define FIRE 30
#define GO_HOME 40
#define AVOID_FRONT_GOAL 91
#define AVOID_BACK_GOAL 92

// type definitions
typedef char State;
typedef char Bool;
typedef char Target;

// global variables
bank1 static State S = GOTO_PUCK; // Start state
bank1 static char buffer[7];
bank1 static signed char motor_left = 0;
bank1 static signed char motor_right = 0;
bank2 static unsigned char current_speed = 0;
bank2 static unsigned int stuck_time;
bank2 static unsigned int stuck_phase_time;
bank2 static unsigned int fire_time;
bank2 static unsigned int return_time;
bank2 static unsigned int avoid_time;
bank2 static char runs_stuck_phase;

// Tune to play when scoring a goal
const char axel_foley[] = {0x05,48,0x08,32,0x05,12,0xFF,12,0x05,12,0x08,
  12,0xFF,12,0x05,24,0x03,24,0x05,24,0xFF,24,0x10,
  12,0x05,12,0xFF,12,0x05,12,0x11,12,0xFF,12,0x10,
  24,0x08,24,0x05,12,0xFF,12,0x10,12,0xFF,12,0x15,
  12,0x05,12,0x03,24,0xFF,8,0x03,12,0x00,12,0xFF,
  12,0x07,12,0xFF,12,0x05,24,0x05,63,
  0,0};

// help functions
void sleep(int msec); // busy sleep
signed char clamp(signed int speed);
void set_motors(signed int left, signed int right);
void print_state_info(State state, int diff, Target target);

// Test functions, not used by the main program
void test_motors();
void test_ir_sensors();
void test_reflex_detectors();
void test_max_speed_to_stop();
void test_encoders();
void fire_loop();
void test_wheel_positions();
void test_longshot();

// state functions
State goto_puck();
State goto_goal();
State return_to_center();
State stuck_with_puck();
State stuck_without_puck();
State fire();
State avoid_front_goal();
State avoid_back_goal();
```
D.4. DEMO.C

// behaviour functions ("part of state"-states)
void turn_to_puck();
void move_to_puck();
State turn_to_goal();
State move_to_goal();
State go_home();

// Functions for entering states
State enter_stuck_with_puck();
State enter_stuck_without_puck();
State enter_return_to_center();

// Functions regulating the maximum speeds forward and backwards
int max_forward_speed();
int max_backward_speed();

// 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);

int main() {
    bank2 static unsigned int last_speed_sample_time = 0;
    unsigned int left_pos = 0;
    unsigned int right_pos = 0;

    main_init();

    //test_motors();
    //test_wheel_positions();
    //test_ir_sensors();
    //test_reflex_detectors();
    //test_longshot();

    /* Main state loop */
    for(;;) {
        /* Sampling of the speed */
        if(passed(last_speed_sample_time + SPEED_SAMPLE_INTERVAL)) {
            current_speed = (unsigned char)((ENCODER_LEFT_POS - left_pos) +
            (ENCODER_RIGHT_POS - right_pos))/2;
            left_pos = ENCODER_LEFT_POS;
            right_pos = ENCODER_RIGHT_POS;
        }
        last_speed_sample_time = soft_time();
    }

    /* Global state transitions */
    if (IR_FRONT(DEF_GOAL) > 375) {
        ENTER_AVOID_FRONT_GOAL;
        S = AVOID_FRONT_GOAL;
    } else if (IR_BACK(DEF_GOAL) > 300) {
        ENTER_AVOID_BACK_GOAL;
        S = AVOID_BACK_GOAL;
    } else if (((FRONT_OVER_GOALYARD && !HAS_PUCK)) {
        ENTER_AVOID_FRONT_GOAL;
        S = AVOID_FRONT_GOAL;
    } else if (BACK_OVER_GOALYARD) {
        ENTER_AVOID_BACK_GOAL;
    }
}
D.4. DEMO.C

```c
if (IS_STUCK) {
    if (IR_STICK(PUCK) > 1000) {
        S = enter_stuck_with_puck();
    } else {
        S = enter_stuck_without_puck();
    }
} else if (!TARGET_VISIBLE(PUCK)) {
    S = GO_HOME;
}

/* Switch into the right state */
switch (S) {
    case GOTO_PUCK:
        S = goto_puck();
        break;
    case GOTO_GOAL:
        S = goto_goal();
        break;
    case FIRE:
        S = fire();
        break;
    case RETURN_TO_CENTER:
        S = return_to_center();
        break;
    case STUCK_WITH_PUCK:
        S = stuck_with_puck();
        break;
    case STUCK_WITHOUT_PUCK:
        S = stuck_without_puck();
        break;
    case AVOID_FRONT_GOAL:
        S = avoid_front_goal();
        break;
    case AVOID_BACK_GOAL:
        S = avoid_back_goal();
        break;
    case GO_HOME:
        S = go_home();
        break;
    default:
        S = GOTO_GOAL;
        break;
}

/* Initialize the hardware and the software */
int main_init() {
    ir_init();  //initialize IR co-processor communication
    init_pwm();  //initialize pwm for motor control
    init_soft_tmr();  //initialize soft timer
    init_sound();  //initialize sound generator
    init_enc();

    // set A/D channels 0, 1, 3 to analogue and input
    ADCUN1 = 0b00000100;
    TRISA0 = TRISA1 = TRISA3 = 1;

    // enable interrupts
    PEIE = 1;  // peripheral interrupts enable
    ei();  // global interrupt enable
```
D.4 DEBO.C

330 // set relay pin to output
331 RELAY_ENBL_PIN_TRIS = 0;
332 RELAY_ENBL_PIN = 0;
333
334 // turn on lcd
335 set_lcd_power(1);
336 sleep(100);
337 init_lcd();
338 return 0;
339 }
340
341 /* Clamps a value to the maximum motor speeds */
342 signed char clamp(signed int speed) {
343 if (speed < 0) {
344 return MAX(speed, ~MAX_BACKWARD_SPEED);
345 } else {
346 return MIN(speed, MAX_FORWARD_SPEED);
347 }
348 }
349
350 /* Sets the speed of the motors */
351 void set_motors(signed int left, signed int right) {
352 if (left < 0) {
353 motor_left = MAX(left, ~MAX_BACKWARD_SPEED);
354 } else {
355 motor_left = MIN(left, MAX_FORWARD_SPEED);
356 }
357 if (right < 0) {
358 motor_right = MAX(right, ~MAX_BACKWARD_SPEED);
359 } else {
360 motor_right = MIN(right, MAX_FORWARD_SPEED);
361 }
362 setpwm(s2pwm(motor_left), s2pwm(motor_right));
363 setdir(left>0, right>0);
364 }
365
366 State goto_puck() {
367 if (TARGET_IN_FRONT_QUADRANT(PUCK)) {
368 if (HAS_PUCK) {
369 return GOTO_GOAL;
370 } move_to_puck();
371 } else {
372 turn_to_puck();
373 }
374 return GOTO_PUCK;
375 }
376
377 /* Goes to the offensive goal with the puck */
378 State goto_goal() {
379 if (HAS_PUCK) {
380 if (TARGET_IN_FRONT_QUADRANT(OFF_GOAL)) {
381 return move_to_goal();
382 } else {
383 return turn_to_goal();
384 }
385 } return GOTO_PUCK;
386 }
387
388
D.4. DEMO.C

void turn_to_puck() {
    if (TARGET_TO_THE_LEFT(PUCK)) {
        if (GOAL_CLOSE_TO_FRONT(DEF_GOAL)) {
            set_motors(-MIN_SPEED, MIN_SPEED);
        } else {
            set_motors(-TURN_SPEED, TURN_SPEED);
        }
        print_state_info(TURN_LEFT_TO_PUCK, 0, PUCK);
    } else {
        set_motors(TURN_SPEED, -TURN_SPEED);
        print_state_info(TURN_RIGHT_TO_PUCK, 0, PUCK);
    }
}

void move_to_puck() {
    bank2 static int speed;
    bank2 static int diff;
    bank2 static int left;
    bank2 static int right;
    speed = PUCK_CLOSE_TO_FRONT ? LOW_SPEED : MAX_FORWARD_SPEED;
    speed = GOAL_CLOSE_TO_FRONT(DEF_GOAL) ? MIN_SPEED : speed;
    diff = IR_FRONT_LEFT(PUCK) - IR_FRONT_RIGHT(PUCK);
    diff = (10 * diff) / IR_FRONT(PUCK);
    left = speed - diff;
    right = speed + diff;
    set_motors(left, right);
    print_state_info(MOVE_TO_PUCK, diff, PUCK);
}

/* Turns towards the offensive goal with the puck */
State turn_to_goal() {
    unsigned char speed = IR_FRONT_LEFT_STRONGEST(OFF_GOAL) ? MIN_SPEED
    : PUCK_TURN_SPEED;
    if (FRONT_OVER_GOALYARD && GOAL_CLOSE_TO_FRONT(OFF_GOAL)) {
        set_motors(-MIN_SPEED, MIN_SPEED);
    } else {
        set_motors(0, speed);
    }
    print_state_info(TURN_TO_GOAL, 0, OFF_GOAL);
    return GOTO_GOAL;
}

/* Moves to the offensive goal with the puck */
State move_to_goal() {
    bank2 static int speed;
    bank2 static int diff;
    bank2 static int left;
    bank2 static int right;
    speed = ANY_GOAL_CLOSE_TO_FRONT ? LOW_SPEED : MAX_FORWARD_SPEED;
    diff = IR_FRONT_LEFT(OFF_GOAL) - IR_FRONT_RIGHT(OFF_GOAL);
    diff = (15 * diff) / IR_FRONT(OFF_GOAL);
    if (FRONT_OVER_GOALYARD && GOAL_CLOSE_TO_FRONT(DEF_GOAL)) {
        ENTER_AVOID_FRONT_GOAL;
    }
D.4. DEMO.C

```c
454         return AVOID_FRONT_GOAL;
455     }
456     if (FRONT_OVER_GOALYARD && GOAL_CLOSE_TO_FRONT(OFF_GOAL)) {
457         //try to score
458         // important to stop quickly to avoid driving into the goal zone
459         set_motors(0, 0);
460         return FIRE;
461     } else if (HAS_HIGH_SPEED && GOAL_CLOSE_FOR_SHOT && GOAL_STRAIGHT_AHEAD) {
462         //try a fast score
463         return FIRE;
464     } else {
465         left = speed - diff;
466         right = speed + diff;
467         set_motors(left, right);
468     }
469 }
470     print_state_info(MOVE_TO_GOAL, diff, OFF_GOAL);
471     return GOTO_GOAL;
472 }
473 }
474
475 /*
476 * Enters the state to handle getting the robot unstuck with the puck
477 */
478 State enter_stuck_with_puck() {
479     stuck_time = soft_time();
480     runs_stuck_phase = 1;
481     stuck_phase_time = soft_time();
482     print_state_info(STUCK_WITH_PUCK, 0, OFF_GOAL);
483     return STUCK_WITH_PUCK;
484 }
485
486 /*
487 * Try to get the robot unstuck and try to keep the puck.
488 * This behaviour is divided into two phases that repeats.
489 * The first phase tries to turn on the spot.
490 * The second phase stops the right engine and reverses the left engine
491 */
492 State stuck_with_puck() {
493     if (passed(stuck_time + STUCK_WITH_PUCK_DURATION)) {
494         return GOTO_PUCK;
495     } else {
496         if (runs_stuck_phase == 1) {
497             set_motors(-MAX_BACKWARD_SPEED, MAX_FORWARD_SPEED);
498             if (passed(stuck_phase_time + STUCK_WITH_PUCK_PHASE_DURATION)) {
499                 runs_stuck_phase = 2;
500                 stuck_phase_time = soft_time();
501             }
502         } else if (runs_stuck_phase == 2) {
503             set_motors(-MAX_BACKWARD_SPEED, 0);
504             if (passed(stuck_phase_time + STUCK_WITH_PUCK_PHASE_DURATION)) {
505                 runs_stuck_phase = 1;
506                 stuck_phase_time = soft_time();
507             }
508         }
509     }
510     print_state_info(STUCK_WITH_PUCK, 0, OFF_GOAL);
511     return STUCK_WITH_PUCK;
512 }
513 ```
 State enter_stuck_without_puck() {
    runs_stuck_phase = 1;
    stuck_time = soft_time();
    return STUCK_WITHOUT_PUCK;
}

 State stuck_without_puck() {
    if(runs_stuck_phase == 1) {
        set_motors(-MAX_BACKWARD_SPEED, -MAX_BACKWARD_SPEED);
        if (passed(stuck_time + STUCK_WITHOUT_PUCK_BACKWARD_DURATION)) {
            runs_stuck_phase = 2;
            stuck_time = soft_time();
        } else if (runs_stuck_phase == 2) {
            set_motors(-MAX_BACKWARD_SPEED, MAX_FORWARD_SPEED);
            if (passed(stuck_time + STUCK_WITHOUT_PUCK_TURN_RIGHT_DURATION)) {
                runs_stuck_phase = 3;
                stuck_time = soft_time();
            } else if (runs_stuck_phase == 3) {
                //Forward alittel bit
                set_motors(MAX_FORWARD_SPEED, MAX_FORWARD_SPEED);
                if (passed(stuck_time + STUCK_WITHOUT_PUCK_FORWARD_DURATION)) {
                    return GOTOPUCK;
                }
            }
        } return STUCK_WITHOUT_PUCK;
    }

 State avoid_front_goal() {
    if (passed(avoid_time + AVOID_FRONT_DURATION)) {
        return GOTOPUCK;
    } else {
        return AVOID_FRONT_GOAL;
    }
}

 State avoid_back_goal() {
    if (passed(avoid_time + AVOID_BACK_DURATION)) {
        return GOTOPUCK;
    } else {
        return AVOID_BACK_GOAL;
    }

*/ Enter the state to handle getting the robot unstuck when not having the puck */
State enter_stuck_without_puck() {
    runs_stuck_phase = 1;
    stuck_time = soft_time();
    return STUCK_WITHOUT_PUCK;
}

 /*
 Tries to get the robot unstuck and get the robot closer to the puck
 * This state is divided into three phases.
 * The first phase backs up
 * The second phase turns on the spot to the left
 * The third phase drives forward
 */
State stuck_without_puck() {
    if(runs_stuck_phase == 1) {
        set_motors(-MAX_BACKWARD_SPEED, -MAX_BACKWARD_SPEED);
        if (passed(stuck_time + STUCK_WITHOUT_PUCK_BACKWARD_DURATION)) {
            runs_stuck_phase = 2;
            stuck_time = soft_time();
        } else if (runs_stuck_phase == 2) {
            set_motors(-MAX_BACKWARD_SPEED, MAX_FORWARD_SPEED);
            if (passed(stuck_time + STUCK_WITHOUT_PUCK_TURN_RIGHT_DURATION)) {
                runs_stuck_phase = 3;
                stuck_time = soft_time();
            } else if (runs_stuck_phase == 3) {
                //Forward alittel bit
                set_motors(MAX_FORWARD_SPEED, MAX_FORWARD_SPEED);
                if (passed(stuck_time + STUCK_WITHOUT_PUCK_FORWARD_DURATION)) {
                    return GOTOPUCK;
                }
            }
        } return STUCK_WITHOUT_PUCK;
    }

 /* Backs away from the defensive goal when coming to close going forward */
State avoid_front_goal() {
    if (passed(avoid_time + AVOID_FRONT_DURATION)) {
        return GOTOPUCK;
    } else {
        return AVOID_FRONT_GOAL;
    }
}

 /*
 * Avoid the defensive goal when aproaching with the back first
 */
State avoid_back_goal() {
    if (passed(avoid_time + AVOID_BACK_DURATION)) {
        return GOTOPUCK;
    } else {
        return AVOID_BACK_GOAL;
    }
}
/*
 * Activate the coil and play the scoring tune
 */
State fire() {
  fire_time = soft_time();
  RELAY_ENBL_PIN = 1;
  set_motors(0, 0);
  while (!passed(fire_time + FIRE_DURATION));
  RELAY_ENBL_PIN = 0;
  set_motors(-MAX_BACKWARD_SPEED, -MAX_BACKWARD_SPEED);
  play_tune(axel_foley);
  return enter_return_to_center();
}

/*
 * Start returning to the center
 */
State enter_return_to_center() {
  return_time = soft_time();
  set_motors(-MAX_BACKWARD_SPEED, -MAX_BACKWARD_SPEED);
  return RETURN_TO_CENTER;
}

/*
 * Return to the center
 */
State return_to_center() {
  if (passed(return_time + RETURN_TO_CENTER_DURATION)) {
    return GOTO_PUCK;
  } else {
    return RETURN_TO_CENTER;
  }
}

/*
 * Print the state the diff for steering and the IR values of the target
 */
void print_state_info(State state, int diff, Target target) {
  sprint16(buffer, state);
  lcd_print1("s", 0);
  lcd_print1(buffer, 3);
  sprint16(buffer, diff);
  lcd_print1("d", 3);
  sprint16(buffer, IR_FRONT_LEFT(target));
  lcd_print1(buffer, 7);
  sprint16(buffer, IR_FRONT_RIGHT(target));
  lcd_print1(buffer, 12);
  sprint16(buffer, IR_STICK(target));
  lcd_print2(buffer, 0);
  sprint16(buffer, IR_BACK_LEFT(target));
  lcd_print2(buffer, 7);
sprint16(buffer, IR_BACK_RIGHT(target));
lcd_print2(buffer, 12);
}

void sleep(int msec) {
    bank3 static int sometime;
    sometime = soft_time();
    for (;;) {
        if (passed(sometime + msec)) {
            break;
        }
    }
}

int max_forward_speed() {
    return ((IR_FRONT(DEF_GOAL) > 200) ? LOW_SPEED : 127);
}

int max_backward_speed() {
    return ((IR_BACK(DEF_GOAL) > 200) ? LOW_SPEED : 128);
}

void turn_to_home() {
    if (TARGET_TO_THE_LEFT(DEF_GOAL)) {
        set_motors(-TURN_SPEED, TURN_SPEED);
    } else {
        set_motors(TURN_SPEED, -TURN_SPEED);
    }
}

void move_home() {
    bank2 static int speed;
    bank2 static int diff;
    bank2 static int left;
    bank2 static int right;
    speed = GOAL_CLOSE_TO_FRONT(DEF_GOAL) ? MIN_SPEED : MAX_FORWARD_SPEED;
    diff = IR_FRONT_LEFT(DEF_GOAL) - IR_FRONT_RIGHT(DEF_GOAL);
    diff = (10 * diff) / IR_FRONT(DEF_GOAL);
    left = speed - diff;
    right = speed + diff;
    set_motors(left, right);
D.4. DEMO.C

702  }
703  */
704  */  * Turns towards the offensive goal without the puck  */
705  void turn_to_off_goal() {
706    if (TARGET_TO_THE_LEFT(OFF_GOAL)) {
707      set_motors(-TURN_SPEED, TURN_SPEED);
708    } else {
709      set_motors(TURN_SPEED, -TURN_SPEED);
710    }
711  }
712  */
713  *  * Go home when the puck isn't visible. */
714  */
715  State go_home() {
716    if(TARGET_VISIBLE(PUCK)) {
717      return GOTO_PUCK;
718    }
719    if (TARGET_CLOSE(DEF_GOAL)) {
720      if (TARGET_IN_FRONT_QUADRANT(OFF_GOAL)) {
721        set_motors(0,0);
722        return GO_HOME;
723      } else {
724        turn_to_off_goal();
725      }
726    } else if (!TARGET_IN_FRONT_QUADRANT(DEF_GOAL)) {
727      turn_to_home();
728    } else {
729      move_home();
730    }
731  }
732  }  /*
733  /**********
734  * Test functions for various parts of the robot
735  * Not used by the main program
736  ***********/
737  */
738  * Prints the sum and the difference between the two frontal IR-sensors
739  * Used to tune the long shot behaviour
740  */
741  void test_longshot() {
742    for(;;) {
743      lcd_print1("\n", 0);
744      lcd_print2("\n", 0);
745      sprint16(buffer, IR_FRONT(OFF_GOAL));
746      lcd_print1("dist: ", 0);
747      lcd_print_at_cursor(buffer);
748      sprint16(buffer, (30 * ABS(IR_FRONT_LEFT(OFF_GOAL) -
749      IR_FRONT_RIGHT(OFF_GOAL)))
750      / IR_FRONT(OFF_GOAL));
751      lcd_print2("diff: ", 0);
752      lcd_print_at_cursor(buffer);
```c
    sleep(500);  
    }
    }
    }
    */
    * Tests how evenly the two motors works
    */
    void test_motors() {
        int i;
        float left_compensation = 1.0;
        int left;
        for(;;) {
            for (i = 100; i <= 125; i += 5) {
                lcd_print1(" ", 0);
                lcd_print2(" ", 0);
                left = (int) (left_compensation * i);
                set_motors(left, i);
                sprint16(buffer, i);
                lcd_print1("speed: ", 0);
                lcd_print_at_cursor(buffer);
                sprint16(buffer, motor_left);
                lcd_print1("ml: ", 0);
                lcd_print_at_cursor(buffer);
                sprint16(buffer, motor_right);
                lcd_print1("mr: ", 8);
                lcd_print_at_cursor(buffer);
                if (i == 125) {
                    left_compensation -= 0.02;
                }
                sleep(2000);
            }
        }
    }
    */
    * Test the values of the reflex detectors to calibrate the whit/black
    * thresholds
    */
    void test_reflex_detectors() {
        for(;;) {
            lcd_print1(" ", 0);
            lcd_print2(" ", 0);
            sprint16(buffer, REFLEX_FRONT);
            lcd_print1("fl: ", 0);
            lcd_print_at_cursor(buffer);
            sprint16(buffer, REFLEX_STICK);
            lcd_print1("st: ", 8);
            lcd_print_at_cursor(buffer);
            sprint16(buffer, REFLEX_BACK);
            lcd_print2("back: ", 0);
            lcd_print_at_cursor(buffer);
            sleep(1000);
```
/*
* Tests the encoder wheels
*/

void test_wheel_positions() {
    for(;;) {
        lcd_print1(" ", 0);
        lcd_print2(" ", 0);
        sprint16(buffer, ENCODER_LEFT_POS);
        lcd_print1("left: ", 0);
        lcd_print_at_cursor(buffer);
        sprint16(buffer, ENCODER_RIGHT_POS);
        lcd_print2("right: ", 0);
        lcd_print_at_cursor(buffer);
        sleep(1000);
    }
}

/*
* Drives and measures the encoder wheels
*/

void test_encoders() {
    bank3 static int sometime;
    bank3 static int i;
    for (i = 80; i < 125; i += 5) {
        lcd_print1(" ", 0);
        lcd_print2(" ", 0);
        set_motors(i, i);
        sprint16(buffer, i);
        lcd_print1("speed: ", 0);
        lcd_print_at_cursor(buffer);
        ENCODER_LEFT_POS = 0;
        ENCODER_RIGHT_POS = 0;
        sometime = soft_time();
        while (!passed(sometime + 2000)) {
            sprint16(buffer, ENCODER_LEFT_POS);
            lcd_print2("l: ", 0);
            lcd_print_at_cursor(buffer);
            sprint16(buffer, ENCODER_RIGHT_POS);
            lcd_print2("r: ", 0);
            lcd_print_at_cursor(buffer);
        }
    }
}

/*
* Tests the values the IR sensors gives for the puck
*/

void test_ir_sensors() {
    for (;;) {
        lcd_print1(" ", 0);
        lcd_print2(" ", 0);
    }
}
888  sprint16(buffer, IR_FRONT_LEFT(PUCK));
889  lcd_print1("1", 0);
890  lcd_print_at_cursor(buffer);
891
892  sprint16(buffer, IR_FRONT_RIGHT(PUCK));
893  lcd_print1("r", 5);
894  lcd_print_at_cursor(buffer);
895
896  sprint16(buffer, IR_STICK(PUCK));
897  lcd_print1("s", 10);
898  lcd_print_at_cursor(buffer);
899
900  sprint16(buffer, IR_BACK_LEFT(PUCK));
901  lcd_print2("bl: ", 0);
902  lcd_print_at_cursor(buffer);
903
904  sprint16(buffer, IR_BACK_RIGHT(PUCK));
905  lcd_print2("br: ", 8);
906  lcd_print_at_cursor(buffer);
907  }
908  }
909
910  /*
911  * function used to tune the timing of the firing mechanism
912  */
913  void fire_loop() {
914    bank2 static int sometime;
915    bank2 static int activation_time = 50;
916    RELAY_ENBL_PIN = 0;
917
918    // main loop
919    for ( ; ; ) {
920      ir_wait();
921      sometime = soft_time();
922      for (;;) if(passed(sometime+10000)) break;
923      sprint16(buffer, activation_time);
924      lcd_print2("a", 10);
925      lcd_print_at_cursor(buffer);
926      RELAY_ENBL_PIN = 1;
927      sometime=soft_time();
928      for (;;) if(passed(sometime+activation_time)) break;
929      RELAY_ENBL_PIN = 0;
930      //activation_time += 3;
931    }
932  }
933

D.5  intr.h

1  // ***************************** intr.h
2
3  extern bit has_passed;
4  extern unsigned int soft_tmr;
5  extern unsigned int soft_tmr_copy;
D.5. INTR.H

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

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

23  // Use above macros like this:
24  // sometime=soft_time();
25  // for(;;) {
26  //   if(passed(sometime+60000)) break;
27  // }
28  //
29  // The for loop will break after 60 seconds
30  // (64.5 seconds max as stated above)
31  // Of course yhe program would normally do other
32  // things than just executing a for loop while waiting
33
34
35  // initialise the soft timer
36  // uses TIMER1
37 extern void init_soft_tmr();
38
39
40  // initialise tone generator
41  // uses TIMER0
42 extern void init_sound();
43
44
45  // start playing a tune
46  // (non-blocking)
47  // tune is an array of chars in the following format:
48  // {note, time, note, time, ..... ,0,0}
49  // i.e. it contains notes and their duration, and always
50  // ends with two zeros
51  // format of note:
52  // bits 0-3 is the number of half tones from the base tone in each octave
53  // (so that if C was the base tone it would be 0, D=2, E=4, F=5, G=7,
54  // A=9, and H=11; cf counting the keys of a piano, including the black ones)
55  // bits 4-5 is the octave number from 0 to 3
56  // if bits 4-7 are all 1 the note will be quiet, i.e. a pause
57  // format of time:
58  // duration of note in units of 8 ms, so that 1 s is 125 units.
59  // Two equal notes (not necessarily with the same duration)
60  // will sound like merged to one longer note. Insert a small pause
61  // to avoid this (and subtract the time of the pause (one unit?)
62  // from that of the first of the equal notes
63  //
64  // a call will abort any already playing tune and start playing the new tune
65  // (i.e. playing the empty tune {0,0} will just abort previous tunes)
66 extern void play_tune(const char* tune);
67
68  // read the encoder positions from these vars

56
D.6 IR

D.6 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-opponent, 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 because 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 great.
// 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]
// To be absolutely safe against interference 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 interrupt routine
* (see below)
* bit defs for spi flags
*/

extern bank2 char spi_flags;

#define GOT_1ST 0 // used by interrupt routine to distinguish
// two consecutive start bytes
#define BAD_DATA 1 // set on failure to receive one ten bit set
// of data for one sensor
#define OVERFLOW 2 // set after rx overflow, cleared on reception of
// sensor 0
#define RECEIVING 3 // 1 when receiving set of 12 bytes over SPI else 0

//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 before 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)
# D.7 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();

# D.8 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 <--> fourth motor control pin**
**//**

59
D.9. SPRINT.H

// 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

void setpwm(char left, char right);
void setdir(char left, char right);
char s2pwm(signed char val);

D.9  sprint.h

// ************ 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);
Bibliography


