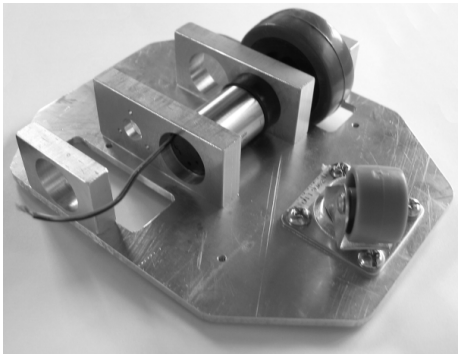
Robotsheep Attraction ’Lambert’



Sebastian Edman[1]
Storängsvägen 22, 4tr
184 31 Åkersberga
Sweden

Henric Thisell[2]
Leiresväg 100
443 51 Lerum
Sweden

Thorsten Gernoth[3]
Eulenstraße 57
22765 Hamburg
Germany

G. Puneel Kumar[4]
S/O G.V.V. Subbaiah
D.No: 10/154-B6-8
Rajendra Nagar II line
GUDIVADA-521301, A.P.
India

V.V.Madan Kameswar[5]
S/O V.Prabhakar
MIG-99, A.P.H.B.Colony
NELLORE-524004, A.P.
KALLURUPALLI
India

10th June 2003

[1]it2edse@ituniv.se
[2]hthisell@yahoo.se
[3]t.gernoth@tu-harburg.de
[4]it2guku@ituniv.se
[5]it2veve@ituniv.se

Abstract

The idea was to make an interactive attraction for the museum of technology in Stockholm. The idea is based on collective behavior and some elements around the behavior. The basic algorithms are primalily based on Craig Reynold's work on boids wich are very well known. The actual attraction consists of a green field roughly 8 to 10 square meters with six robots. Five sheep and one combined wolf or sheep dog. The sheep are moving in a way to show collective behavior and will run away from the wolf and try to keep away from the dog (especially when barking). The wolf/sheep dog is controlled by the audience and when a wolf, the task is to separate the sheep and 'kill' them off one by one. This is achieved when the wolf is very close to the actual singled out sheep. The task of the dog is to gather up the stray sheep with the help of barking.

Contents

Chapter 1

Introduction

To get what this project was all about it is quite good to read this chapter as it describes the history why this project started and the final idea about what this project is aiming at to become.

1.1 History

In the masters program 'Intelligent Systems Design' there is a course where a live project should be conducted. This project should either be done for a company or result in something commecial. When this course was about to begin we had recieved a wish from the Tekniska Museet in Stockholm to make something intelligent for their exhibition starting at May 2003. They had some initial ideas about flocking or collective behavior. We were five persons that were interested in this project so we started to come up with ideas.

Some of the guidelines we were thinking of was following.

Interactivity to capture the audience and make it more fun to go to a museum.

Attraction value the final attraction should be formed so a large amount of visitors can view or visit it.

Runtime the attraction must be able to function for a whole day.

Robustness the attraction must be able to withstand the cruelness of the visitors.

1.2 Initial Idea

There are six robots, five sheep and one wolf/dog. The sheep are moving in a way that show collective behavior and the wolf/dog is user controlled. The task for the user when controlling the wolf is to separate the sheep and then 'eat' (disable) them. When all sheep are eaten, the user switches to the dog by moving into the farm corner and start bringing them back to life again by barking close by. To switch back to the wolf the user has to go into the forest corner of the field.

1.3 Thanks to

Jonas Brandén for making the aluminium platform for the robots.

Jimmy Eiterjord for always being around and answering all kinds of wierd questions.

Jason Libsch for accepting the task as main designer.

Chapter 2

Off Board Computer

At the beginning when the robots were considered, sensors, electronic and hardware was discussed. Not many of these ideas were about how the user would control the wolf/dog. There were some search for joysticks, buttons and there was an idea to have a micro controller built in some kind of panel. Because of the complexity of such a device and most important of all, the robustness of such a device, the choice to move to an off board computer for vision processing was most welcome. This meant that the interface to the attraction could be just a simple game pad or similar device which then would be connected to the computer. The web camera, the game pad and the transmitter are briefly explained below.

2.1 Web Camera

We were using the ADS Pyro web camera for this project, it is a firewire web camera that can deliver 640x480 uncompressed at a frame rate of 30 frames per second. In order to get it to work we had to use the libdc1394 library and tweak some parameters. All automatic calibration of the camera were turned off, otherwise the green field were calibrated to almost gray. The brightness, exposure and white point values were set to acquire the best color ranges as possible with the lighting armature. It was necessary to run in DMA mode otherwise lots of frames were lost. At a frame rate of 7.5 frames per sec, the read of the image, the YUV-conversion and the display took about 7% of the CPU of a PIII-666MHz. But since the quality of this camera was really awful it was exchanged for the Philips Webcam Pro.

The Philips camera is only USB v 1.1 which means 12 mbit capable, while the Pyro was 400 mbit capable. This means that Philips uses a compression scheme that will degrade the image. Though the compression does not add as nearly as much artifacts or interference to the image that the Pyro had. The Philips camera can run 640x480 at 15 frames per second and already deliver the correct image format (YUV4:2:0). This made it an excellent choice and it is also very easy to control manually, like setting the white point, brightness and so on. The vision processing is described in chapter 8.

2.2 Game pad

In order to get a good user interface to this attraction, a game pad is connected to the off board computer. This game pad should have four direction keys (accelerate, left, right

and maybe brake) and at least one button to bark when being a dog. However finding a game pad that has only five buttons is quite hard to find unless you build one yourself. The suggestion is to buy a commercial one and mark the buttons used, all other buttons should be disabled or simply removed if possible.

2.3 Transmitter

To enable the vision processing to send all the data to the sheep there must be a transmitter connected to the off board computer. This transmitter is most easily connected to the serial port of the computer with just a line buffer circuit since there are very few transmitters that accept ± 15 Volts. This small circuitry needs a 5Volt supply which can most easily be taken from the computer power. The transmitter used is actually an transceiver and is the same that is fitted into every robot (see section 3.4.1 and 4.3.2).

Chapter 3

Robot Hardware

The most important about this robot construction is that the hardware must be robust. It must withstand a one and half year of beating and rough play from the audience, people at a museum can be all but nice to robots, especially if they can interact with them [12]. Other design issues involved a long running time and of course sensors so that all robots can see each other.

Each section (3.1, 3.2 and 3.4) begins with some history about the topic in question. This will give a more complete background to some decisions and there is generally not anything written of technical importance that describe the final robot in these history sections.

3.1 Platform

The first idea was to make the platform in Plexiglas, this however was later changed to aluminium because of the fragile nature of Plexiglas and the issues when mounting the motors as described in sections 3.1.1 and 3.2.4.

3.1.1 Platform History

The first piece that resembled a platform was made of Plexiglas, 4mm thick. This was because it was heard that Plexiglas is easy to work in. However it turned out that this was not the complete truth. The Plexiglas was quite easy to saw in and make the form and size wanted. But it was very hard not to create any cracks or weak spots in the material. Also the thin Plexiglas flexed quite a lot when the motor was mounted on it, so that more cracks would arise from the vibrations and eventually break the platform in two.

This made us move to a 6 mm thick Plexiglas. The idea was that it would better withstand the flexing and hold better than the previous 4 mm. The mounting of the motors was still an issue though as described in section 3.2.4 and because of that an alternative material was sought for.

The choice to move to aluminium was not happening over night. The idea grew as people started to talk about the properties of aluminium. The actual change did not come until we found a person who could manufacture them for us. As soon as this person was found the work of cadding all the wanted pieces started. The design of these pieces had grown over time so it was more or less already decided how they

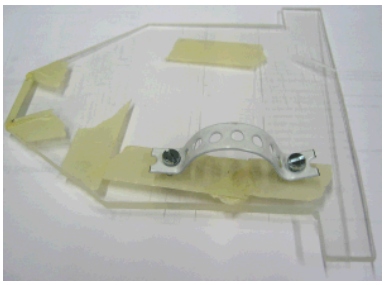


Figure 3.1: Image of the first plexiglas model.

should look like. The workshop person found did however not seem to be able to make these parts. Therefore the cadding was slightly simplified and handed to another workshop. This change of workshop resulted in a very long setback for the project and contributed greatly to the project not being finished on time.

3.1.2 Final Platform

The final aluminium platform consists of five parts. The main part is a 4mm thick aluminium plate which is 13 by 16 cm. The corners of the platform are rounded and the two big holes are for the wheels. The reason to have holes instead of just a cut out area is that now it is possible to mount the bearings on the outer side of the wheels as an extra weight carry. This is more closely described in section 3.2.5.

For the third wheel, the caster wheel, there are four holes with threads so that the wheel can be easily mounted. There are also eight holes where the motors with gearboxes and bearings should be mounted and an additional four holes for the PCB.

3.2 Power and Driving force

The robots should be able to run for at least 8 hours without the need of changing batteries. At first this sounded very long for such a robot and doubts arose whether it was possible or not. But with a few calculations it showed to be possible. These calculations are shown below.

The positioning of the motors was also a big problem that is described below, since the robots are going to move on a restricted area, it is good if they are not too big. Therefore it was decided on a quite uncommon motor configuration (see section 3.2.5).

3.2.1 Battery Decision History

Batteries exist in a wide variety of sizes and capacities. It was clear that a voltage between 6 and 9.6 Volts was required, this is the voltage needed for the motors to be able to drive a robot of our size. The batteries should also be able to provide 5 Volts to the electronic circuitry. High voltage is always good in order to get the extra 'push', but the cost for each additional battery is substantial and the weight would also increase. Later it showed that the weight and the cost would be too high with 9.6 volts so the preferable value was 6V.

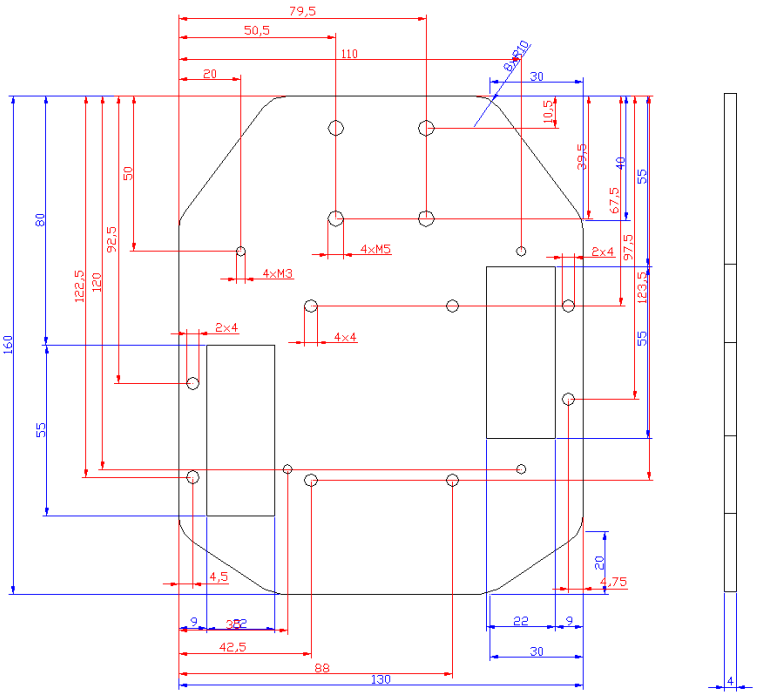


Figure 3.2: The CAD over the platform.

A battery with a high capacity-per-weight ratio and the Voltage of 6 to 9.6, would weigh approximately 0.2 to 0.5 kg and the capacity would range from 1900 mAh to 6800 mAh as can be seen in table 3.1. A very price worth battery was found from GP-Batteries. It had 3700 mAh and it was decided to use a 7.2 volt battery in order to get the extra 'push' to the motors and to run the 5 Volt regulator (7805) powering the processor according to the data sheet values.

Battery	Type	Volt	Capacity	Weight	mAh/g
GP-AA	NiMH	1.2	1900 mAh	26 g	73
GP-C	NiMH	1.2	3400 mAh	79 g	43
GP-D	NiMH	1.2	6800 mAh	170 g	40
GP-380AFH	NiMH	1.2	3700 mAh	53 g	70

Table 3.1: A list of different common and uncommon batteries. The one at the bottom of this list was chosen for this project.

3.2.2 Current Calculation History

With a battery of approx 3.7 Ah and a running time of at least 8h means that the robot must draw less than 470 mA. With two motors and electronics, the motors must draw max 200 mA each and the electronics max 70 mA. These values must not be treated as strict maximum values, because of the nature of sheep and the audience, the robots will

not move all the time and the high current consumption is when the robots accelerate and turn. Since this is not going to happen continuously, the limits above will fit quite well as average values.

3.2.3 Motor History

The choice of motor was between the manufacturers Portescap and Micro-Drives. Both make good motors that met our demands in quality and lifetime. The Portescap motors also had a good simulator at their Swedish local retailer web page. With this simulator and the presence of a local retailer, the choice fell quite naturally on the Portescap DC motor series, called Escap.

The specifications we were looking for was a 6 to 9.6 Volt motor with enough torque to move the robots at a speed of 1 m/s. Also the current consumption must not exceed 200 mA (see section 3.2.2) at average consumption.

To get an idea what motor was needed, some calculations were made. It involved how much torque is required to get an acceleration of approximately 1 m/s^2. This was calculated on a 4 cm diameter wheel and an approximate weight of 1 kg. This resulted to a torque of 20 mNm (formula 3.3).

$$F = ma \tag{3.1}$$

$$\tau = Fr = mar \tag{3.2}$$

$$\tau = 1 \times 1 \times 0.02 = 20mNm \tag{3.3}$$

The other calculations were to find the torque that was needed to move at all and also a calculation of how much torque that would be needed to start spinning. But both of these were highly estimated and just for us to get an idea of how much torque is reasonable to have since none of us had this experience from before.

To get an idea about what gearbox was needed, a calculation from the Portescap main catalogue [7] was made. Dividing the maximum rpm input of the gearbox with our wanted rpm gives us the ratio wanted (see formula 3.6). In this case a 20:1 gearbox is a good choice.

$$i \leq \frac{n_{max}}{n_{ch}} \tag{3.4}$$

$$n_{ch} = \frac{v}{2\pi r} = \frac{50 \times 60}{2 \times 2\pi} = 238.7 \tag{3.5}$$

$$i \leq \frac{5000}{238.7} = 20.94 \tag{3.6}$$

The first motor that met most of these specs was escap 16N28 with a 27:1 ratio gearbox. This motor was a great candidate. Though there was some distance left to the magical 200 mA limit and it felt like a good idea to overdo it somewhat. A second motor was looked at, the Escap 22N28. The 6V version of this motor really looked promising and was still under the magic 200 mA limit at a torque of 20 mNm. The Escap 22N28 motor was declared winner and contact with the Swedish retailer was initiated.

After contact was initiated we were told that there existed a box full of Escap motors that we could use for this project. After locating the box these motors turned out to be

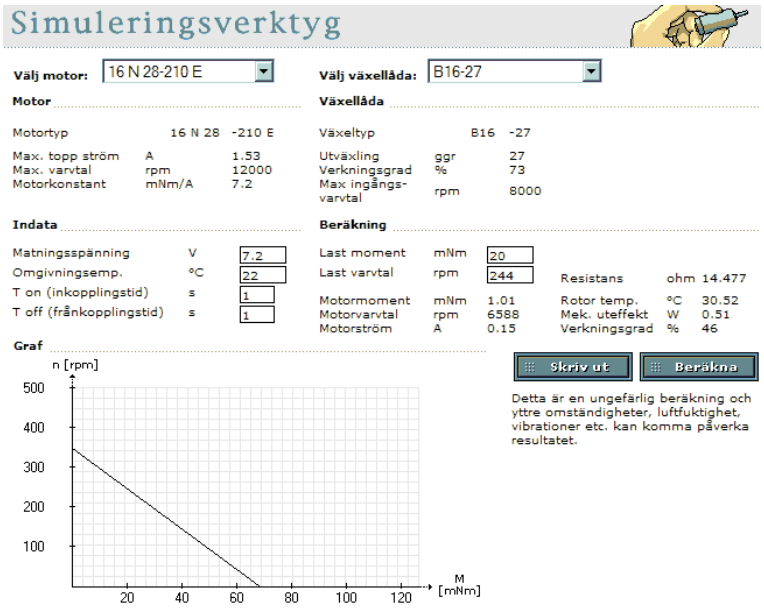


Figure 3.3: Simulation of the Escap 16N28 6V motor with the B16 27:1 ratio gearbox.

the 9V version of exactly the same motor we wanted. New simulations was done with the new motor right after this.

Both gearboxes mentioned above is of the type planetary. There exists gearboxes with the output shaft rotated 90 degrees. This would be very good since it will allow us to place the motors basically as close as possible and thus reducing the robot size. There are also some gearboxes that has the output shaft not in line with the motor shaft but at an offset. This would also decrease the size of the robot since we can have the two motors lying beside each other but still having the wheels homogeneously placed. However these gearboxes are not in Portescaps 'speedyline' [8] and there was no retailer or factory which had twelve of the ones wanted. Thus the waiting time for these was far to big and the planetary ones were chosen.

3.2.4 Motor Mounting History

The first obvious problem was how the motors with gearboxes should be mounted on the platform. The initial idea was to use a 1 cm wide zinc band with holes in and fasten with scews (see fig. 3.5a). This will make the motor sit firmly in the direction the robot is moving. To prevent the motor from rotating and from sliding sideways, a plate could be glued to the base platform in front of the motor so the three holes on the gearbox can be used (see fig. 3.5b).

The problems showed when this was mounted together, the Plexiglas flexed under the motor and the glue would probably not hold forever with tensions like that. This made us think to move the fastening points for the motor closer to the motor so that the arm of force became smaller. A better solution might be to bend a pole around the motor (see fig. 3.5c). This however has also some drawbacks. It is very hard to bend a pole in such a way that it has contact everywhere.

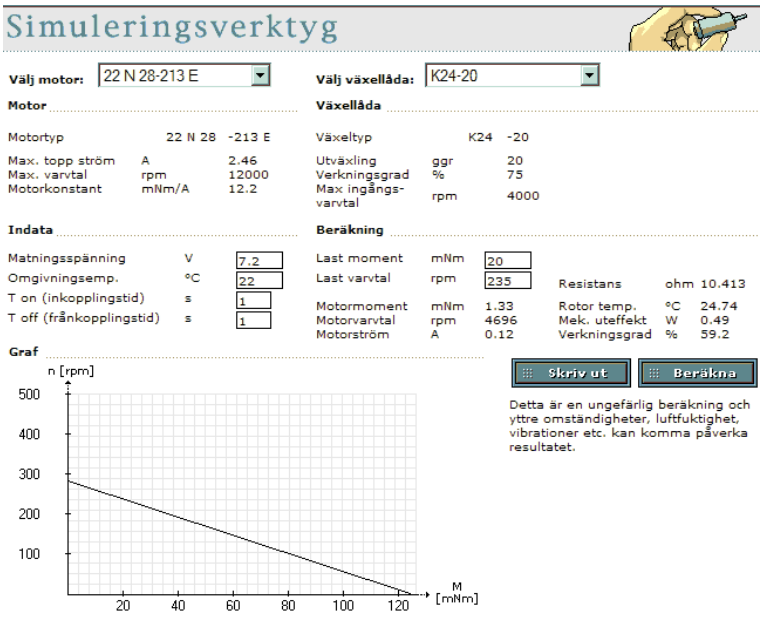


Figure 3.4: Simulation of the Escap 22N28 9V motor running 7.2V with the K24 20:1 ratio gearbox.

3.2.5 Motor and Wheel Configuration

The motor is tightly connected to the gearbox, the wheel axle is mounted to the shaft of the gearbox with a stop screw and the wheel is firmly fitted to the wheel axle. The bearing is placed in the bearing holder and the wheel axle is placed in the bearing. Everywhere where things can start to shake apart, it is important to use lock tight or similar fluid which increases reliability.

The motors with the gearboxes, axes, wheels and bearings should together with the caster wheel be placed in a triangle to get the pest performance (see fig. 3.7a).

The size of the motor with the chosen gearbox was considerably larger than wanted. With the motor configuration described above that is about 9.5 cm long, the width of the final robot would be at least 20 cm. Such big robots on a 10 m^2 field does not leave much room. Therefore the final placement of the wheels was shifted as shown in figure 3.7b.

Quite early in the project the idea came up to run the robots backwards, with the caster wheel in the back because that would make them move more like a sheep with a trailing behind rather than a sliding front. So the final model was designed to have the caster wheel in the back.

3.2.6 Motor and Bearing Holders

The platform consists of a plate with several drilled holes. Eight of these holes are for the motor holders and the bearing holders (see fig. 3.8), each of these is mounted with two M4 screws.

The two motor holders are placed at the beginning and end of the two motors.

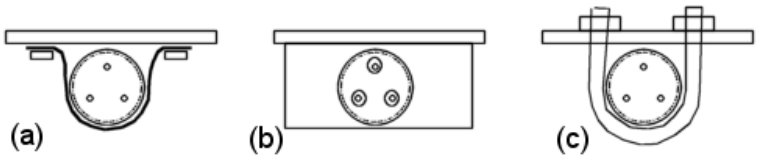


Figure 3.5: Three different ways to mount the motors with gearboxes. (a) to use a 1 cm wide band with holes in and fasten with screws. (b) to have a plate in front of the gearbox and use three screws to mount the gearbox. (c) to bend a pole around the motor and gearbox and fasten it with nuts from the upside.

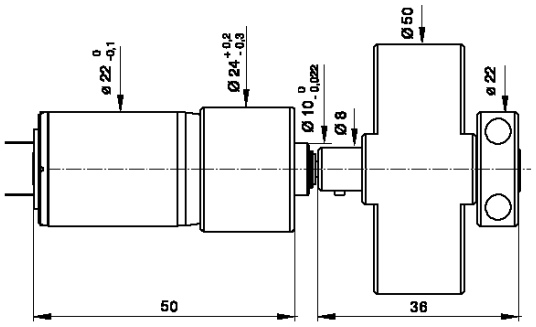


Figure 3.6: The motor connected to the gearbox, which is connected to the wheel axle, the wheel and finally the bearing.

Each motor holder holds one gearbox and the back of a motor as can be seen on figure 3.9. The gearbox is mounted in one of the motor holders with three M2 screws to avoid rotation and sideway sliding. The bearing holders are mounted at each side of the platform and are made to fit the bearings with precision so no other fastening is necessary.

3.3 Discarded Ideas History

At the initial stage of our planning, different possible techniques for sensing was looked at, which could be used for the robots. Some of the primary difficulties that was faced concerning the sensing techniques. A good sensing technique was wanted, which could be used by every robot to know about the other robots on the field. With these sensing techniques it is very possible that the robots may collide with each other. So we wanted to have a kind of sensing which allows the robot to recognize about their collision and react fast. In this section we will describe about the possibilities we thought of and the problems with them.

3.3.1 Sensing for Interaction with other Robots

Some of the commonly known sensing technique used in robotics is infra red communication. But this technique can be used in various ways. Infrared sensor circuits have many varieties. The simplest uses an IR LED to deliver the light and an IR sensitive

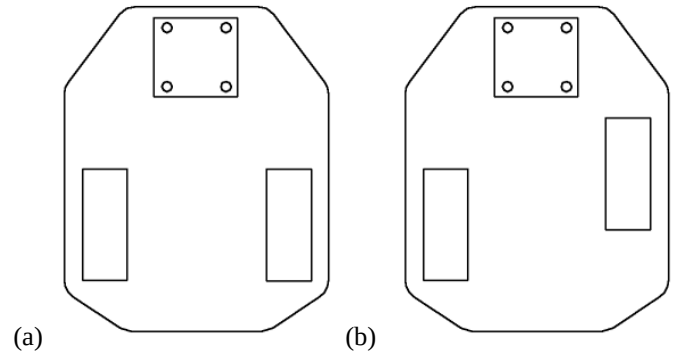


Figure 3.7: The position of the wheels. (a) the wanted position. (b) the final position

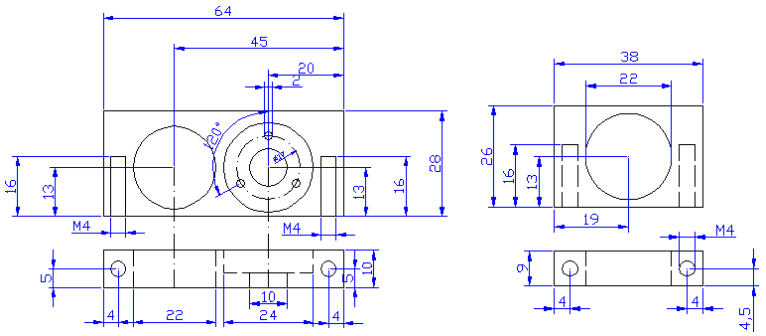


Figure 3.8: The CAD over the motor holder (left) and the bearing holder (right).

photo transistor to see it. The photo transistor turns on when IR light hits it, whether from the LED or ambient sources or reflected. More sophisticated IR usage involves pulsing IR light at a known frequency, and using a detector that is attuned to that frequency. There are some circuits that tunes the detector to a specific frequency. Each is sensitive to IR that has been pulsed at that given frequency. This encoding of the IR helps to mask out external sources (ambient light) that might have been in the way of information we really want, specifically how close something is or the receiving of coded data. There are some more IR sensors which come in all in one packages and provide range information, in either analog or digital form, or simply be a trigger point when something is within adjustable range.

We thought of using some kind of IR-sensors. For certain interval of time, IR LEDs on each robot should sent out unique information. Based on the information the surrounding robots get it would be possible to analyze if a robot is a sheep, wolf or dog. Since light intensity decreases with the square of the distance it should be possible to estimate the distance to the surrounding robots.

So the above details explains about the IR sensing and some of the above ideas though fit into our requirements and would make the robots really autonomous. Some of the main reasons why this idea was discarded is the bad reliability of IR sensors for distance measurement. So taking into consideration various other factors like availability, affordability, complexity while designing, programming to extract information

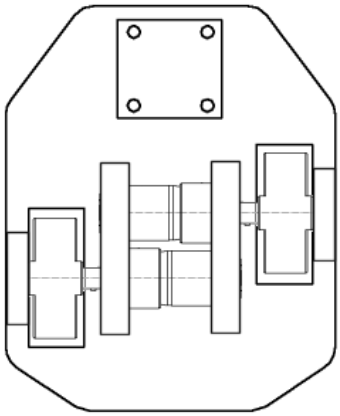


Figure 3.9: The two long holders in the middle are the motor holders and the two smaller on the sides are the bearing holders.

from them, reflections, proper placement of them etc., we wanted to shift our design from using IR sensing to using a video camera.

3.3.2 Collision Detection

This is also one of the important factors to be considered. Though a proper sensing technique used for communication should try to avoid collisions, most of such systems are not so reliable. So as a precautionary measure, we initially wanted to have a kind of bump sensors which could sense the collision and let the robots move away accordingly based on the information they get. Though in this case it is not at all complicated with either the electronics or the software part, it is much difficult with their placement and mechanically fastening them. And since the vision system could also be used for collision avoidance we tried not to use them. Since we have not tested everything together yet, we are not sure if we can avoid them completely or not. But we are ready to add some kind of collision detection in case the situation compels us to do so.

In the both the cases there was a possibility to use sonic sensors, ultra sonic sensors or some other techniques. But we spotted some how similar problems like with the IR sensors. Though by putting more efforts it might be possible to use a sensing technique other than a camera (which is also not an ideal/best fitted sensor). We realized that a video camera is much suitable to our project in many point of views like complexity, time frame for the project , availability of spares etc.

3.4 Other Peripherals

Aside from the mechanical construction there exist a few other main functions of the sheep. The first one can think of is sensors. The robot must be able to see the other robots and act accordingly. here the sensor information is received by the FM receiver. There is also an audio speaker and a processor to run all this.

3.4.1 FM Receiver

Since the position and direction of all robots is broadcast over FM radio a receiver is fitted on each sheep. Actually it is a FM transceiver but only the receiving part is used in the robots. It is made by Nordic and is running on the 433MHz band, it has 2 channels and it is run on 9600 baud. It is very nice to work with since it can run the RS232 protocol directly and the evaluation kit includes all components pre mounted. It also draws only 11 mA when in use, which is good for this project.

3.4.2 Audio Speaker

In the middle of the project it was suggested to us that a speaker should be added. This was thought of and the result was to use the current processor connected to a DAC and then an audio amplifier. The 0.2W, 8 ohm speaker chosen has a register ranging roughly from 300 Hz to 11 kHz. There are also ideas to use computer speakers.

3.4.3 Processor

The processor used for this project is a microprocessor running at 16 MHz. It is an AVR128 made by Atmel and it has 128KB internal flash memory and 4KB internal RAM. It has some useful functionality like internal UART, two PWM modules, internal timers which can be used for timing and sound creation. The gcc compiler can be used to compile programs to it. This is a huge advantage since our current simulator runs on a gcc platform and it means that it is very easy to port the code.

3.5 Specifications

The size of the platform is 13 by 16 cm and the height with everything mounted will roughly be 8 cm. But the casing is the one deciding the size, since the horns or nose will extend the size a bit, and a good estimate of the final robot is 15 by 18 cm and 12 cm high. The weight is estimated to about 0.8 to 1.0 kg.

It is running on a 7.2V battery with 3700 mAh, this will allow the robot to function for at least 8 hours. Maximum speed is 60-80 cm/s with an acceleration of about 1 m/s^2 and the turning radius is technically close to nothing. But practically set to about 0 cm (measured from the inner wheel) so that the caster wheel in the back will not rotate that much.

Each motor will consume about 150 to 200 mA and the electronics will approximately consume 60 to 70 mA. There is no on board sensors, instead the position and direction of all robots will be acquired from an off board computer that monitors all robots with a camera. The resolution of this camera is somewhere between 0.5 to 1.0 cm (because of the YUV format used).

3.5.1 Design

The design of the robots are not yet decided. It has been handed on a person studying Art & Technology so the expectations are quite high.

Chapter 4

The Electronics

The Electronic layer which acts as the interface between the physical hardware and software layer of each robot is designed and developed as a simple, reliable and low power circuit. This section of the report explains all about this circuit. To meet the demands of our robots, we tried to frame certain basic requirements. After research on different possibilities for the robots, different parts are chosen, keeping in mind, the availability, affordability, limitations and reliability.

4.1 Requirements

There were some basic requirements which had to be fulfilled by the electronic circuit. Our primary goal of the whole system is to have a robust system which could be controlled easily. Some of the required, basic qualities are:

Reliable, Simple and Cheap: We wanted to have a reliable, simple and economic circuit platform.

Low Power Consumption: Since the robots were planned to operate for about 8 hours continuously with battery power, this feature is quite essential. The whole circuit should not draw more than 470 mA as described in section 3.2.2.

Same Circuits: We wanted to have the same circuit platform for the sheep and for wolf/dog robots. This helps in reducing the complexity and at the same time enables an easy way to replace the wolf/dog with a sheep robot with almost no change.

Timing Constraints: There were some timing constraints which had to be fulfilled by the electronic circuit. For example, the chosen transmission system must be fast enough to broadcast the information about the position of every sheep in the herd from the off-board computer to the whole herd. The audio circuit must work with the chosen sampling frequency.

Using ATMEL ATMega 128 Board: Since we were using the Atmel ATMega 128 Micro controller [2] in hand before we started with the rest of the circuits, we wanted to use it for our system.

4.2 System Description

The electronic circuit consists of several different parts and they are:

Motor Control Circuit: To control the motors and separate the power supply to the motors, a small circuit was needed. The motors are controlled using a pulse width modulation technique.

Receiving Section: Every sheep robot receives information about other sheep robots positions from the off board computer. And the wolf/dog receives their steering commands via this section from the off-board computer. This need is fulfilled with an RF transceiver working on FM.

Audio Circuit: This circuit is mainly to add some attractive features to each robot. We wanted each robot to imitate sheep sounds for certain interval of time. The wolf/dog robot should imitate their respective sounds. So an audio circuit was added to play audio samples.

Behavior Switch: Since we wanted to use the same circuit for sheep and wolf/dog, a switch was needed to change the behavior. A small electronic switch will send a signal to the processor so thet the processor knows which roll this robot should play.

More details about each part and their functions are described in the following sections.

4.3 Discussion of Parts of the Electronic Circuit

The electronic circuit can be described shortly as shown in the block diagram (see fig. 4.1). How we met the requirements with these parts is also described along with their specifications in the following section.

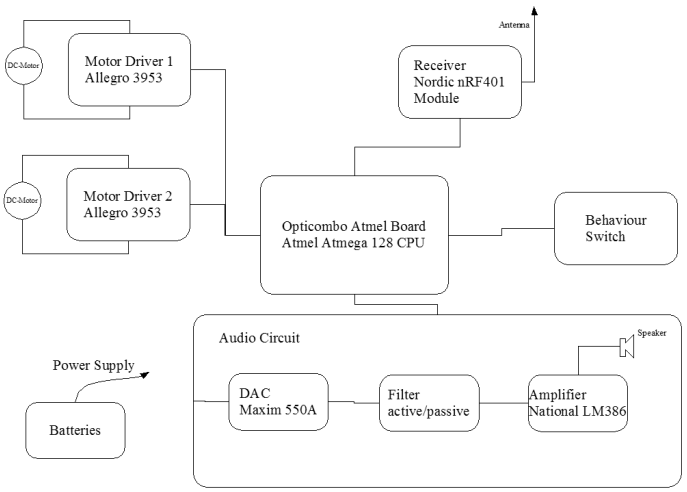


Figure 4.1: Block diagram of the electronic circuit

4.3.1 Motor Control Circuit

The Atmel ATMega 128 processor like most other processors can not pass enough current and voltage to spin the DC motors. Also motors tend to produce a lot of electronic noise like spikes on the supply signals when powering the motor. An extra circuit for controlling the motors was therefore required. We were examining several different approaches.

The first approach we were looking at was an H-Bridge which is a common circuit for driving a DC-Motor. An H-Bridge can be built very simple with transistors and diodes. With an H-Bridge it is possible to separate the motor supply from the logical signals to turn the motor on or off. There are two logical input signals and two output signals. The outputs are driven by a separate supply voltage which is sufficient to drive the DC-Motors. The outputs are connected to the supply of the motors. When one logical signal is raised the corresponding output is going high too and the other output is going low. The motor is rotating in one direction. If the other logical signal is raised the opposite happens and the motor is rotating in the other direction. The speed of the motors can be controlled with pulse width modulation of the logical inputs.

There are also H-Bridge ICs available. We were examining several since using an existing IC would reduce the complexity and the required components for the motor control circuit.

There are Full- and Half-Bridge Motor Driver ICs available. Since we wanted to have the possibility to drive the motors in both directions we decided for a Full-Bridge Driver. With the motor resistance of 10.3 Ohm and the supply voltage of 7.2V the absolute maximum current drawn from the power supply is 0.7A.

There are motor drivers which control the current and there are motor drivers which control the voltage. Voltage control is more used with stepper motors and to control the current is the simplest way of controlling our DC-Motors. Therefore we decided to use motor drivers which control the current. There are also Dual Motor Drivers which can control two motors. This would have simplified the design, but there was no one available from Elfa which satisfied our requirements.

The motor driver we finally chose was the Allegro 3953 Full-Bridge PWM Motor Driver[1]. It is designed for pulse width modulated current control of DC-Motors. The load current is proportional to the motor speed and the motor resistance. The maximum continuous output current is ± 1.3A which satisfies our requirements. It is also possible to limit the peak load current with a reference voltage and an external resistor. We limited the maximum load current to 0.1A. There are several ways of controlling the direction and speed of the motor with a pulse width modulated signal. We decided to pulse width modulate the enable signal of the motor driver (speed) and use the phase for direction control of the motor. PWM control of the phase signal, which is switching the polarity of the motor supply, could be more linear since the PWM control of the enable signal could produce discontinuous currents at low current levels. We decided to use PWM control of the enable signal since when changing the direction of the motor the kinetic energy and load inertia stored in the motor would be converted into current that charges our supply batteries. We also thought it is easier to control the motor with pulse width modulation of the enable signal. To make the motors stand still pulse width modulation of the phase would require that the low and high period have exactly the same length. The motor driver can be used in a slow and fast current control mode. We decided for slow current control mode since it also prevents our batteries from being charged by the kinetic energy in the motor and the load inertia when the direction of the motors is changed abruptly. The driver has the ability to stop the motors abruptly.

This signal is connected to the processor as brake signal.

The Allegro 3953 was a good choice. The control of the motors is working well and only a few extra components were required.

4.3.2 Receiver

We examined several different available receiver ICs. The basic requirements are that it should be possible to broadcast the information about the position of the sheep and that we can send the steering commands to the wolf/dog. The information must be send at least 15 times per second since the image processing updates the information about the position of the sheep 15 times per second. The IC should also consume as little power as possible. The data should be transmitted within a range of about 4m. We decided to use a frequency modulated radio signal since it is commonly used, reliable, cheap and simple to handle. We decided to encode the position of the sheep and the steering commands for the wolf/dog into one data package so we can broadcast the same data package to all robots and only one channel is required. The required throughput for sending this information is roughly 600 byte per second. The receiver IC should also be easy to connect to the Atmel ATMega128 processor in a way that no extra decoding of the signals is required. For example it could use a RS232 like protocol that we can connect to the USART module of the microprocessor.

The first IC chosen was a Telecontrolli receiver chip. It fulfilled our requirements very well, but since it was hard to order this chip we looked for another solution. We decided to use the Nordic nRF401 transceiver[6]. There is an evaluation kit for this chip available. The evaluation module is equipped with an antenna and an oscillator. This simplified the design a lot since there was no need for external components. It is possible to connect the evaluation module directly to the RS232 interface of the processor. Another advantage of this module is the very low power consumption. The typical current drawn is 11mA. The maximal throughput is 20 kBit/second. Two channels can be used with this transceiver. This is more than sufficient for our requirements.

This transceiver module is working very well. We did not run into any problems so far.

4.3.3 Audio Circuit

The basic requirements for the audio circuit are that it should be computational cheap for the processor to play a sound and that the audio circuit uses as less power as possible. The audio circuit is not a necessary part of the robots but it makes the robots much more entertaining.

We were evaluating two different methods to make the robots sound. First method we evaluated was using the pulse with modulation feature of the Atmel microprocessor. The audio signal should be played by using pulse width modulation. With a counter the pulse width modulation output should be switched on for a time span witch is proportional to the sample value which should be played. The pulse width modulation frequency must be significant higher than the audio signal sample frequency. An output filter should then smoothen the signal and remove the high frequencies (the PWM carrier signal). This works similar to a Delta-Sigma DAC.

The second approach we discussed was using a separate digital to analog converter, which could be connected to the SPI interface of the Atmel microprocessor. We decided to go for the second approach since we expected it to be simpler and to use less extra components.

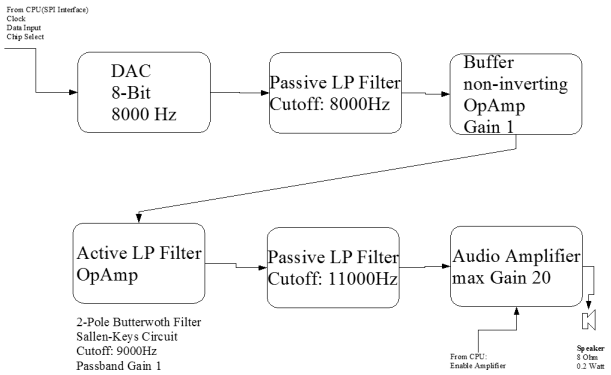


Figure 4.2: Block Diagram of the Audio Circuit

We chose an MAXIM MAX550A 8-Bit DAC[4]. This DAC is very simple because it requires just a few extra components. It consumes very little current, which is very important. It is designed to be used in battery powered applications. It has an SPI interface which can be connected directly to the Atmel Microprocessor. The DAC consists of an R-2R ladder network that converts 8-bit digital inputs into analog outputs in proportion to the applied reference voltage. The reference voltage we connected to the DAC is 5V. This is the maximal analog output voltage. The DAC is dual buffered and knows several commands to convert a digital signal. Since we want to do it as simple as possible we chose that when the chip select signal is raised the DAC should convert the digital signal into a proportional analog voltage. This allows us to always send the same command to the DAC and reduces the required connections to the Atmel microprocessor.

Since the outputs are un-buffered an energy pulse is coupled into the DAC output when the chip select signal is raised. We are using a passive low pass at the output to suppress that pulse. The analog signal from the DAC is then fed into a buffer since the DAC required a high resistance load at the output. It was hard to realize a high resistance load with the following active low pass filter. The buffer is therefore for decoupling the signal.

The operational amplifier we chose for the buffer and the active filter is an ST LM358[11]. It is a standard operational amplifier and commonly used. This IC was not the best choice since it is mainly designed for dual supply and requires a more complex circuit for single supply. The buffer is non-inverting. The following active low pass filter is used as an anti-image filter. It should filter out the image frequencies which are created by the digital to analog conversion. We decided to build a 2-pole Butterworth filter. 2-Pole is the maximum order which is possible to get from one operational amplifier. A Butterworth filter is used since we do not want to have ripple in the pass band. The cutoff frequency is 9 kHz. The filter circuit is a Sallen-Key circuit, which is the most simple and reliable one. Before the signal is fed into the audio amplifier it is filtered by another passive low pass. With this passive low pass we can achieve better sound quality.

The following amplifier is a National LM386 audio amplifier[5]. It has very low power consumption, which was the main reason to choose this IC. We added a small transistor switch to be able to shut down the audio amplifier when it is not used. This

can be done with a signal from the microprocessor. We are achieving a maximum gain of 20 with this audio amplifier. The amplifier can achieve higher gains but then more extra components are needed. This gain seems to be sufficient for our application. The speaker which is currently used is a 0.2 Watt 8 Ohm speaker.

The circuit (figure 4.2) is working well and the power consumption is acceptable (~40mA). We can also shut down the audio circuit when it is not used. This keeps the average power consumption low. We were facing some problems with the operational amplifier we are using. This operational amplifier is not optimal for use with single supply.

Other problems were decoupling everything and reducing the electrical noise which is generated by the chips. In the end we needed some extra components and therefore using an external DAC was probably not the best choice. By using the PWM features of the Atmel microprocessor we could probably have reduced the complexity. Audio signals can also be represented in an efficient one bit resolution. This could have been used with the first approach.

Chapter 5

The Simulator

To understand how the robots would move when different algorithms were tried, a simulator was programmed. The simulator is only for this purpose and it was only intended to see how the algorithm behaved and not including everything. Therefore it makes some simplifications and assumptions.

5.1 Assumptions

Since the purpose of this simulator is to see how the boid algorithm behave when approached, there is no simulation of collisions, no actual forces or natural laws. The simulated parts in this simulator is first the interface to the user. Second, the interface to the program and third, the translation of motor speed to movement.

If the purpose of this simulator instead was to use as a genetic algorithm training playground, this simulator would not be sufficient, because of the nonexistance of laws that can be bent or used to its advantage, the world is simply to strict and to non informative. Of course learning can be used in this simulator but things as braking or accelerating cannot be optimized in this simulator, since the simulator is not that flexible.

5.1.1 Collisions

In the simulator itself there is no collision detection, the idea was to program the robots so they do not collide at all. So if a collision will occur, the algorithm or the collision prevention code should be modified instead. (also see section 6.2.5)

If collision detection would be implemented, a simple way of doing it would be to just check if any other robot is within its radius. If this would be the case then both robots should halt directly. But then to make them slide against eachother is a lot more difficult and has nothing really to do with this project. The purpose of the simulator was to get a simple and fast idea about how the algorithm behaved and not to explore the physics of nature.

5.1.2 Braking

This is a problem that is hard to estimate. What is the braking force when the motor is turned off. Will it continue for another 5-10 cm or will it continue a whole meter?

Since none of us has this experience this is very hard to estimate. Because this definitely inflicts problems, since there is no real good way for the robot to know its own speed (other than the value gotten from the vision processing).

How does the robot brake? is it enough to reverse voltage for a brief period of time, or should the brake halt function of the motor driver circuit be used? And when braking this way, how do we know when to brake? and most importantly how *much* will it brake?

One could imagine a feedback loop between the speed that is reported by the vision processing to use as an estimate for speed. This however can create an unstable moving force if not careful enough when programming. It would be kind of irritating if the robot would continuously accelerate then brake.

When turning it is also very difficult to estimate the turning radius because the outer wheel will always move with the previous speed (when heading straight ahead) and the inner wheel will move slower. But as described above, how can the speed of the inner wheel be estimated. The turning itself is a feedback loop involving the vision processing so the final destination angle will always be correct, but not the rate of turning or turning radius.

5.2 Interface

The visible information consists of a green field, 320 by 240 cm big with six moving robots. Five of the robots are white and are the sheep, the final sixth robot is black or grey and is user controllable. The sheep will move independently, and they will try to move together and also to run away from the wolf.

The mouse can be used to rotate the field in order to view it from any desired angle and the tilde key (~) can be used to toggle fullscreen mode. To switch between wolf and dog, the enter key is used and to steer the wolf/dog, the up, left and right direction keys are used. To quit the simulator a simple press of the ESC key will suffice.

5.3 Structure

OpenGL is used as an overlaying process. And since the GLUT library is used this will make the overlaying structure GLUT like. In the last revision of the program, the simulator and the actual logic is very easy to separate and the structure of the simulator has improved greatly. This means that it is relatively easy to add functionality or to modify existing functionality.

5.3.1 Sheep

The robot information is stored in a struct defined in sheep.h. There is also some basic geometry settings.

Initialization this function randomizes all starting points for all the sheep and the wolf. It also sets some other starting values.

moveRobot The speed input to this function is of the motor speed format defined in sheep.h. The two different motor speeds are split up and multiplied with dtime in order to get the movement of each wheel since last time. This movement is then passed onto moveMotor.

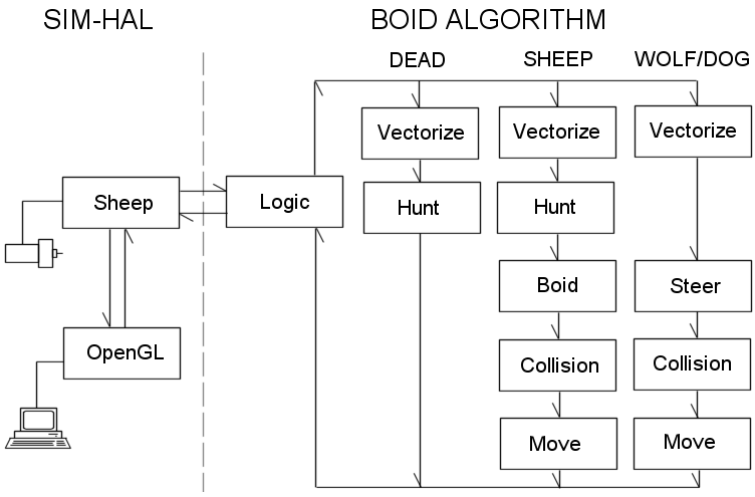


Figure 5.1: The block description of the simulator conencted to the boid algorithm part.

moveMotor First this function checks if the robot should traverse straight ahead. If this is the case, both x and y coordinates are updated simply by using sin and cos. The angle of the robot should of course not change. If it is not traversing straight ahead, then the angle and radius shown below needs to be calculated.

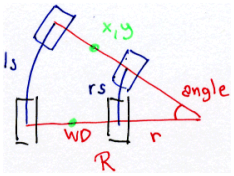


Figure 5.2: Figure of the movement of the wheels.

$$\frac{ls}{r} = \frac{rs}{R} \tag{5.1}$$

$$R = r + WD \tag{5.2}$$

$$r = \frac{ls(r+WD)}{rs} \tag{5.3}$$

$$r = \frac{WD \times ls}{rs - ls} \tag{5.4}$$

$$angle = \frac{O}{d} = \frac{ls + (rs - ls)/2}{r + WD/2} \tag{5.5}$$

With the angle and r values the future position in x and y (relative to the origin of r) can be calculated. Also the current position is calculated (also relative to origin of r). Then theese two are subtracted to get the relative movement of the robot. Last the x and y positions as well as the direction values are updated.

5.3.2 OpenGL

This block handles the drawing of everyting. It uses the GLUT library to display and handle keypresses and mouse.

DrawRobot, DrawField and DrawText These functions simply draw the things you see on the screen and the the function display causes the actual update.

Keyboard and Mouse These two functions just read from the mouse and keyboard to control different things in the program.

FullScreen if the tilde key (~) is pressed during simulation the fullscreen mode is engaged, it can behave somewhat buggy on some platforms.

Chapter 6

Boid Programming

One of the main goals with this project was to get a nice movement of the sheep herd, a collaboration of the movements but without a big information flow between the sheeps. Out of these requests the choice fell upon an algorithm that provided simple behaviors of individual sheeps and also had an organized group behavior [9], it is called the boid algorithm. This chapter does not only describe the boid algorithm, but it also includes a description of the collision avoidance function.

6.1 Flocking algorithm

According to Craig Reynolds[9] it is only needed to implement three basic rules to get a flocking behaviour. Those three rules are:

separation steering to avoid crowding local flock mates.

alignment steering toward the average heading of local flock mates.

cohesion steering to move toward the average position of local flock mates.

To fit this project these rules has been expanded and changed a bit.

6.2 The Main Logic

Here the actual algorithms are placed, most of them are based on the vector function class (see section 6.3.1). This is executed for each robot and the input to this layer is basically all other robots position and direction. With this information, the goal is to return the desired speeds of the two motors.

6.2.1 Vectorize

This function takes all the robots positions and directions and converts them into vectors describing the position relative to one self. After this function there is no longer any information that could be viewed as cheating information (about the sensors). Since after this point all the positions are relative to one self and no absolute values at all.

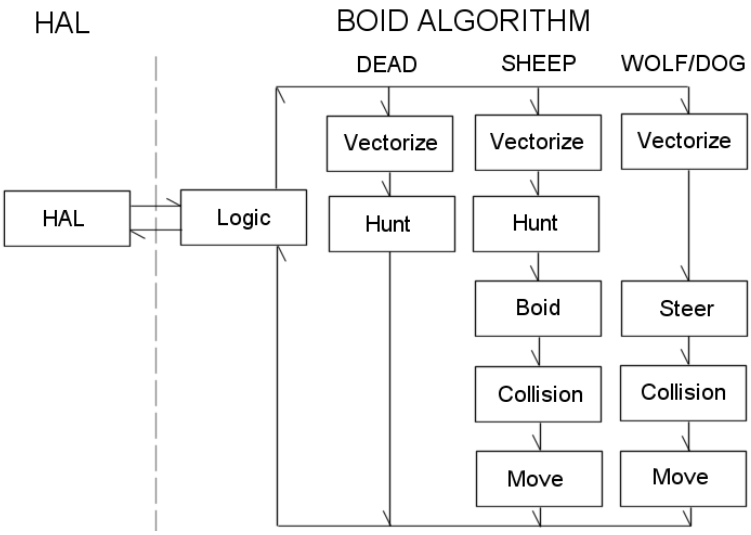


Figure 6.1: A block diagram over the HAL and boid code.

6.2.2 Hunt

This function simply tells whether the sheep should die or be reborn. If the wolf is close by and the sheep does not see any other sheep, it gives up and dies. If on the other hand the dog is barking close by it gets reborn again.

6.2.3 Boid

A normal sheep is of course only interested in what the neighboring sheeps are doing and it has a maximum range of sight. The algorithm used takes this as an advantage and uses only the sheeps in the closest neighboring area around it. The rules used are:

Moving towards the center of the herd. this rule takes the average position to the local flock mates, but not the ones real close because of the possibility to collide. This is not a important rule and therefor its influence on the final output is small.

Moving in the same speed and direction as the herd. The reason of using this rule, even if its importance is not big, is to get a smooth group behavior. This rule influences the final output very little.

Avoiding collisions with the other sheeps. The reason of this rule is obvious. It is an important rule so the influence on the final output is big, but it is only in use when it is close to any other sheep.

Avoiding collisions with the walls. This is the rule with the highest importance, but only when close to a wall, because of the need to avoid collisions and that it can never be close to more than two walls at a time.

Flee when they see the wolf. The importance of this rule is not as big as them preventing collisions but it is still important and it has a big influence as long as the wolf is in sight.

Keeping a safe distance to the dog. This rule is less important than the wolf rule and its importance changes slowly according to the distance to the dog. But it is in use as long as a sheep can see the dog.

All these rules are expressed as linear functions (see fig. 6.2). One of the axes is

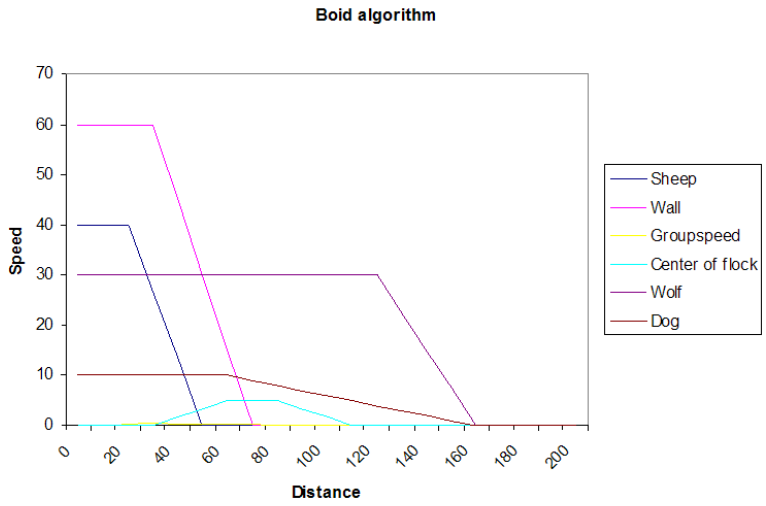


Figure 6.2: Linear function of the boid rules.

representing speed and the other distance, the angle to/from the object is kept as it is. After calculated all the rules, the velocity vectors are added together to update the current velocity and angle see table 6.1).

```
flocking_algorithm()
{
  v1 = rule1();
  v2 = rule2();
  v3 = rule3();
  return v1+v2+v3;
}
```

Table 6.1: Psuedocode of the boid algorithm.

6.2.4 Steer

This moves the wolf or dog depending on the user input. Pressing the up button accelerates the robot and a press of the sideway buttons, rotates the robot.

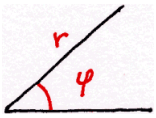
6.2.5 Collision

The result out from the boid algorithm is not 100 percent collision secure, so to prevent collisions a special function was implemented. According to the distance and angle

Algorithm 1 The boid algorithm in mathematical terms.

Variables:

- $\overrightarrow{d} = \{r, \varphi\}$
- $|d| = r$
- $\angle d = \varphi$



$\overrightarrow{s_i} = \{f_i(d), \angle d\}$

1. $f_i(x) = \begin{cases} l_1, & x \leq m_1 \\ \frac{h-l_1}{n_1-m_1}x + l_1 - \frac{h-l_1}{n_1-m_1}m_1, & m_1 < x \leq n_1 \\ h, & n_1 < x \leq n_2 \\ \frac{h-l_2}{m_2-n_2}x + h + \frac{h-l_2}{m_2-n_2}n_{2,} & n_2 < x \leq m_2 \\ l_2, & m_2 < x \end{cases}$

i	l_1	h	l_2	m_1	n_1	n_2	m_2
1	0	5	0	60	70	90	110
2	0	0,5	0	10	30	30	100
3	40	40	0	10	10	10	40
4	60	60	0	10	10	10	50
5	30	30	0	120	120	120	160
6	10	10	0	60	60	60	160

2. $\overrightarrow{s_R} = \sum_{i=1}^{6} \overrightarrow{s_i}$

to an object, an area was set as forbidden (see fig. 6.3). The closer the object was to the robot, the larger the forbidden area become. First a small part is set and a short length is added to the distance, if the distance is still to short this procedure will be repeated. This is repeated three times in figure 6.3. If the desired direction is towards this forbidden area then it is rotated so that it is not. In other words, the closer the object is, the sharper it has to turn, if it tries to move towards it.

6.2.6 Move

The input to this function is the current direction and the desired speed and direction. The purpose of this function is to convert the desired movement to two values representing the speed of the motors. This value is then concatenated and returned.

First there is a check if angle is out of bounds. Then the overall speed is set. With this speed, the two speed values are updated depending on the angle. If the angle is greater than one radian (or less than minus one) the turning speed is set to a fixed value. Otherwise it is depending on the angle.

As a last procedure the two values are concatenated. The values first bit represent the direction and the last seven bits represent the speed in this direction, where 0x7F is

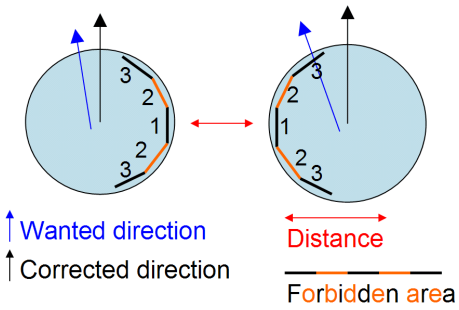


Figure 6.3: Collision avoidance with two sheeps. One of them had to change direction. The forbidden area is number according to the distance.

full speed.

6.3 Other Functions

In this programming there is also some code that is used here and there and not really belongs anywhere.

6.3.1 Vector

This is a lot of functions that operate on the vector struct described in vector.h. The purpose of this was to reduce the complexity of the boid programming. These functions is used throughout the boid algorithm code. There are functions that do adding, subtraction, division, multiplication, normalization, length, direction and even add a list together.

Chapter 7

AVR Programming

This chapter describes the programming needed to use the boid algorithm and other functionalities on the AVR platform. The code that depends on the hardware is of course never movable so that part of the code used in the simulator could not be used again, it had to be rewritten.

7.1 Platform dependent code

A initialization functions had to exist to set up all the hardware. And the ordinary functions, earlier simulated, had to be written. As shown in fig. 7.1 there are three different parts in the AVR HAL. One communication, for receiving the environment data, one for setting and controlling the motors and the last is the main which also calls the boid algorithm.

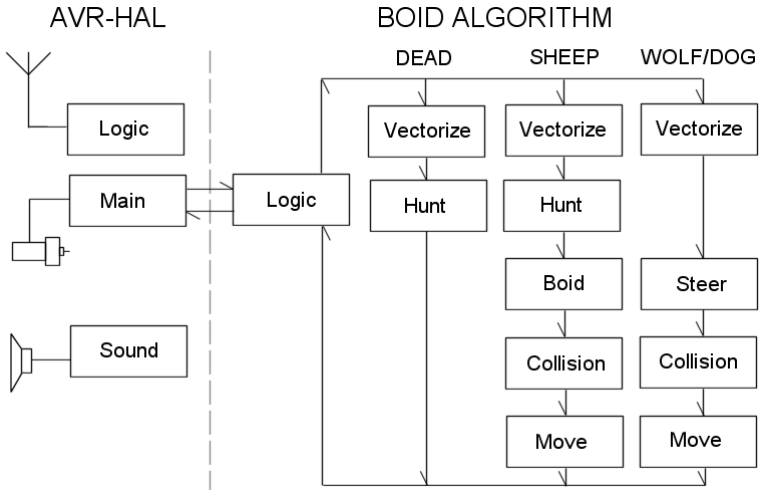


Figure 7.1: Block structure over the AVR HAL and boid algorithm programs.

7.1.1 The main function

In the main function only two things are done. Initialization of all the hardware and runs a loop that handles the *execLogic* function and sets the speed of the two motors. These functions are only executed after the positions have been updated. The *execLogic* calls the boid algorithm and all around it.

```
main()
{
  init();  //initialize every thing on the micro
           //controller.
  loop()
  {
    if(complete)  //if the positions has been updated
                  //via the receiver function.
    {
      speed = execLogic(robotPos);
      moveRobot(speed);
    }
  }
}
```

Table 7.1: Pseudo code of the main function.

7.1.2 Communication

As mentioned earlier (see section 2.3) the data is sent to the sheep via an RF-transmitter that is connected to the serial port of the computer and received by the sheep using the receiver connected to one of the USARTs. To get a robustness of this part of the system the choices made was to use a baud rate of 9600 bits/second, sending start bit, 8 bits of data (one byte), one parity bit and one stop-bit per transmission. These settings fulfils more than enough the needs of data transmitted to the robots (see algorithm 2).

Algorithm 2 Data needed to transmit.

$15 * 11 * (6 * 6 + 1) = 6150$

- The off-board computer will deliver data 15 times per second.
- For every sent byte it will take 11 bits to send it.
- It is 6 robots.
- For every robot it has to send 6 byte.
- One byte is sent for the wolf/dog control.

All functions controlling the robots position and driving can be traced back to the interrupt function (*SI_UART0_RECV*) called when a byte is received. This means that the robots position and all the settings to the motors can be calculated as often as 15

times per second, the speed used is depending of how often the off-board computer can transmit new data.

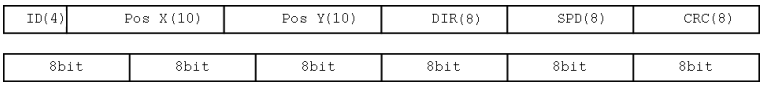


Figure 7.2: A package and the size of its different parts.

A byte received is temporary stored, using the *changePos* function, until a whole package is received. A package contains a sheep identity number, position (in X and Y), direction and speed and a CRC byte (see fig. 7.2) of that sheep. When the last byte is received the position is updated and some error controls are made. Both are made in the changePos function, and a new speed and direction is calculated. The speed and direction is then used to set the corresponding speed and direction to the two motors via the loop in the *main* function.

```
SIG_UART_RECV()
{
  while(!wholePackageReceved)
  {
    receiveByte;
    complete = changePos();  //Add the byte to the
                             //received package.
  }
}
```

Table 7.2: Pseudo code of the receiver function.

7.1.3 Driving

The motor control circuit is taking a pulse width modulated signal as input for controlling the speed of a motor. These signals are generated from two of the PWM modules, using the same timer, on the micro controller. They are set to output a phase and frequency correct signal at 4kHz. The changes, in size of the signal, is generated using an interrupt on rising edge and can be set to 250 different values. All of these parameters are set to constant values in the *PWM_Init* function, except the PWM values that are changed in the *moveRobot* function.

It is in the *changePos* function the received bytes is first controlled for any parity errors and then, if no error found, it is temporary stored until the entire package is received. When the whole package is received, CRC control and an update of the robots position is made. After the last robots position is updated the function returns *true*, so a new speed can be calculated. If an error is detected in any of the received bytes the function returns *false* and then the positions is not updated that time.

To set the speeds and directions a function called *moveRobot* is used. As input it takes the speed and direction for each motor and sets the corresponding values to the PWM modules and to the driver circuits.

```
changePos()
{
  addByteToPackage();
  while(receivedBytes < packageSize)
  {
    if(checkCRC() == true)
      updateRobotPos();
  }
  if(updatedRobots == numRobots)
    return = true;
  else
    return = false;
}
```

Table 7.3: Pseudo code of the changePos function.

```
MoveRobot(speed)
{
  phaseLeftMotor  = speed:7;
  phaseRightMotor = speed:15;
  speedLeftMotor  = speed | 0x007F;
  speedRightMotor = (speed | 0x7F00) >> 7;
}
```

Table 7.4: Pseudo code of the moveRobot function.

7.2 Audio

Since we decided to use an external DAC there was a need to write a software part to send the audio data to the DAC. There are several commands to make the DAC convert the digital audio data into an analog signal. This should be done as simple as possible, therefore always the same command is sent to the DAC and after this the chip select signal is raised which updates the DAC output register. Always a 16-Bit data packet is sent to the DAC. The first Byte is the command for the DAC and the second Byte is the audio data which should be converted into an analog signal.

```
Command: 0000 1001 **** ****
load DAC input register with second Byte and update DAC register
```

To generate the correct sampling frequency of about 8000Hz an internal oscillator of the Atmel microprocessor is used. The generated frequency is about 7800Hz which is accurate enough to play the sounds. The audio data is stored in the flash memory of the processor board. Every byte of the audio data is loaded into the main memory when the data should be sent to the DAC.

The low sampling frequency and the low resolution already reduces the size of one sample a lot. But since the flash memory of the Atmel processor board is limited in the size (128Kb Flash) there was a need to reduce the size of the audio data further. We decided to use a lossless audio compression. We choose lossless audio compression since it is computational cheap to decode the data. There is no need for frequency analysis of the data for example. Another reason was that we did not want to reduce the quality of the audio sample further.

7.3 Lossless Audio Compression

A special audio compression algorithm was used since general compression algorithms like PkZip do not perform well with audio data since they do not take into account the special structure of the data. The approach we are using is used in many lossless audio compression algorithms like Monkey, Shorten[10], AudioPaK[3] or LPAC. The algorithm which is used first removes the redundancy from the signal and then codes the resulting signal with an entropy coding scheme. The three stages (see fig. 7.3) of the audio compression are blocking, predictive modeling and entropy coding.

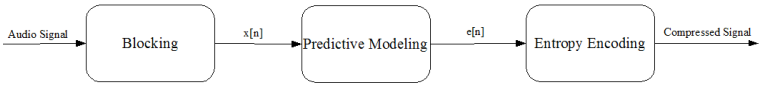


Figure 7.3: Block Diagram of the Audio Compression

7.3.1 Blocking

A predictive model of the audio signal is built. To predict the value of an audio sample $x[n]$ the previous samples $x[n-1]$, $x[n-2]$ and $x[n-3]$ are used. Four different fixed polynomial predictors are used which are described below. A digital audio signal is not stationary and therefore the predictor which is working the best is different from

one section of the audio signal to another. But there exists a time-span over which one predictor is working well for a section of the audio signal. Therefore the audio signal is divided into several blocks. For every block the predictor which is working the best is used. A fixed block length is used to simplify the decoding. The information about the used predictor is coded into a small header at the beginning of every block. Since we have four different predictors 2 bits are needed for the header. At the moment no other information in encoded into the header.

7.3.2 Predictive Modeling

The second stage of the audio compression is the predictive modeling of the audio signal. An established model for audio signals is linear predictive coding (LPC). The predicted sample ($\hat{x}[n]$) is a linear combination of past samples. (a_i: coefficients for the predictor)

$$\hat{x}[n] = \sum_{i=1}^{p} a_i x[n-i]$$

The encoded signal is the difference between the estimate of the linear predictor and the audio signal. We are using a simple FIR prediction method which uses only integer coefficients. This method was first introduced by the lossless audio compression algorithm, Shorten.

Four simple FIR predictors are used. The prediction coefficients (see fig. 7.4) are those specified by fitting a p order polynomial to the last p data points.

$$\begin{aligned}
\hat{x}_0 &= 128 \\
\hat{x}_1 &= x[n-1] \\
\hat{x}_2 &= 2x[n-1] - x[n-2] \\
\hat{x}_3 &= 3x[n-1] - 3x[n-2] + x[n-3]
\end{aligned}$$

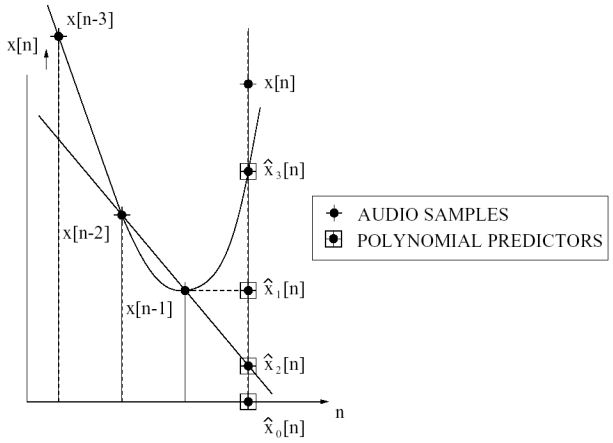


Figure 7.4: Polynomial Approximation of the Predictors (This diagram is taken from [3])

The resulting residual signals which are encoded can be retrieved from very simple recursive equations.

$$
\begin{aligned}
e_0[n] = &\quad x[n] - 128 \\
e_1[n] = &\quad e_0[n] - e_0[n-1] \\
e_2[n] = &\quad e_1[n] - e_1[n-1] \\
e_3[n] = &\quad e_2[n] - e_2[n-1]
\end{aligned}
$$

The decision which predictor is used for one block is simple since the sum of the absolute error values is linearly related to which predictor is working the best for one block. The predictor with the smallest sum of absolute values of all residuals is used for that block. As described above the information which predictor is used is stored in the header at the beginning of every block. The residuals which are going to be encoded are in the optimal case small positive or negative integers.

7.3.3 Entropy Encoding

The last step of the lossless audio compression algorithm is the entropy encoding. Rice-Coding is used as encoding algorithm. This algorithm is used in many lossless audio compression algorithms. It is a special form of Golomb coding which is defined to be optimal for exponentially decaying probability distributions of positive integers. The Golomb code divides a positive integer into two parts. The lower bits are stored in a binary representation, and the higher bits are stored in a unary representation. Since the residual which are encoded are positive or negative values a mapping into the positive domain s needed.

$$
M(e_i[n]) = \begin{cases} 2e_i[n], & e_i[n] \geq 0 \\ 2\,|e_i[n]| - 1, & else \end{cases}
$$

The probability distribution of these positive integers is looking similar to an exponential decaying probability distribution (figure 7.5). There are a lot of small values and just a few large values. Rice-Coding seems to be a good way of encoding these values.

As described the values are divided into two parts. The lower m bits are stored in a binary representation. The second part consists of n consecutive ones. n has the same binary representation as the yet unused most significant bits of the value which is encoded. After this zero is inserted as a stop bit. The residual is encoded with a variable length encoding scheme. Since it is expected that there are a lot of small values this is supposed to be a more efficient representation. In the current implementation the value of m, which is the number of bits is stored binary and is also the same for all blocks of the audio signal.

7.3.4 Decoding

The decoding of the audio signal is supposed to be computational cheap and straight forward. First from the header of every block information about the used predictive model is read. The values are decoded and then the block of the audio signal is reconstructed according to the used predictive model. No sophisticated calculations are needed to reconstruct the audio signal. There are only a few multiplications and additions used in the decoding. All calculations can be done with integers.

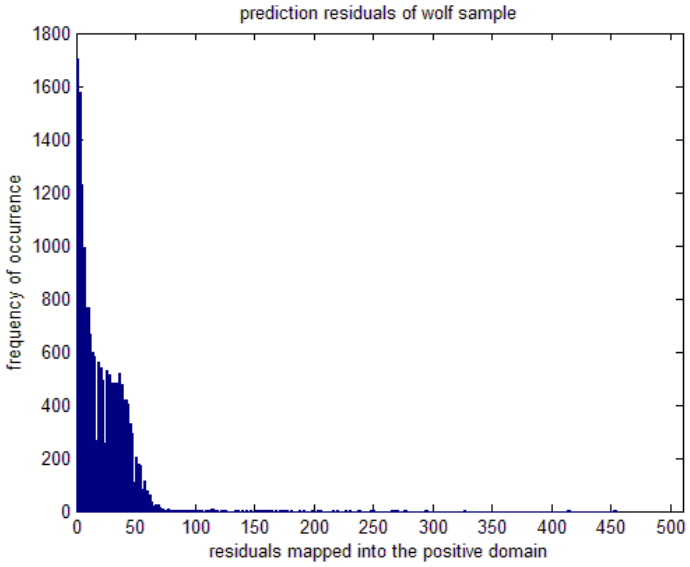


Figure 7.5: Prediction Residuals of Wolf Sample

7.3.5 Discussion

The compression rate which can be achieved with this audio compression algorithm is up to 1.7 (original size/compressed size) with our samples. A better ratio could probably be achieved by choosing a more optimal m ,which is the number of bits which are stored binary, for every block of the audio signal separately. The optimal number of low order bits to be stored binary is related to the variance of the residuals of the block. A simple estimation of the optimal m can therefore be found. This could be encoded into the header of every block.

There are some problems with the decoding of the audio signal in the Atmel microprocessor. At the moment real time decoding is not possible. The required memory and bit shifting operations are not fast enough in the current implementation. The audio decompression is also not the most important work for the processor. It should be possible to do the audio decompression in the background while the processor is working on more important tasks. There is more work required to accelerate the audio compression. This should be possible with more efficient memory access strategies. Therefore in the current implementation just one predictive model for the whole audio signal is used.

$$\hat{x}_1 = x[n-1]$$

Also the same m for the whole audio signal is used. With just one predictive model the ratio of the compression algorithm is smaller and is working well just for some audio signals. For our samples an acceptable compression ratio is achieved for the longer audio signals (1.4). As already said it should be possible to accelerate the decoding and make use of the four predictive models.

Chapter 8

Vision Processing

In a real-time mobile robot application where interaction with humans or a dynamic world is required, vision system employing region segmentation by color is crucial. Fast color image segmentation can be accomplished by careful attention to algorithm efficiency using a commodity image capture and CPU hardware. This section describes a system capable of tracking the color regions in the robot environment to provide required information for robot localisation and navigation.

8.1 Criteria for Vision System

The robot in the environment need some source which can provide the information to know its own location and in which direction it is heading. Without such a system the robots are simply blind. To make the robots aware of their environment they should contain some sensors on board or off board. We are using an off board vision system for this purpose.

8.2 Vision vs Other sensor Systems

We thought about having IR sensors or sound based sensors on each of the sheep. Besides having more electronic, these sensors these also increase the complexity of software part. Each robot should have sensors around it that also provide redundant information about the environment which should be handled properly to make efficent use of it. This in turn would make the control logic and the software in the robot more complex. So we made trade off between vision and other sensor systems and decided to have one camera from above which can capture the whole robot working area. Instead of running a logic in every robot to use on board sensory data we use one camera and a processor along with transciever to process and send the location and direction of each robot to all robots.

8.3 Selection of Image Formats

Our approach involves the use of thresholds in a three dimensional color space. RGB and YUV are two color spaces which are widely used. The choice of color space de-

pends on digitizing hardware which provide the images and the utility for the particular application.

When a color is to be specified in RGB it is done in terms of a ratio of the intensities of Red, Green and Blue in the pixel. But the volume implied by such a relation is canonical and cannot be represented with simple thresholds. Adding to that, the classification software should be robust in the face of variations of brightness or illumination. In contrast YUV have the advantage that chrominance is coded in two of the dimensions while intensity is coded in the third. This color space is therefore more useful than RGB for color segmentation.

The previous camera (the Pyro Webcam) used provides a YUV raw image so there is no need for color tranformation. The YUV format from the camera is of the type UYVY (YUV422) or UYYVYY (YUV411) at 640x480 resolution. The xv-library that is used to present the image needs to have the data in YUV 4:2:0 planar format so a conversion was neccessary.

The vision system we designed needs an environment with specified colors like green background, Robots having white colored body, tail in black color and ears of each sheep should have different colors. The surfaces of the robots should also not shine on lighting.

8.4 Image Processing Algorithm

The software system is composed of four main parts; a novel implementation of a threshold classifier, a merging system to form regions through connected components, a seperation and sorting system that gathers various region features, and a top down merging heuristic to approximate perceptual grouping. These parts are explained below.

8.4.1 Thresholding

The mechanism used for thresholding is an important consideration for efficiency, because the thresholding operation must be repeated for each color at each pixel in the image. In our approach, each color class is specified as a set of six threshold values: two for each dimension in the color space. One way is to check the membership of a color class by using a set of comparisions. But it is not efficient because it needs 6 conditional branches to determine membership of one color class for each pixel.

Our implementation uses a boolean valued decomposition of the multidimensional threshold. Such a region can be represented as the product of three functions, one along each of the axes in the space (see fig. 8.1).

The decomposed representation is stored in arrays, with one array element for each value of a color component. Thus a class membership can be computed as a bitwise AND of the elements of each array indicated by the color component values:

```
pixel_in_class = YClass[Y] AND UClass[U] AND VClass[V]
```

The resulting boolean value of pixel_in_class indicates whether the pixel belongs to the class or not. This approach allows the system to be implemented as a few array lookups per pixel. The operation is much faster because the bitwise AND and the table lookup is a singnificantly faster than six conditional branches.

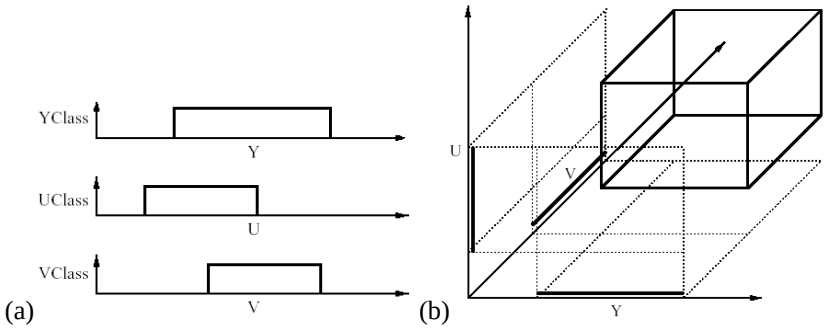


Figure 8.1: Figures describing (a) the binary signal decomposition threshold and (b) the visualization as threshold in full color space.

8.4.2 Connected Regions

After the various color samples have been classified, connected regions are formed by examining the classified samples. This is typically an expensive operation that has a huge impact on real-time performance. The procedure of merging our connected components is implemented in two stages for efficiency reasons.

The first stage is to compute a run length encoding (RLE) version for the classified image. In many robotic vision applications significant changes in adjacent image pixels are relatively infrequent. By grouping similar adjacent pixels as a single "run" we have an opportunity for efficiency because subsequent usage of data can be done by an entire run rather than individual pixels. There is also a practical benefit that region merging now need only to look for vertical connectivity, because the horizontal components are merged in the transformation to the RLE image.

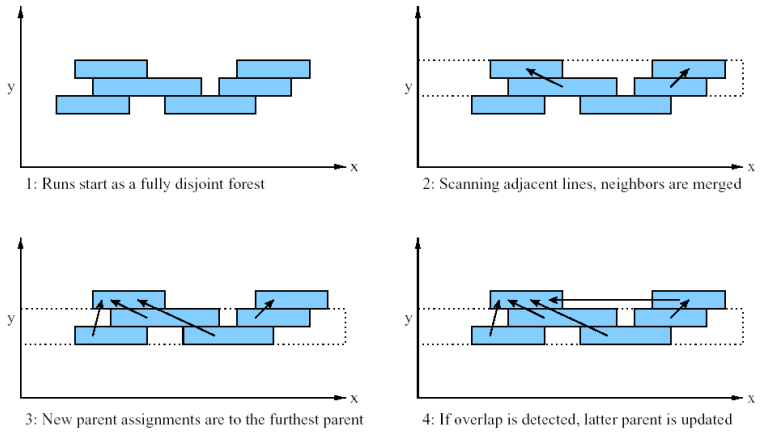


Figure 8.2: Merging regions

The second stage is the merging method which employs a tree-based union find with path compression. The merging is performed in place on the classified RLE image. Each run contains a field with all the necessary information, an identifier indicating a

run's parent element (the upper leftmost member of the region). Initially each run labels itself as its parent, resulting in a completely disjoint forest. The merging procedure scans adjacent rows and merges runs which are of the same color class and overlap under four-connectivity. This results in a disjoint forest where each run's parent pointer points upward toward the region's global parent. Thus a second pass is needed to compress all of the paths so that each run is labeled with its actual parent. Now each set of runs pointing to a single parent uniquely identifies a connected region like in (see fig. 8.2).

8.4.3 Extracting Region Information

In this part we extract region information from the merged RLE map. The centroid and the size of the region are calculated incrementally in a single pass over the forest data structure. We made a look up table where we can see all the regions by just mentioning the label of the region which was made as an index of the region structure in the region table. All the regions in the table are sorted by size so that high level processing algorithms can deal with the larger regions (which are more important) and ignore relatively small ones which are most often the result of noise.

8.5 Implementation and Program Structure

The implametation part involves two parts; Image capturing and processing the image. The xv-library is used in conjunction with the Philips pro web cam drivers to get the raw image in YUV 420P format. The image is then sent to the user defined image processing functions which in the end calculate the position of the robot and heading direction of the robot. In the following sections the structure of the whole project is described.

8.5.1 camusb.c

This file contains the functions called

initCam To get a camera hardware and initialise it to the user defined settings

getImage To get a picture from the camera and put it in a buffer

closeCam This releases the camera

printSettings This prints all the settings of the camera to make sure that we are getting correct parameters

8.5.2 xv.c

This is the source of the xv library. Its functionality is to present a videostream to the monitor as quick as possible. It is using the overlay functions on the graphics card for this.

8.5.3 xvshow.c

This contains functions which are useful to see the images captured by the camera. It is meant as an easy interface to the xv-library.

initDisp Contains some settings needed to start the display

showImage To show the image

8.5.4 datastructures.c

All the user defined as well as predefined data structures are declared in this file. The structures which are defined in the header datastructures.h are

```
typedef struct
{
   int start_x;
   int end_x;
   int rowno;
   UCHAR color_class;
}RUN;

typedef struct
{
   UCHAR Color_Class;
   UCHAR Y[10];
   UCHAR U[10];
   UCHAR V[10];
}Color_Table;

typedef struct
{
   UCHAR y[IMAGE_WIDTH * IMAGE_HEIGHT];
   UCHAR u[(IMAGE_WIDTH >> 1) * (IMAGE_HEIGHT >> 1)];
   UCHAR v[(IMAGE_WIDTH >> 1) * (IMAGE_HEIGHT >> 1)];
}yuvpic;
```

8.5.5 colortable.c

Each dimension of the color space is divided into 10 levels depending on the color intensity values. If the color class is black it is represented like

```
YClass[] = {1,1,1,1,1,1,0,0,0,0}
UClass[] = {0,0,0,1,1,1,0,0,0,0}
VClass[] = {0,0,0,0,1,1,0,0,0,0}
```

A big array was made which contained all the represented color classes one after the other.

8.5.6 allfunctions.c

This contains all the image processing funtions

AssignColorDivision This assigns the level of each color component to which it belongs depending on the color intensity value. There will be 6 conditions to assign an integer between 1 - 10 indicating the level to which the color component of a pixel belongs.

MakeColorTable This constructs the color table with the structure Color_Table defined in datastructures.c using the array defined in colortable.c

Image_Decomposition A two dimensional array called DecompMat[][] is made to contain the color level of each color component of every pixel which is the return value of AssignColorLevel

AssignColorClass Bitwise AND operations is used to find the colorclass of each pixel. For example: if we have to test whether a pixel belongs to black color class or not and the pixel color values in DecompMat[][] are (2, 6, 5). Then we just have to do the operation:

```
YClass[2] AND Uclass[6] AND VClass[5]
```

Which in this case resolves to 1, or true indicating that pixel belongs to black color. If it is 0 we have to check for other color class.

FindRuns This function forms the runs in the given image by looking for the adjacent similar pixels in each of the row.

ParentOf This function looks at runs in the adjacent rows to assign the top left most run of same colored run as the parent.

FindParentRuns This function calls the ParentOf function for every run in the RLE image.

FindCenterOfRegion In this function the center of each colored region is calculated.

8.5.7 FunctionCalls.c

This function calls the required functions in the allfunctions.c.

8.5.8 main.c

In this file, the initCam function is called first to get a frame and the frames are deliverd at a rate of 15 frames per second. Each of the frames is sent to FunctionCalls function which calls all the image processing functions.

8.6 Results

The algorithm implementation was first done for RGB images because we were not sure of the image format and the camera we are going to use. The program worked well for still images that has some colored objects on a white background. It is also able to calculate the center of all the colored regions. We use OpenCV library in windows for

the purpose of grasping the images. But when we came to know that the YUV format is more suitable for our application we had to give up OpenCV as it does not handle YUV images very well, so we need to use the driver software directly which is fairly easy in Linux along with the xv library to view the images that were captured. The program written for YUV worked well for still images, it was also tested with the model sheep keeping it at 1 meter distance from the camera. The program is working for realtime images and it manages color segmentation at a rate of 15 frames per second fairly well, and it was able to calculate the centers of each region.

Chapter 9

Conclusion

This project is not yet finished and that makes it harder to come up with some really good conclusions, since we are not yet looking back as much on the past, but rather are concentrated on the things that need to be done. However, this chapter describes the conclusion in general and also has some discussion topics that was thought of in the project. Other results or conclusions can be found i each chapter as well.

9.1 Software

The software written for the robot has only been tested quite briefly on the electrical platform, since the robot has not yet been built. However does all the different elements work together and the actual boid algorithm was very easy to port to the avr-processor environment.

9.1.1 boid

The boid algorithm works with simple individual rules and produces a nice group behavior. But it takes a lot of tuning to get all the implemented rules so the sheep behave in a satisfactory way.

The trimming of the extra, and more complex, collision avoidance is probably necessary, because of the safety of the robot and their environment.

9.1.2 Speed vs. Forces Discussion in Boid

The vectors set by the boid algorithm was first used to represent speed in the vectors direction. This sounded like a good idea to us and we found it to reflect the movement of the robots very well. Later it was suggested to change this value gotten from the boid algorithms to a force instead. This created several problems. First it was not possible to stand still since you always needed a braking force. This is of course true and very logical and such a force was also added to the simulator as well.

The big problem came when the wolf was quite near a sheep and the sheep tried to run away from it for a longer period of time and then approache a wall (which is often the case). Then the wolf builds up this huge speed of the sheep so when it eventually reaches the wall, the wall will not be able to brake the sheep before a collision has occurred.

Also when implementing this algorithm in forces, one can look at the sheep moving towards the flock. If very far away from the flock the sheep will move slowly, with a speed that always increases until the sheep is at a good distance, then it starts braking and thus comes quite a bit too far. If the sheep vector is treated as a speed instead, the movement will be quite big when far away and as the sheep approaches the speed will get smaller and smaller, and when at the desired distance it will be zero.

9.1.3 Simulator

The simulator worked very well for the purpose it was programmed, the movement and the algorithm problems could easily be seen and one could get a feel of how the boid algorithm behaves. Every time a problem or thing appeared in the simulator you needed to think through if this would be a problem in reality or not. Though, even if this program would have been using more advanced physics, this would be needed and maybe even be harder than now.

9.1.4 Code Porting

In the beginning a quite well defined structure of the code was designed, over the time of the project it was changed a little, but is mainly still the same. With this well thought of structure it was very simple to move parts of the code, namely the boid algorithm, from the simulator environment to the the actual hardware platform. The only really problem encountered was the lack of a linker flag that made the linking complain about program size.

9.2 Hardware

When this project was begun the main fear was if we would get the hardware components on time. Now, when the project should be finished it showed that this also was the case. The gearboxes that was ordered 8 weeks ago has still not come because of payment issues. And the platform was heavily delayed but is now finished.

9.2.1 External vision

The choice to move to an off board computer has yet only proved good. First there is no additional micro processor platform that the user should interact to, but a computer. The keypad or game pad was also greatly simplified since a commercial can be used. Also does the information sent to the robots not require lots of heavy calculations but can be used directly and thus not take as much processor time.

9.2.2 Problems, Limitations and Discussion about Vision

The main problem in the project is that the vision system has to identify each of the robots and differentiate between different robots. If the problem is limited only to knowing the direction of movement there are techniques like Optical flow to detect the movement of an object. But we need to know which robot has moved. We might use some electronic system to allow each of the robot register its own identity whenever it is moved. But then dead reckoning is a must to find the location of the robot. To solve

this complexity we go for the choice of having color identities on each of the robot. So color segmentation seems to be the reasonable technique to solve the problem.

We had problems with having shadows because those shadows are classified as black pixels. This type of classification confuses the system. But it is hard to know whether the output from the program is correctly calculated or not until we have the actual robot moving environment. So in the mean time we concentrated more to get accurate enough segmentation and correctly calculated results.

We are trying have density based region merging to eliminate some of the errors generated in the region generation. For example if a region were to have a single line of misidentified pixels transecting it, the lower levels of the vision system would identify it as two separate regions rather than a single one. Thus a small error leads to vastly differing results. One solution to this is to employ a sort of grouping heuristic, where similar objects near to each other are considered a single object rather than distinct ones. Since the region has its own area and the bounding box, a density measure can be obtained. We set one threshold density value for each colored region. If two similar colored regions are separated by a small area and if the density of the region formed by connecting those two regions is higher than the threshold then the two regions can be treated as a single region.

It is possible to evaluate the vision system performance only when it is tested in the colored robot environment.

9.2.3 Electronics

The electronics finally made it on a PCB with the size of 8 by 10 cm, which was the limitation of the electronic CAD software. This PCB was made dual layer in order to get a ground plane. To make a PCB by hand that is dual layer is not an easy task but this seems to have worked fine. The mounting of the components has begun and it looks promising.

Chapter 10

Further Work

Since the project is not yet finished there exists a lot of things that need to be done. This chapter will present them and also some additional improvements or 'wannabees'.

10.1 Audio Further Work

There is more work required to accelerate the decompression of the audio signals in the Atmel microporcessor. As described in the dicussion (see section 7.3.5), a more efficient memory access strategy has to be found to be able to use the full audio decompression algorithm. The current implementation achieve acceptable compression ratios only for some audio signals.

10.2 Vision Further Work

Our future work is to have the robot environment working and test the vision system. We need to do mathematical calculations so that we can map the coordinates we got from the algorithm to the actual physical environment. We are planning have double thresholding in situations where the system is experiancing dilemma with nearly similar colors which mostly occurs at illumination changes.

10.3 Hardware Further Work

Since the actual hardware is not built yet. There is quite a substatial amount of work thet need to be done here. Basically the motor-wheel-configuration (see section 3.2.5) must be mounted with the motor and bearing holders, the batteries must be mounted together and the PCB which is under construction must be mounted on the platform. When this is done and all electrical connections are made the robot could be ready for testing.

10.3.1 Robot Hardware Evaluation

When the robot is built the first thing to do is to test speed, acceleration and trim the values in the software, so that a speed which is withing expectations is achieved. The

turning and braking (see section 5.1.2) must also be checked as well as the current consumption.

10.3.2 Receiver Evaluation

One quite crucial part is the receiver, if it will pick up a lot of interference from the motor drivers or not, during the testing this was not the case but the conditions have changed and the RF tranciever is of course sensitive to interference. Though the grounded aluminium platform and the grounded layer on the PCB should remove lot of interference.

10.3.3 Speaker Evaluation

The speaker needs to be tested because there was problems with noise in the first tests outside of the PCB. It would be good to know if this has been solved by a ground plane on the PCB or not.

10.3.4 Design

The design is not yet finished, there are many design suggestions and the only thing needed is the time to make them.

10.3.5 External Computer

The external computer needs to be set up. A transmitter need to be connected to this computer to broadcast the information from the vision processing to the robots and to send the steering commands to the wolf/dog. Also the camera and the gamepad need to be connected to this computer.

10.4 Additional Improvements

There is of coure a lot that can be improved so here is a few of those things collected.

10.4.1 Larger field and more robots

Add more robots and a bigger working space, so it will be more of the flocking behaviour and it will be possible to divide the flock into two parts with the wolf.

10.4.2 Evolve the wolfs and dogs

Implement a hunting and herding algorithm for the wolf/dog, maybe even a possibility to add more wolfs/dogs and let them hunt together. And maybe even to let the dog defence the sheep herd against the wolfs. In other words to make the wolfs and dogs autonomous

10.4.3 Training Algorithms

Generate the values to the boid functions using genetic algorithms. And if the idea above is implemented even the wolf and dog behaviours can generated using genetic algorithm.

10.4.4 Improve simulator

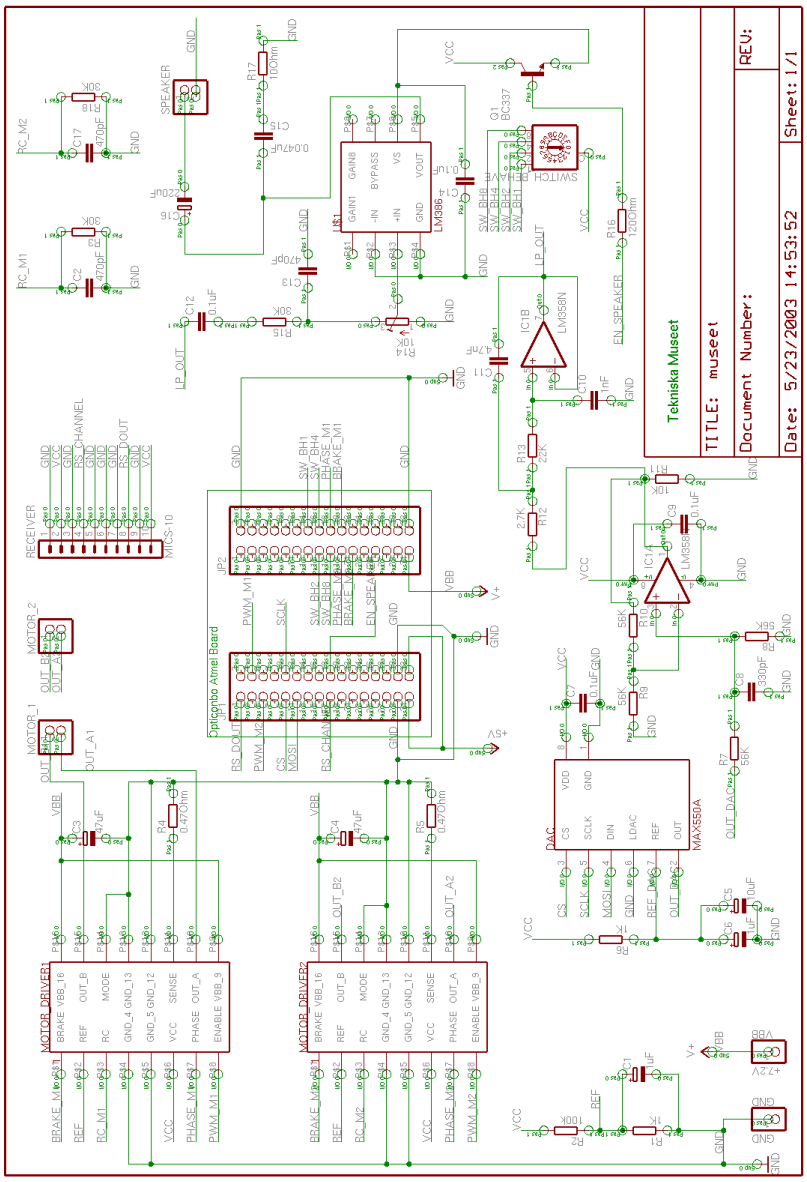
Implement the Newtons law in the simulator, with measured accelerations, frictions and weights.

10.5 Weird ideas

Remove the camera, the off-board, the transmitter, and the receivers and replace them with sensors on board the robots.

Appendix A

Electrical Schematics



Bibliography

[1] Allegro. *3953 Datasheet*.

[2] Atmel. *ATmega 128 Datasheet*, 2003.

[3] Ronald W. Schafer Mat Hans. Lossless compression of digital audio, 1999.

[4] Maxim. *MAX550A Datasheet*, 1997.

[5] National. *LM386 Datasheet*, 2000.

[6] Nodic. *nRF 401 Datasheet*, 2002.

[7] Portescap. *Full Line Motor Catalog*, 2001.

[8] Portescap. *Speedy Line Catalog*, 2001.

[9] Craig W. Reynolds. Boids, background and update. Homepage, 2003. www.red3d.com/cwr/boids/.

[10] A. Robinson. Shorten: Simple lossless and near-lossless waveform compression, 1994.

[11] ST. *LM358 Datasheet*, 2002.

[12] Sebastian Thrun, Maren Bennewitz, Wolfram Burgard, Armin B. Cremers, Frank Dellaert, Dieter Fox, Dirk Hahnel, Charles R. Rosenberg, Nicholas Roy, Jamieson Schulte, and Dirk Schulz. MINERVA: A tour-guide robot that learns. In *KI - Kunstliche Intelligenz*, pages 14–26, 1999.