

Looking like humans do

An approach to robust robot vision



Jan Peter Jansen

Student nr: 0550873

JP_Jansen@hotmail.com

University of Rotterdam

Research facilitator: TU Delft

Foreword

My acknowledgements to my mentors ir. M.M.M. Abdelghani and dr.drs. L.J.M. Rothkrantz.

Summary

This paper describes the research that is done on current image processing techniques in order to improve the level of robustness. The final goal of robot vision is of course to reach the quality of human vision but this is, for now, a thing of the future. The platform on which this research is done is Sony's AIBO ERS-7.



Figure 0.1. Research platform AIBO ERS-7

The platform features relevant to this research are:

- 500 Mhz. RISC Processor
- 640 x 480 camera

A lot of nowadays robot vision systems use colour sequences to determine object classes because this method is fast and reasonable reliable. There are, however, some drawbacks to this system.

First of all, the colours need to be calibrated by a human, which is making the robot less autonomous and the surroundings in which the robot can see limited. The method of colour calibrating is also a temporary one, since lighting conditions change as the day changes and colour definitions are no longer what they should be.

To conclude, robots with colour sequence system can only be placed in highly artificial surroundings and could never cope with changes in any of the pre-defined surrounding boundaries.

The first step towards a more independent way of watching is to replace the current classification method by something more robust and use colours only to fine-tune object classes. . The most robust classifiers available are shapes, because they do not change in different surroundings, so this is the classifier chosen as the new solution.

To be able to classify objects by their shape there are some pre-processing steps to be done. These steps are:

- image compression
- result separation
- similarity grouping (clustering)
- result preparation ordering
- classification

For each of these steps, there are several methods, the ones that were researched are:

Image compression:

To decrease the amount of data and to take the first step in detection the boundaries of recognizable objects, the image needs to be compressed.

- edge detection
Filter the image until only the intensity changes that are larger than a certain threshold remain.
- active contour models (snakes)
Lay a line around a certain area of the image and have it adapt its function according to inner and outer energies until the total energy is zero.
$$E_{snake} = E_{internal} + E_{external}$$
- filling
Seed a certain amount of points on random places in the image plane and expand them until they meet an intensity slope of a certain level.

Result separation

This step is only made to make the total process clearer, in the program it is merged with the clustering step. The first assumption was that clustering algorithms could only cope with 2 dimensions (x,y coordinates of result pixels), so the third dimension (gradient magnitude) should be pre-processed. These methods are used to separate objects in the third dimension.

- colour band separation
Separate the compressed image by colour band
- relative colour separation
Skip step one and arrange the image by relative colour definitions
Example (yellow colour range):
Yellow(low) = $R * G / (B + 100)$
Yellow(high) = $R * G / B$
- gradient magnitude range separation
Divide the gradient magnitudes into ranges, filter ranges with a result amount of lower than a certain threshold.

Similarity grouping

Algorithms to group (cluster) result sets by X-dimension properties

- hierarchical clustering
- K-means clustering
- Kohonens self organising maps (SOM)

Result preparation and ordering

Pre-processing the clustered result sets to give classification processing the right input

- location based recognition
use result coordinates as classification inputs, order by outer boundary line
- angle based recognition
order by outer boundary line and use the angle between the consecutive pixels

Classification

Static insensitive shape clustering algorithms

- back propagation network
- radial basis function network
- linear & quadratic separation

Because of performance and goal-fitness the used methods are:

- edge detection
- K- means clustering combined with gradient magnitude range separation
- Angle based algorithm
- Back propagation network

After testing it seemed that the determined approach on the designated platform was performance technical not realizable. Therefore, some solutions for future researches are proposed:

- location based scanning
Do a couple of global scans and use this information to scan only the regions of interest in the next steps
- variable pixel density according to a Gaussian function
Because of the high-resolution camera, the amount of comparisons (which determines the performance) was too high. The easy solution is to compress the image (take the average of x pixels and store it into one – only additions / subtractions). The disadvantage is that the detailed information is also thrown away. When combined with location based scanning, the image grid can be built up dynamically with the high resolution centre on the object of interest centre.

Overview

1.Introduction	8
2.Goals & project boundaries	9
3.Platform	10
3.1.Hardware	10
3.2.Drivers	11
3.3.Interfaces	11
3.4.Used software	11
3.5.Colour space	12
4.Global solution	13
5.First assumptions	14
6.Methods & techniques	15
6.1.Object boundary determination	15
6.1.1.Edge detection	15
6.1.2.Active Contour Models (Snakes)	15
6.1.3.Filling	16
6.2.Object separation	16
6.2.1.Hierarchical	17
6.2.2.K – means clustering	17
6.2.3.Self Organising Maps	18
6.3.Transformations	19
6.4.Object classification	21
6.5.Pre-classification processing	24
7.Considerations	25
7.1.Object boundary separation	25
7.2.Result separation	25
7.3.Clustering	25
7.4.Pre-classification processing	26
7.5.Classification	26
8.Design	27
9.Implementation	28
10.Tests	29
10.1.Debug surrounding	29
10.2.Test runs	30
10.3.Test results (issues)	30
10.3.1.Performance	30
10.3.2.Local variable lighting	30
11.Solutions (reference for future work)	31
11.1.Performance	31

<i>11.2.Variable lighting</i>	31
12. Conclusion	34
References	35
<i>Websites</i>	35
<i>Papers</i>	35
<i>Books</i>	35
Attachments	36
<i>I. Procedure & Time scheme</i>	36
<i>II. Open-R methods</i>	37
Open-R objects	38
Inter-object communication	39
Shared memory	39
<i>III. Soccer software framework (German Team 2003 code)</i>	41
Main modules	41
Cognition	42
Motion	42
The Architecture behind XABSL	43
Vision	43
Debugging	47
<i>IV. Concepts of edge detection</i>	48
Introduction	48
Goals	48
Methods & Techniques	48
Kernels & Methods	52
Conclusion	54

1.Introduction

The scientific world has been working on computer vision for the last decade. Today we can see a lot of the results of these researches in our daily lives. Take, for instance, the security cameras in large concerns that can recognize faces and take the appropriate action if somebody unwanted is entering the shop.

We are moving towards a world in which computers and robots can do jobs that would normally have been done by humans.

To improve research, the Robocup project is providing an environment in which research can be done on a level that speaks to everyone's imagination, robot soccer.

Teams from all over the world are participating in this yearly event, and are trying to become the world champion of their own league.

The subject of this research is vision, applied to the module in the AIBO's (Artificial Intelligence Bot –a robot dog) software package. Vision is a low level process that provides parameters to the higher-level processes such as behaviour and tactics. Just like human soccer players, robots need to know their position relative to the field, the location of the ball and goal and the location of team members and opponents. Only if these parameters are known, higher-level decisions can be made.

Currently, the games are played on a field with very distinct markers (every field marker has it's own set of colours), so localization is fairly easy.

Since we want robot soccer to move towards human soccer, field specifications are adapted every year in order to finally apply to the soccer field as we know it. With these changes, new issues rise: if you don't know the goal colour, how do you know in which goal you have to score or even more relevant, where is the goal? If you don't know the colour of the opponent's shirt on forehand, how do you adapt? If it's dark or rainy, how do objects change and can you still recognize them?

All the above problems can surely not be solved in one year, or even a decade. But the first step towards a more humanly way of seeing the world is to create a more robust, colour independent vision module. This is, in a nutshell, the goal of this research. Is it possible to replace object recognition by colour sequences with a more robust method of recognizing objects and thereby being able to place robots in an environment that is not predefined, such as the world?

2.Goals & project boundaries

As described in the introduction, the main goal of this research is to create a more robust way of analyzing camera images. The meaning of robust in this context is that the robot can cope with more than the predefined boundaries of a football field. If it were to be placed in a similar surrounding with different field colours (for example, a yellow grass mat), it would still have to know its location (relative to the field). Knowing this, we can specify the following goals:

- Objects need to be recognized by something else then their colour
- Changing lighting conditions may not create differences in the results

Because the robot is only equipped with one camera, it cannot use stereo vision (2 camera's, 2 eyes) to determine the distance to, and the size of, any object. The solution to this problem is to map the real size of the object to the size of the object's reflection on the camera. However, this contradicts with the goal of being independent of pre-definitions. Therefore, an additional boundary is set:

- The sizes of all objects that need to be recognized are known to the robot and may not change

3.Platform

Before any scanning can be done, there must be something to scan, for instance, a software matrix or structure in which rows, columns and colour bands are represented. The following diagram displays the data streams from the robot sensors to the software project.

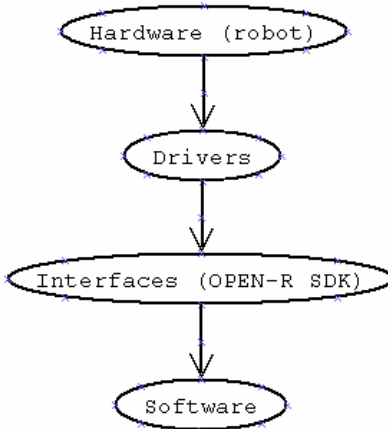


Figure 3.0.1. Data flow from robot to software

Each of these steps contains enough information to fill 4 reports, therefore only a brief description will be given here. The extensive descriptions can be found in attachments II and III.

3.1.Hardware

The hardware specifications are summarized in the following image:

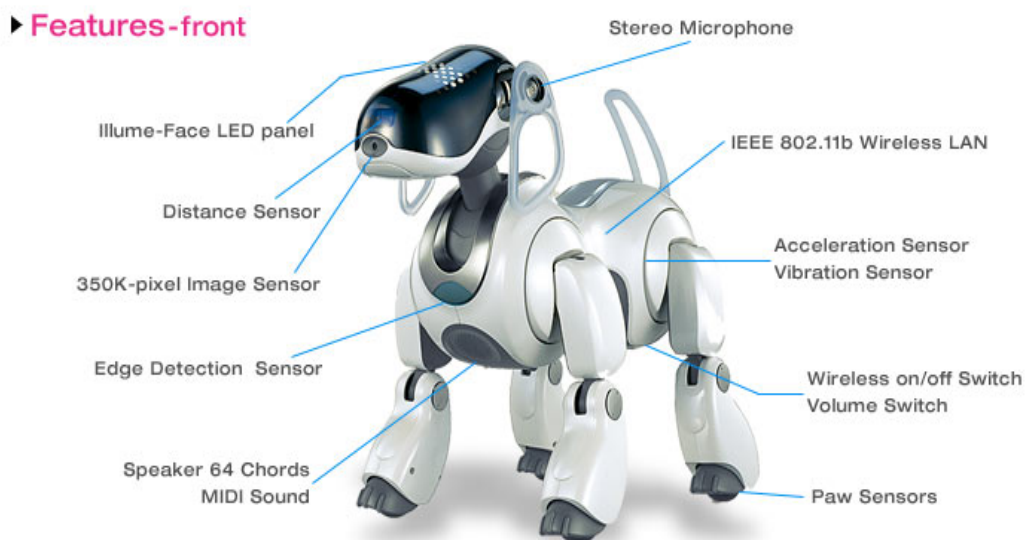


Figure 3.1.1. AIBO ERS-7 features

3.2. Drivers

Driver specifications are excluded from Sony's manuals because of business reasons. Therefore no information on these features are included in this report.

3.3. Interfaces

When one would want to be building software for the ERS-7, the first thing to do is to learn OPEN-R. The OPEN-R SDK provides the interfaces between the drivers (the hardware handles) and the software to be written.

Open-R provides an environment in which multi-threading (concurrent processing) and inter-object or process communication can be done in a fairly simple way.

In Open-R, only two object can communicate with each other. The object that is sending data is called the subject, the receiver is called observer. The following schedule visualizes this system:

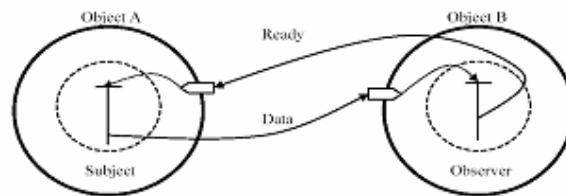


Figure 3.3.1. Inter object communication

When more than two objects are to communicate, a mediator object is to be implemented. This mediator works like a telephone central and is passing messages from object A to object B,C etc.

Objects are bound together by the file connect.cfg. This file ties subject gates to observer gates and defines the type of data that can be sent over the connection.

In runtime, the Open-R interface contains an instance that represents the robot. This instance can be seen as one of the objects described above with predefined gates that can either send sensor data or receive activation commands.

3.4. Used software

The software for which the vision module is written is the Dutch Team 2004 Software project. This project is the ported version of the German Team 2003 code, which was meant for the AIBO ERS 220.

More information on the software can be found in chapter "design" and an extensive description in attachment III.

3.5. Colour space

Sony's OPEN-R SDK provides image structures in YUV colours. YUV was originally developed for backward compatibility with the black and white television.

Originally, TV stations only transmitted the black and white. When the time was ripe for colour TV while most people still only owned a black-and-white set, it became clear that transmitting an RGB signal separately from a black-and-white signal would be highly impractical.

Instead, a system was needed in which a TV station could transmit a signal that could be seen as monochrome on the older but still common black-and-white TV sets, but the same signal should be in colour on the new colour TV sets.

That meant, first of all, that the signal had to continue to have the black-and-white image, and the colour information had to be added to it transparently.

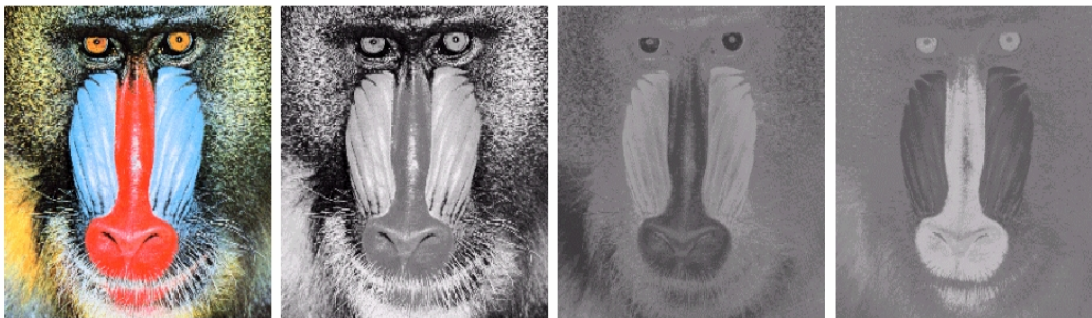
The black-and-white signal is simply the luminosity of the colour and can be calculated by a formula of this type:

$$Y = a * red + b * green + c * blue$$

In that formula, a, b, and c are constants (their value depends, among other things, on the type of phosphors used in the TV monitors of that era).

Because of that, it is possible to transmit any RGB colour as luminosity, plus any two of the three colour channels, then recalculate (with an analog circuit) the third colour channel.

Furthermore, all of the new signal had to fit within a limited bandwidth as apportioned during the black-and-white days. Hence, the transmittal of the difference of the two channels from luminosity.



Original

Y component

U component

V component

Figure 3.5.1. YUV colour components

4.Global solution

The first question that came to mind when the idea of robust (non-colour sequenced) object recognition and localization was brought up, was:

If we cannot use pixel colour sequences, by what features can we recognize a specific object?

When in trouble, always look at the system that comes closest to the system one is trying to build. In this case: people.

How do people recognize objects in their living space? Do we have a database with all the objects we know and for every object the concurrent colour sequences in our mind? Highly unlikely.

However, we do use colours to determine the class of an object, so how does this work? The first thing we do is to separate the object in question from it's surroundings, in other words, we determine it's boundaries. Where does object one end and object two start? To do this, we have to use colours, or at least colour changes, to see where the object's edges are.

Once we've determined the boundaries of an object, we use a combination of shape, size (absolute or relative) and colour to match the object to a previous encounter of it, in order to determine it's class.

It is very important to realise that people can only see details of a very small part of their total image plane. It is impossible to see details of two objects concurrently if they are more than 10 cm apart. This subject will be treated more extensively in the next chapters.

We can summarize object classification into the following steps:

- determine object boundaries
- separate different objects
- use previous encounters of the object to determine it's class

If this process was to be implemented as a software project, the problem is step 2, the separation of different objects. For instance, if there would be a pile of pencils of the same colour, humans could easily separate one pencil from the other, whereas computers would have no way of determining where pencil one ends and pencil two starts. We use separation and classification concurrently while computers need to separate the objects before they can be classified.

5. First assumptions

The first step towards a more robust way of classifying objects is the assumption that it is better to look for intensity changes, instead of checking whether an intensity value exists in a list of calibrated values. This assumption is based on the fact that whenever lighting conditions change, the colour values change accordingly. Thus, the colour tables are provided with false information and need to be recalibrated but the difference between colours remains the same.

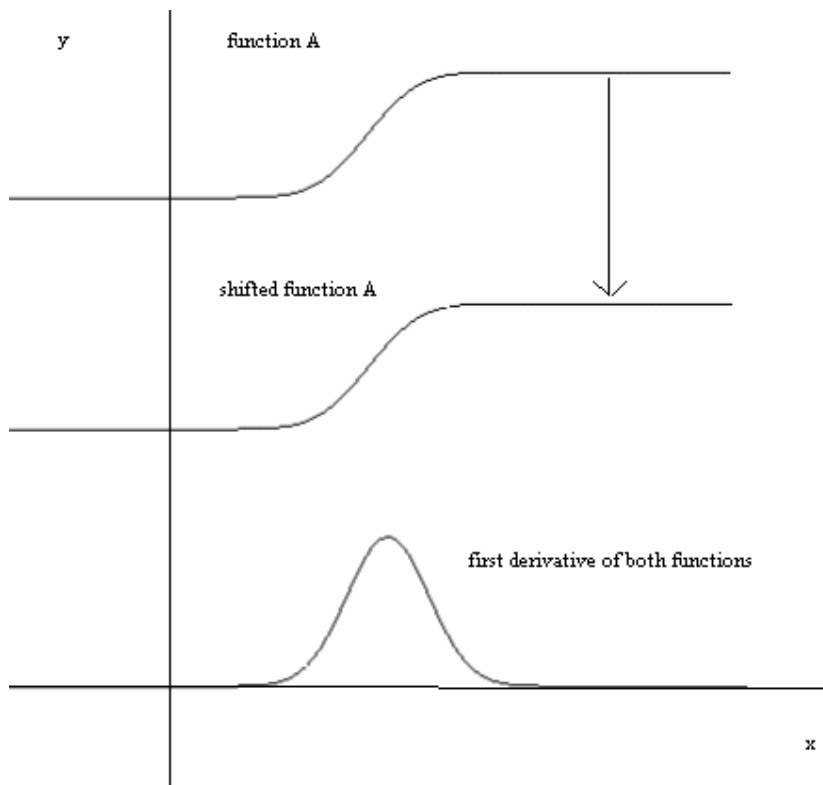


figure 5.1. first derivative remains the same while intensity level drops

In other words, use the first (or second, attachment IV) derivative to find object boundaries. To make the clustering process less expensive and to do some pre-classification steps, the edge detection is done on the separate colour bands. The edges are then thresholded with a function depending on the local intensity (Y-band) to correct for lighting differences.

6.Methods & techniques

This chapter describes the possible methodologies to implement the vision steps described in chapter “global solution”. The techniques are described by step and are:

Object boundary determination

- Edge detection
- Snakes
- Filling

Object separation

- Hierarchical
- Self Organising Maps (SOM)
- K – means

Transformations

Object classification

- Linear & quadratic separation
- Back propagation networks
- Radial basis function networks

Pre-classification processing

6.1.Object boundary determination

6.1.1.Edge detection

The concept of edge detection is to detect the optimal intensity gradient in a two dimensional grid. The optimal gradient is found by taking the second derivative of a function and determine its zero-crossings. Since we do not have a intensity function, we need another way of determining the gradient magnitude (for each point) and the second derivative zero-crossings. Edge-detection uses horizontal and vertical convolution kernels to determine the gradient magnitude and angle for each point. The simplest way of getting the right edges (enough change in intensity) is to threshold the derived magnitude to a certain value. The mathematical and technical concept is extensively described in Appendix A.

6.1.2.Active Contour Models (Snakes)

The concept of Active Contour Models consists of 4 simple steps:

- Define an area of interest within an image plane
- Create a contour function that is lying around this area
- Define energy functions the force the contour in a certain direction
- Update the contour according to the defined forces until the total force (or energy) is zero

In other words, have a contour shrink until it is totally lying on the outer intensity boundaries of the object. The hard part is to define energy functions that make the contour move towards the object boundaries. The energy function of a snake consists of two parts, the external and the internal energy, so:

$$E_{\text{snake}} = E_{\text{internal}} + E_{\text{external}}$$

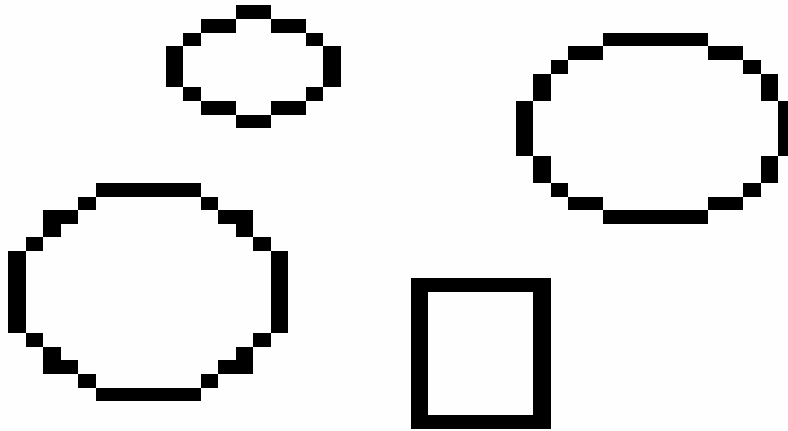
The internal energy is the part that depends on intrinsic properties of the snake, such as its length or curvature. The external energy depends on factors such as image structure, and particular constraints the user has defined. So, the external energy forces the snake out and the internal energy forces it in. When $E(\text{internal}) = E(\text{external})$ the snakes has reached its destination.

6.1.3. Filling

The concept of filling is quite the opposite of the snake method. A certain amount of fill points are “seeded” in the image plane. They are growing in every direction until an intensity boundary of some level (threshold) is found or a user constraint is reached. After all seeds have evolved into the acquired shapes, the boundaries of the fills should be the boundaries of the required objects.

6.2. Object separation

After an image has been scanned a certain set of points remains. Assuming that we do not know the amount of objects to be found on beforehand, the result set should be divided into subsets. For example, if a person were to see the following image:



He or she would know that there are 4 separated subsets, of which three are circular and one is rectangular. A computer however, doesn't know the connection between any of these points, it is just a collection. In order to classify the objects in an image correctly, the result set should be clustered. There are several ways to cluster a certain set of points, the most commonly used algorithms:

- Hierarchical
- Self Organising Maps (SOM)
- K – means

are described in the following chapters.

6.2.1. Hierarchical

The concept of hierarchical clustering is to produce a hierarchical tree in which the nodes represent subset of a certain set S.

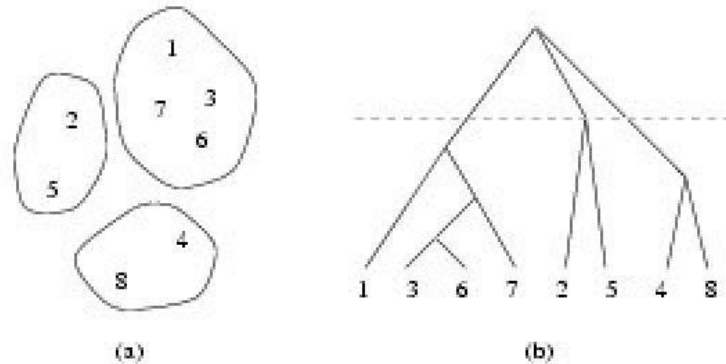


Figure 6.2.1.1. Hierarchical clustering

Each level of the tree represents a partition of the input data into several (nested) clusters or groups. This means, that in the example, point 3 and 6 are a subset, this subset combined with 7 creates the following subset, combined with 1 the following subset etc etc.

The way to determine whether certain points should be merged into a subset is to define a *merging cost function*. This result of the function defines the *distance* between two points. The two points with the shortest distance are merged. The algorithm can be summarized by the following steps:

S is a list of elements

T is a tree

1. Place each instance of S in its own cluster (singleton), creating the list of clusters L (initially, the leaves of T):

$$L = S_1, S_2, S_3, \dots, S_{n-1}, S_n.$$

2. Compute a **merging cost function** between every pair of elements in L to find the two closest clusters $\{S_i, S_j\}$ which will be the cheapest couple to merge.
3. Remove S_i and S_j from L.
4. Merge S_i and S_j to create a new internal node S_{ij} in T which will be the parent of S_i and S_j in the result tree.
5. Go to (2) until there is only one set remaining.

6.2.2. K – means clustering

The concept of the K-means clustering algorithm is to divide a certain set S into k (a predefined integer) subsets. The implementation is fairly simple and therefore quite fast. K cluster centroids are randomly determined. For each point in S the distance to every cluster centroid is calculated. The point is then assigned to the closest cluster. After all points are assigned to a cluster, the centroids are recalculated. The new centroid is the average of all the

points that are assigned to it's cluster. This process continues until convergence (no more points are moving from one cluster to another).

6.2.3. Self Organising Maps

Self-organizing maps (SOMs) are a data visualization technique invented by Professor Teuvo Kohonen which reduce the dimensions of data through the use of self-organizing neural networks. The problem that data visualization attempts to solve is that humans simply cannot visualize high dimensional data as is so techniques are created to help us understand this high dimensional data. The way SOMs go about reducing dimensions is by producing a map of usually 1 or 2 dimensions which plot the similarities of the data by grouping similar data items together. So SOMs accomplish two things, they reduce dimensions and display similarities.



figure 6.2.3.1. 3-dimension colours displayed in 2-dimension coordinates ordered by similarity

6.3. Transformations

Because the head of the AIBO can be moved and the camera is mounted on it, the camera result is almost always rotated. Furthermore, the percepts that need to be implemented are to be defined relative to this rotated axis-system.

The clusters also need to be defined according to the rotation angle because the classifiers are expecting non-rotated results. In other words, the cluster points need to be stored relative to the rotated axis-system.

There are several ways to transform coordinates from one axis-system to another.

1. If the axis-lines of the other axis-system are known in the axis-system every point can be transformed by calculating the distance between point and axis-line.
2. Rotate and translate the original axis-system until it matches the other one. Thus, translate and rotate the coordinates in the opposite direction.

The first solution is only working in one direction unless axis-functions are known in both axis-systems.

The second solution requires some explaining. Every point in matrix one should be able to be transformed to a point in matrix two and the other way around. To do this, the axis-system has to be rotated and translated (moved according to dX and dY).

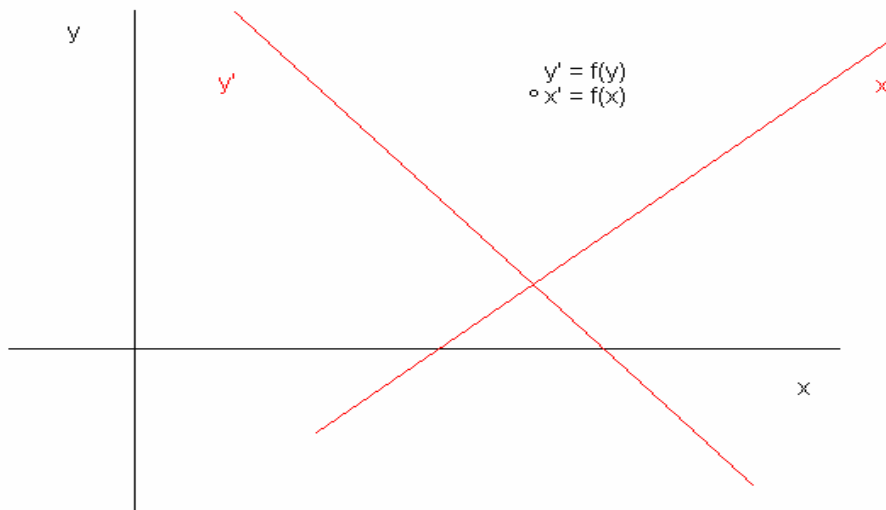
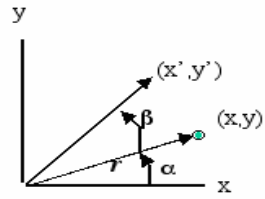


figure 6.3.1. axis system translation

The reason why the axis-system first needs to be rotated and then translated is that the translation direction should be the same for each axis-system. Matrix rotation are done as follows:

$$x = |r| \cos \alpha$$

$$y = |r| \sin \alpha$$



$$X' = |r| \cos (\alpha+\beta)$$

$$= |r|(\cos \alpha \cos \beta - \sin \alpha \sin \beta)$$

$$= (|r| \cos \alpha) \cos \beta - (|r| \sin \alpha) \sin \beta$$

$$= x \cos \beta - y \sin \beta$$

$$y' = |r| \sin (\alpha+\beta)$$

$$= |r|(\sin \alpha \cos \beta + \sin \beta \cos \alpha)$$

$$= (|r| \sin \alpha) \cos \beta + (|r| \cos \alpha) \sin \beta$$

$$= y \cos \beta + x \sin \beta$$

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos \beta & -\sin \beta \\ \sin \beta & \cos \beta \end{pmatrix} \begin{pmatrix} r \cos \alpha \\ r \sin \alpha \end{pmatrix} = \begin{pmatrix} \cos \beta & -\sin \beta \\ \sin \beta & \cos \beta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

figure 6.3.2 Matrix rotation

After the two axis-systems are aligned, one of them can be translated to match the other one. This process is to be executed for each result point and can be done forth and back.

6.4.Object classification

After both boundaries and clusters have been determined, the class of the object should be derivable. There are several ways to classify an object. The most simple object features, such as size or color, can be taken and used to derive an object's class, but if we want to get a more robust classification we need to use the whole package of object features, which includes the shape of the object. The best way to take n-dimensional features and get one result is to use a neural network.

Neural networks are computer systems that are based on the parallel architecture of animal brains and consist of neurons and inter neuron connections also referred to as weights.

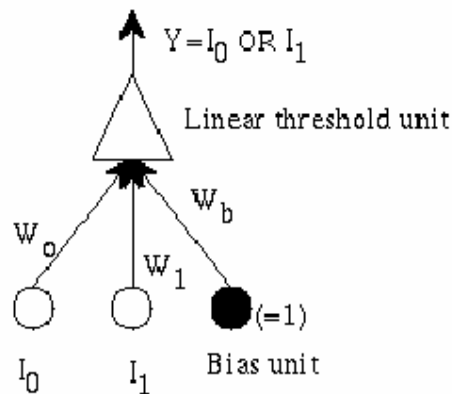


figure 6.4.1. a single perceptron

The advantage of neural networks over “regular” computer systems is that they are capable of getting the right results from noisy data and that they are able to adapt to circumstances (i.e. they can learn).

The technique neural networks use to learn a certain solution is training. The network is set up with random weights and then training data is inserted into the network. With each set of training data, a set of “true output data” is delivered. The network computes the input data and compares the derived output with the desired output. The difference (error) is then transferred back into the network and every weight is adapted to the following equation:

$$\Delta W_i = \eta * (D-Y).I_i$$

where η is the learning rate, D is the desired output, and Y is the actual output. The learning rate determines the speed at which the network adapts.

Figure 6.4.1 displays the simplest of neural nets which is able to separate 2-dimensional data with a line. In this model (single perceptron) the amount of input units determines the dimension of the data and the threshold unit determines the function of the separation line. This can be used to determine whether n-dimensional data belongs to class A or B. When the data is not linearly separable, such as the classic XOR problem, another, more complex network is required.

The most commonly used non-linear supervised networks are back-propagation networks (BP) and radial basis function networks (RBF). They both consist of more than one input and one output layer.

The RB network has an undefined amount of hidden layers. The more complex the data to be separated, the more hidden layers and the more nodes in the hidden layers. A typical RB network is displayed below.

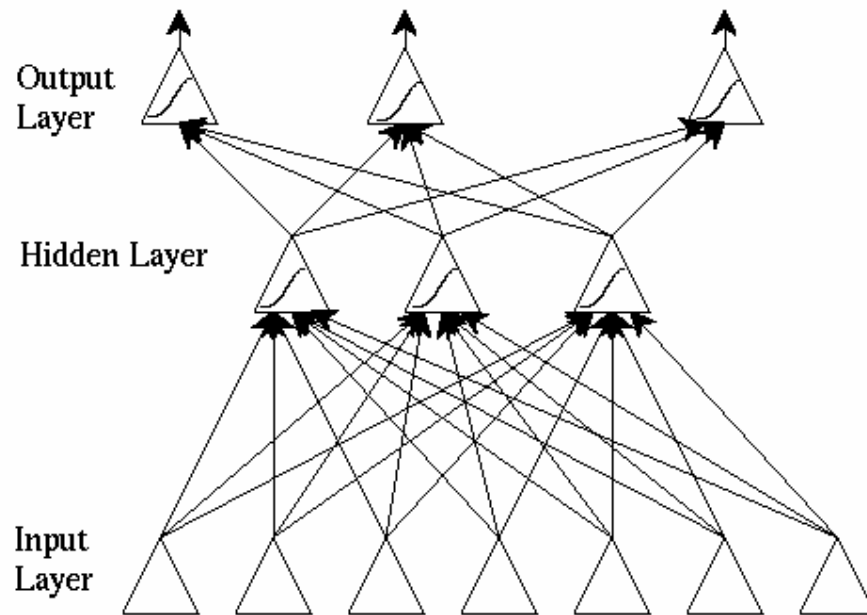


figure 6.4.2. Back propagation network

The activation (input -> output function) for hidden and output nodes is the sigmoid function.

$$Y = 1 / (1 + \exp(-k \cdot (\sum W_{in} * X_{in})))$$

Where Y is the node's output. This can be displayed as:

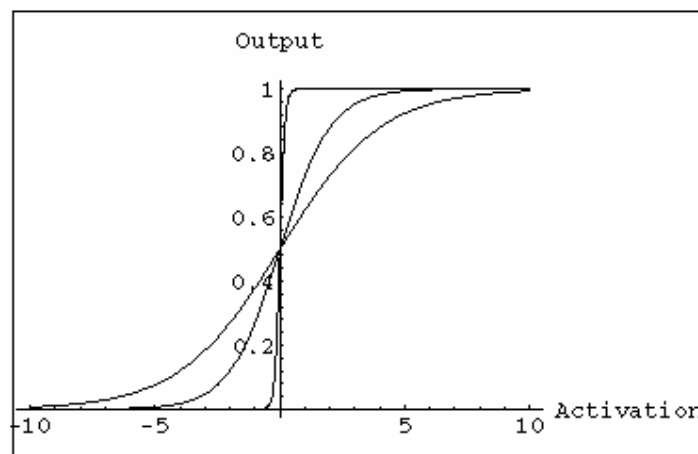


figure 6.4.3. Back propagation activation function

Because of the activation function, the backward pass has to recompute errors over the layers, thus taking the first derivative of the sigmoid function.

The RBF function has only one hidden layer and has a centred activation function (i.e. the closer the input values are to the centre of the function, the higher the output). This function usually is a Gaussian.

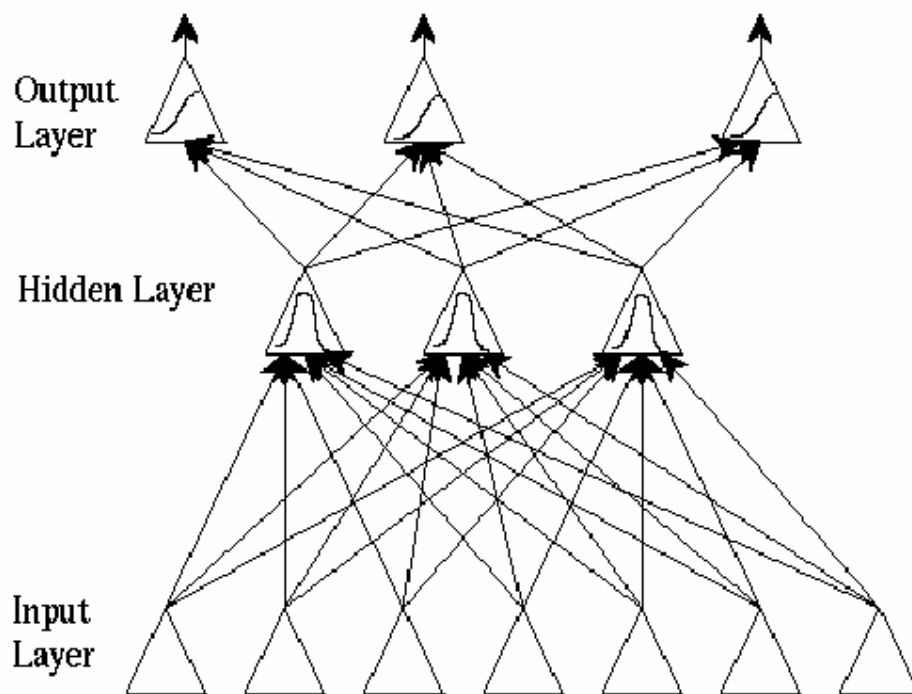


figure 6.4.4. radial basis function network

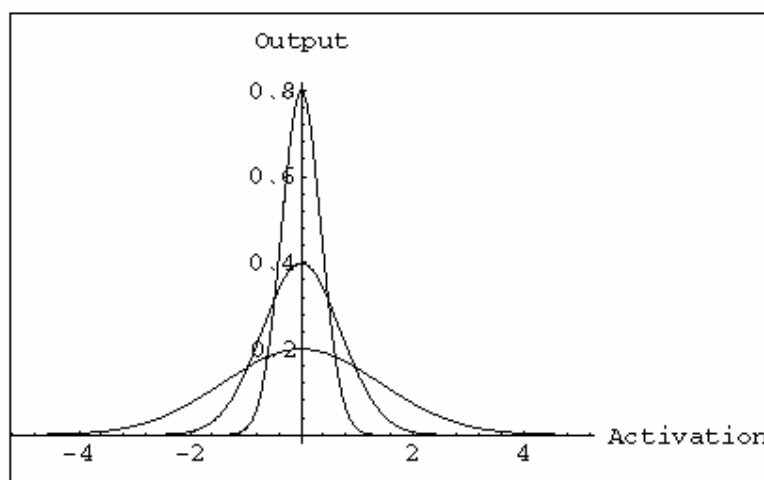


figure 6.4.5. radial basis activation function

6.5.Pre-classification processing

Before passing data to a neural network it is necessary to determine whether the order and the type of input is suitable to be classified. The simplest input is to use consecutive pixel values as input. The biggest problem is that the cluster size is never the same, so the neural net should have a variable amount of input nodes.

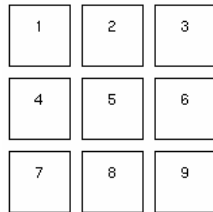


figure 6.5.1. Consecutive pixel values

The next solution is to pre-set a certain amount of inputs. This amount is equally divided over the input cluster. The average value of the determined pixel gradient magnitude and all its neighbours is taken as input.

The following figure describes a grid where 6 input values are equally divided. The green circles indicate the pixels that are taken into account when calculating the average pixel value.

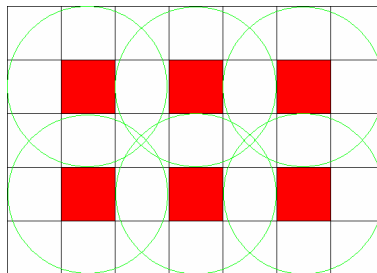


figure 6.5.2.

equally divide sample pixels (red), take neighbours and calculate average (green)

A third way of preparing input pixel is to take a pre-defined number of edge pixels along the outer boundary and use the angle between consecutive pixels as the input for the neural network.

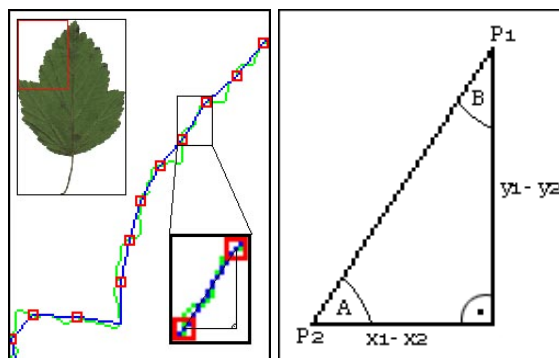


figure 6.5.3. consecutive pixel angles

7.Considerations

After researching all possible methods to implement the three steps proposed in chapter 13, these methods are compared in this chapter. The comparison factors are:

- performance
- quality
- usability for cause on platform

The best way of estimating the performance of an algorithm is to determine its order. The order depends on how the amount of comparisons increases compared to the increase of the amount of elements. For instance, if one were to compare the value of every element to a certain value the order would be $O(n)$. If the distance between each element were to be determined and compared to a certain value (min / max distance estimation) the order is $O(n^2)$. Any constants are left out of the equation (independencies of N are not taken into account).

7.1.Object boundary separation

The discussion on which boundary separation algorithm to use is mainly based on usability and less on performance. Snakes and filling techniques are quite fast, however, these techniques do either have unknown input variables or potential large errors.

The problem with snakes is the location of the initial deformable model needs to be set and this needs to be done on some basis. This basis can be made by a human supervisor or some pre-processing steps. Human interference is not possible and the pre-processing steps are just the thing that this technique was to be used for.

Filling has the potential of filling the whole image because there are gaps in the boundaries on its filling-path.

This leaves us with edge detection. The disadvantage is the speed of the algorithm. Every pixel needs to be convolved in order to find all edges.

7.2.Result separation

There was only one method described on this subject. Result separation by colour band is the easiest way of hard-coded differences between objects. In later chapters, better solutions will be proposed.

7.3.Clustering

The order of K-means clustering is $O(n)$, because for every point, the distance to each cluster centroid is calculated and compared to the current minimal value. Since K is constant it is not relevant and the amount of loops before convergence is also independent of the amount of elements.

The problem with K-means clustering is that the amount of clusters is set before the algorithm runs. This makes the algorithm fast but not robust.

A SOM also has order N (distance to one sample), however, after the minimal distance is found, neighbour adaptation needs to be done and the amount of loops before convergence is far larger than the amount in K-means.

Furthermore, SOMs are not meant for clustering two dimensional data but to decrease the dimensions of high dimension data in order to derive similarities between a group of data.

Hierarchical clustering has order $O(n^2)$ (min distance calculation between every point), but the amount of elements decreases as the amount of loop increase. The final result, however, is two precise for our cause. Only first level subsets are used to determine the element cluster.

To conclude the above, none of the algorithms suite the cause of dividing N elements into the right amount of clusters. The algorithm is either to slow or has the wrong output.

The solution is a variant of K-means. The new algorithms sets K to an amount that is always larger then the expected amount of clusters. If a cluster doesn't have any elements it is removed. After convergence the clusters that are to close (some threshold) to each other are merged into one. This way, the right amount of clusters remains while the order of the algorithm remains $O(n)$.

7.4.Pre-classification processing

The choice between direct pixel mappings and the other two solutions was fairly easy since the first solution isn't possible to create because neural nets cannot cope with a variable amount of input nodes.

So now the choice is between the "divide and average" algorithm or the "divide along the line and take the consecutive angles as input" algorithm. The quality in this matter is how the structure of the input reacts on scale and rotation changes. If the rotation angle of the object is known and the cluster adapted, both algorithms will have similar results independent on scale and rotation. However, when the rotation is not known, the structure of the divide and average algorithm drastically changes whereas the second algorithm's structure remains the same. Therefore, the chosen solution was the angle algorithm.

7.5.Classification

The choice of classification network is not too relevant. Because an infinite amount of objects is not linearly nor quadratically separable the choice to be made is between back propagation and radial basis function networks. The difference between the networks is the amount of hidden layers and the activation function. Since we don't know anything about the estimated values for the input nodes, the sigmoid function proves to be more common (radial basis functions needs a defined centre). Therefore, the back propagation network is the chosen as the classification method.

The classification can now be done in two different ways:

1. The amount of output nodes defines the amount of objects. After passing through the network the output node with the highest value defines the class.
2. For every object there's a network with one output node. The classification process exists of passing the input values through every network. The network with the highest output value defines the class.

The second method has a lot of advantages over the first, such as expandability and quality. However, this method is also particularly slower then the first and is therefore not the chosen solution.

The amount of input nodes should be more than the estimated amount of different angles. This amount is set to 20. The hidden layer amount is based on the complexity of the object shapes and is, after testing, set to 50.

8.Design

The global of the framework on which this image processor is built is caught in the following schedule.

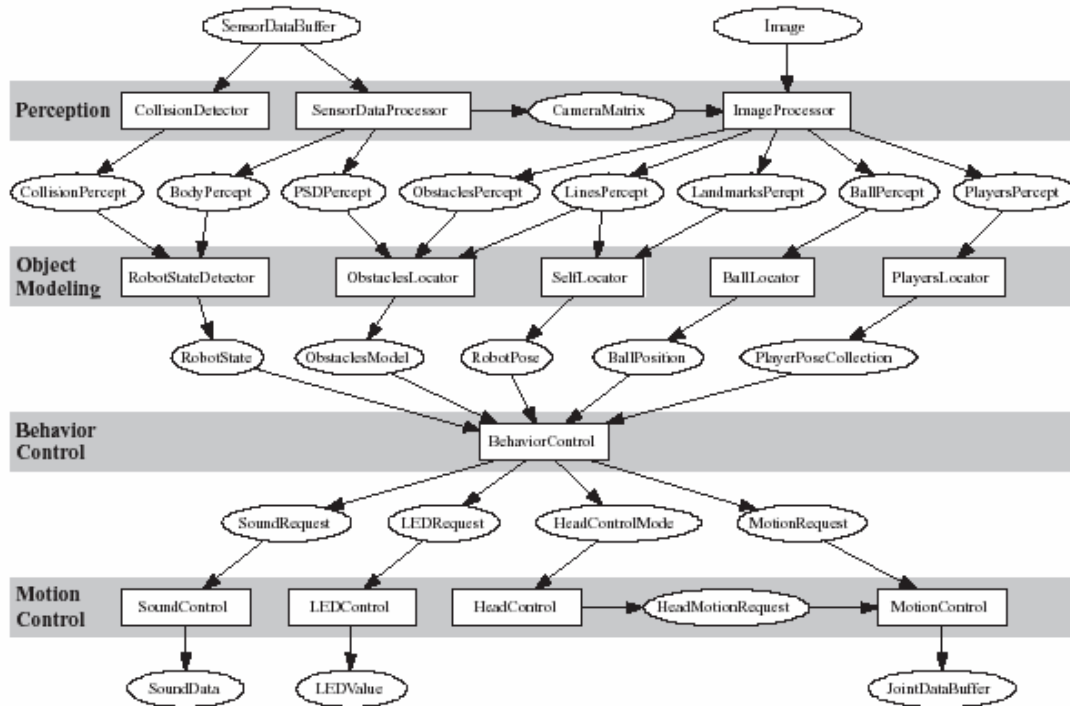


figure 8.1. German team global software schematic

As can be seen, the framework has a very modular fashion. One of these modules is the image processor. A small part of the framework has been worked out as a class diagram which is displayed on the next page. The upper part of the schedule display the linkage with the framework and the classes where global data is stored (such as the percepts and the current self locator). The image processor interfaces are initialised by the core of the framework and contains references to all the necessary global instances. By passing a reference of the instance of this class to all the objects that have anything to do with image processing, some sort of shared memory region is created. For instance, the image processor determines locations of several objects and implements the percepts whereas the self locator is concurrently reading the percepts and using this data to determine the robots location relative to the flags.

The specific image processor holds a list of the super classes of the different solutions (ClusterMachine, FilterMachine, ClassificationMachine). These instances are filled with one of the child classes. These classes all overwrite a certain function in the super class. This is done to improve the ease of switching between solutions.

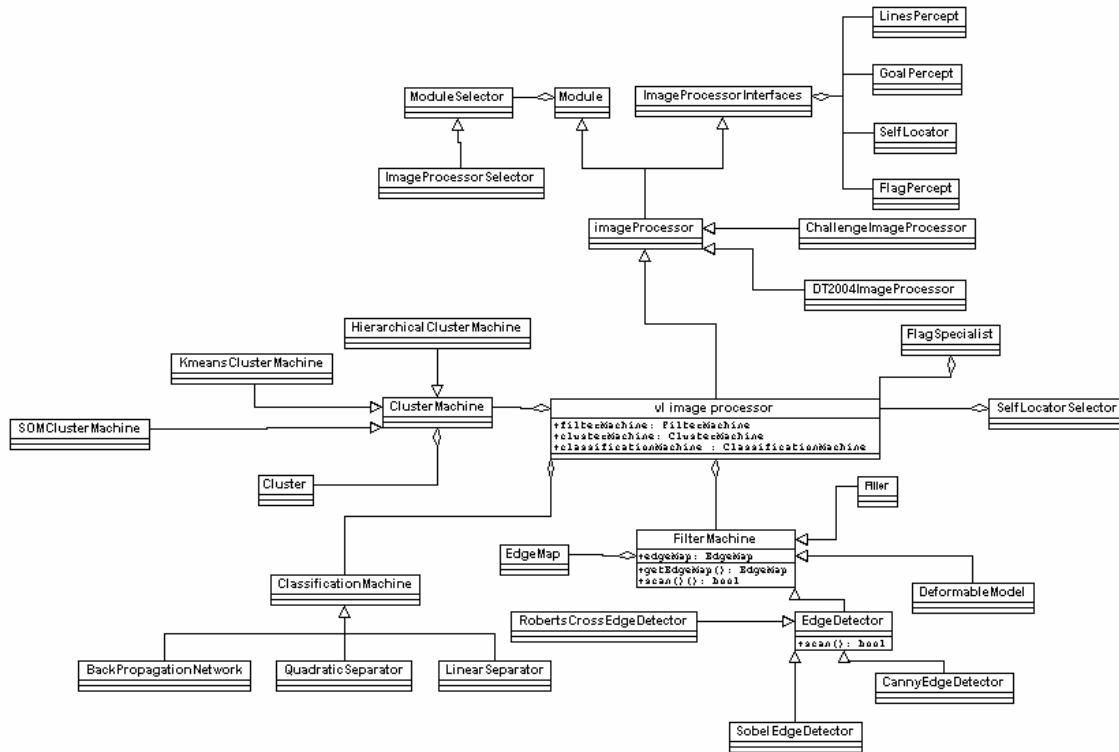


figure 8.2. image processor class diagram

9.Implementation

As can be seen in attachment I, the implementation phase was combined with several research steps. This was done to prove the usefulness of the studied cases and to lay the groundwork for the final software module. The pre-fabricated modules were later used as separated objects or algorithms.

The design chapter describes the final result (output) of the image processor: the implementation of the robots percepts (ball / flags / lines / goals). The advantage of using an external software project, and in particular a modular one, as the framework for something new is that developers can use black-box programming. This means that once the input and output structures are known, the development team can focus on one module without bothering with knowledge of all the related modules.

In my case, this meant that once I knew how to hook my image processor into the framework and how to implement the percepts in the correct manner, I could focus on the internal processes of the image processor.

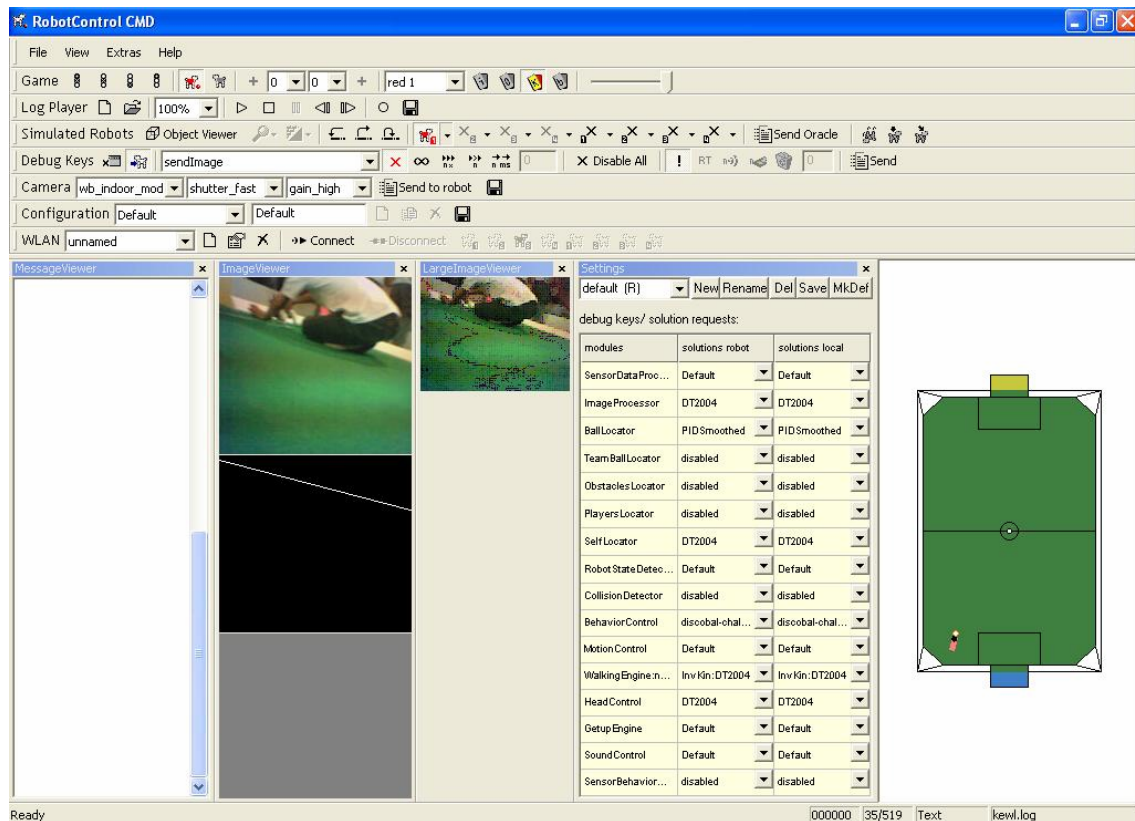
10. Tests

10.1. Debug surrounding

The problem with an external platform (a robot) is that testing costs a disproportionate amount of time, because software has to be transferred from the base computer to the AIBO, the robot has to be booted and a wireless connection has to be set up. If for some reason, the software encounters run-time problems, the robot crashes and cannot send debug data through the wireless. In other words, it is very hard to debug the created software.

Fortunately, the framework (DT2004) consists of two parts, one part that can be run on the robot (and is compliant with Aperios regulations) and a part that can run under Windows and is meant to debug the data that is sent by the robot. The second part, robot-control, is also capable of sending action commands to the robot(s) and can be seen as some kind remote control.

The beauty of this system is that all separate modules are the same for robot and robot-control and can be used in the same way. In other words, when the robot is sending images to the robot-control, the local program can use the same image processors as the robot to analyse the images and can also display real-time debug information.



The situation described above still uses the robot and is therefore relatively time consuming. The local program is capable of storing all the debug information (video stream, sensor data etc.) in a log file which can be loaded afterwards so the robot is no longer required to debug the written software.

There is however some danger in this method of testing the software. The modules might be the same but the platform is not. When testing is done on the base system, there is much more

processing power available and this could lead to problems when the software is run on the AIBO. Therefore it is very important to do some regular test on the robot as well.

10.2. Test runs

The test cycles consisted of a couple procedures. At first, the only goal was to determine object classes under varying lighting conditions. This was done by placing the AIBO in a pre-defined surrounding and covering the lights in random order.

The next step was localization relative to the defined flags. This meant implementing the percepts and defining real object sizes.

The final test was the “variable lighting challenge” on the Robocup 2004 in Lisbon in which the goal was shoot the ball in the goal with four corner lights being lit or turned of dependent on a certain schedule. Two turned-off AIBO’s were placed on the field to stop the ball from going in the goal.

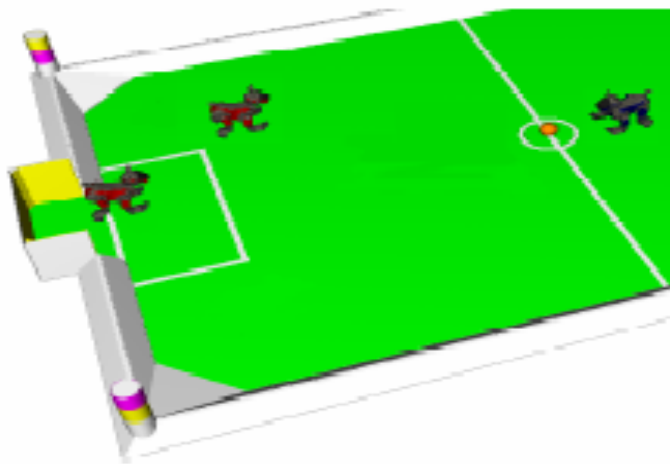


figure 10.2.1. variable lighting challenge

10.3. Test results (issues)

10.3.1. Performance

The major problem that came up after testing the software on the robot was obviously performance. Since the initial process of determining the edges takes such a huge amount of processing time (due to the fact that the edge detection process needs to do a full scan of all the pixels), there’s hardly any time left for additional processes like clustering, perception and concurrent actions (walking, shooting etc.).

10.3.2. Local variable lighting

The problem with thresholding based on local lighting solutions is that in areas with a large amount of static the light intensity is usually low. This leads to a low threshold and therefore more static.

The next problem with this kind of object detection is that the gradient magnitude also depends on the background. For instance, an orange ball against a green background has got a different intensity gradient (on a certain band) then when it is placed against a blue background. This can of course be corrected but not in a global fashion.

11.Solutions (reference for future work)

11.1.Performance

The problem with performance has always been an issue when it came to robot vision. The fact remains that there's always a large amount of pixels to be scanned and this amount increases while technology moves on and better camera's are placed on nowadays robots. There are many ways to solve performance problems, the easiest one being the increase of computer power. This is of course quite a simple way that can solve every problem and also one that is not always an option (like in AIBO-robots).

We are looking for smarter ways to improve the speed of the detection algorithm. Let's look, once again, at our own way of interpreting our "camera input". Do we have an endless amount of processing power available? Most doubtfully.

First of all, we cannot look at everything in our range with the same detail. Only the centre of our vision plane is sharp and quickly becomes more fuzzy to the sides. In other words, the pixel density function is not smooth like in camera's but answers to a kind of Gaussian function. This is saving a lot of time, because the objects on the outer bounds of our vision range are only scanned superficially. If we're interested in its details we just have to look at it. Next, we remember object features and location. The same rule applies here; if we need details, we look at the object.

A last "handy feature" of humans is our capability to merge all the input of our senses and create one percept of it. We can estimate the class and distance to an object by what we hear, and then fine-tune it by looking at it.

All the points written above can be transferred to a computer system. The transformation to the Gaussian density function needs to be at the first layer of the software. Because looking at an object can be done in three ways; moving the robot's head, moving the robot's camera and setting the centre of the Gaussian function to the estimated location of the object in the projection plane. Therefore, the system should be able to set the centre of the density function for the next frame.

Another important issue is to keep the real coordinates separated from the transformed coordinates and to define a transformation function (depending on pixel radius).

Remembering object locations requires fusion between odometry and vision. The location of the object relative to the robot changes as the robot moves and therefore the "real" location of both robot and object need to be known as well as the pose of the robot (angle to the real world). Since robots always have small errors in their measurements and because objects seen in the past should not have too much impact on calculations in the present there's need for an uncertainty factor that increases in time.

The last method of determining global locations of objects by using other senses can also easily be implemented. This is however something that is running concurrently with the vision system and has no impact on the actual vision module structure.

11.2.Variable lighting

When the solution of region based scanning is used, the problem of extra static in non-interesting regions is automatically solved. The variable gradient, however, needs a totally new approach. Test results showed that it is not possible to define a constant threshold function if the background of the object changes. The reason why these constants were

defined is to separate objects from each other. This is something that was done because the clustering algorithm needs separate result sets to define cluster regions. The problem is the fact that whenever an object is placed against a background with different colours, the gradient magnitude of the object boundary is always variable and can therefore not be used to determine the object's boundaries. The only thing that remains then is the object's colour and that is just what we want to avoid.

There are two ways to solve this problem:

1. Define relative colour schedules per object
2. Assume that the occurrence of different background colours isn't significant enough to disturb the result and split the total range of gradient magnitudes into a histogram of gradient magnitude ranges.

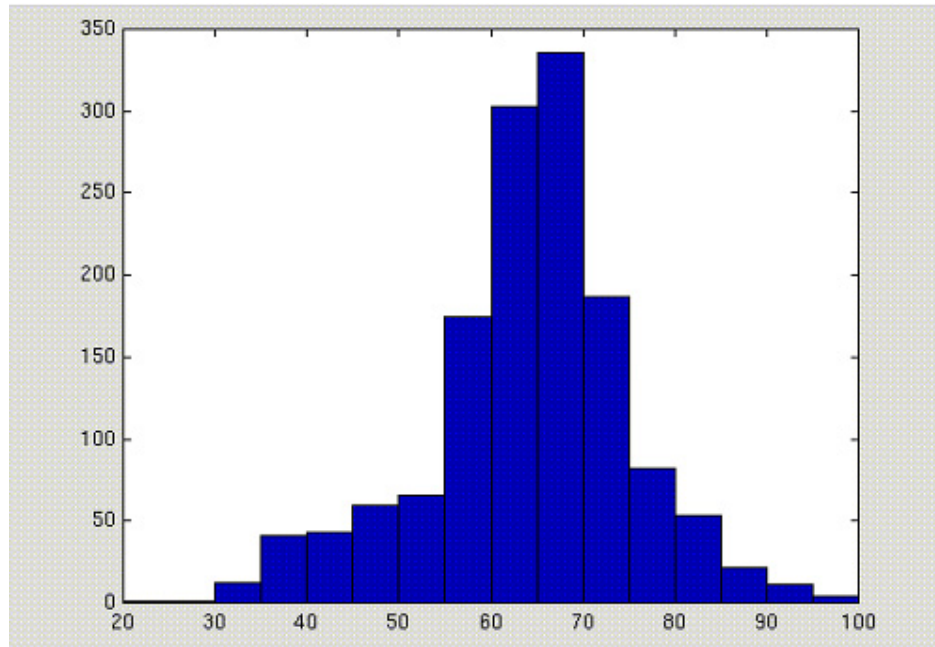


figure 11.2.1. gradient magnitude histogram (value – occurrence)

In other words, when this situation occurs, ignore it and wait for the right background.

3. The third solution is a mix of the second solution and the clustering step. The gradient magnitude is seen as a distance measure in the clustering algorithm. In other words, clustering becomes a 3-dimensional minimal distance calculation algorithm. The minimal distance is the Euclidean distance between two points in the 3D space.

The first solution is working with relative colour definitions. This is based on the answer to the question “why is yellow still yellow when lighting conditions slightly change?”. The point is that in the RGB colour space yellow is defined by a lot of red and green and some blue. This can be placed into a yellow function:

$$\text{Yellow}(\text{low}) = R * G / (B + 100)$$

$$\text{Yellow}(\text{high}) = R * G / B$$

This way, all base colours can be relatively defined and objects can be classified by their relative colour sequence. However, this is taking only a small step ahead instead of the more robust (shape based) solution.

Therefore, the second solution proves more useful to the cause. The solution is to split the edge map into a couple of layers by their gradient magnitude range. This looks something like the image below. The magnitude ranges that do not contain a certain amount of result points are not taken into account (static filtering). After layering, each layer is clustered and classified by shape.

The downside of this solution is that objects can fall into different ranges and therefore be classified erroneously thus causing unwanted static.

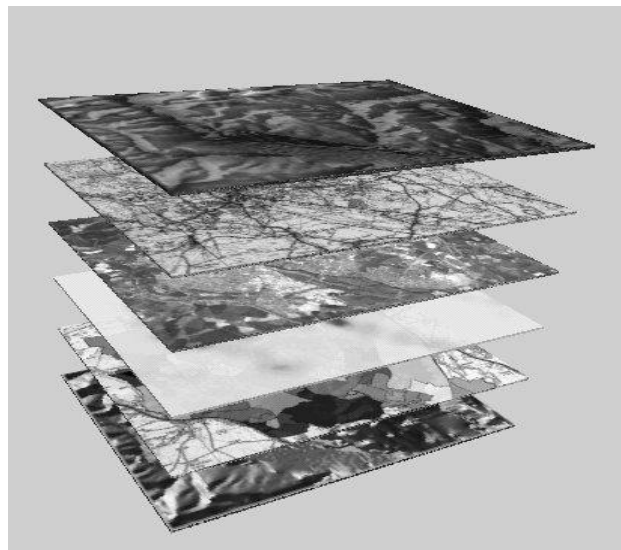


figure 11.2.2. gradient magnitude layers

The third solution doesn't layer the gradient magnitude but makes this a third distance dimension. When plotted, this would look something like the image below. The cluster algorithm remains the same. K clusters are randomly seeded in the 3D space and for each point the minimal Euclidean distance is taken. This is the optimized form of the previous solution.

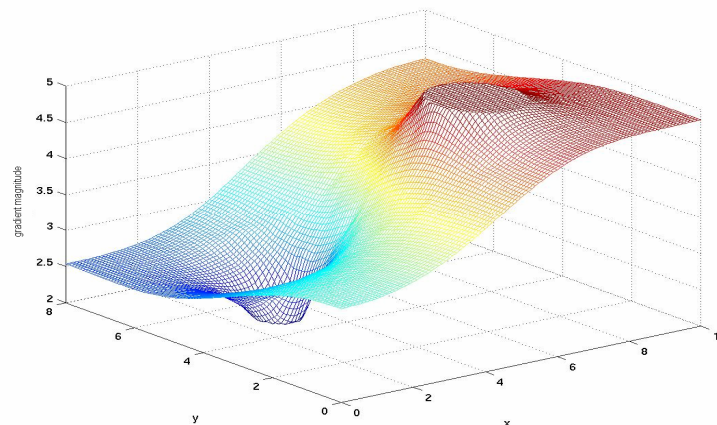


figure 11.2.3. dynamic distance estimation

12. Conclusion

The answer to the question “is it possible to recognize object without using colour segmentation?” is yes. After looking at the human vision system, the proposed solution was to use the logical steps:

- determine object boundaries
- separate different objects
- use previous encounters of the object to determine it's class

The implementation of these steps was, after researching a scalar op possible solutions, done by edge-detection, K-means clustering and a back-propagation neural network. The results of these chosen methods were promising, but were too heavy to perform well on the designated platform (ERS-7).

The final chapters described a couple of methods to increase performance and to solve problems that were found in the test period.

The second question that was stated in the introduction of this paper was whether it would be possible to create a vision system that allows the robot to walk around in the world.

Obviously, it is possible to create such a system, the question however is whether the proposed system is able to handle all the challenges of walking around in an undefined surrounding.

If the current (optimized) solutions were to be extended by placing a learning algorithm in the classification module the robot would actually be capable of a form of vision that resembles human vision. It could ask a supervisor (human / database) what the class is of an unidentified object and so create its own database of object classes. Once again, the lack of performance would eventually catch up with this method of creating a “world view” since the database (set of neural networks) would soon get too big.

To conclude, human vision is still far away but current vision systems are more and more capable of giving robots the opportunity of joining humans in their daily lives.

References

Websites

[An introduction to neural networks](#)

[Neural networks](#)

[Self organizing maps](#)

[Clustering methods](#)

[Different techniques of data clustering](#)

[DMS tutorial](#)

[Clustering](#)

[Canny edge detection tutorial](#)

[Feature detectors](#)

Papers

[German team description paper](#)

Sony's AIBO ERS-7 papers

Sony's OPEN-R SDK 1.1.5. description papers

Books

Title

author

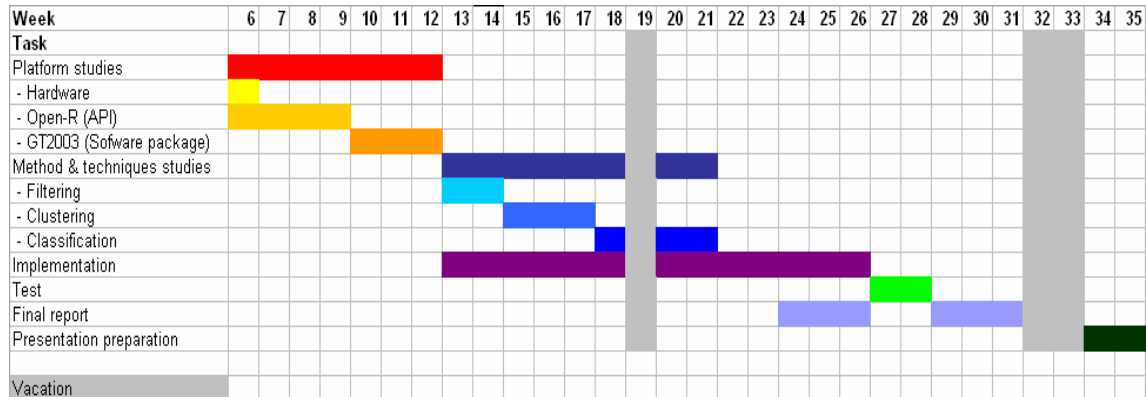
The essence of neural networks

Robert Callan

Attachments

I. Procedure & Time scheme

The follow time box describes the arrangement of time in this project.



As can be seen, the project starts with a definition study of the AIBO platform. Studies of methods and techniques are done concurrently with the implementation phase. This is not the usual approach in a software project, however, sample programs are very useful when it comes to understanding certain methods and can, in a later stadium, be used as modules or algorithms in the total program.

II. Open-R methods

Sony provided the AIBO with Aperios, a platform which is built by Sony and optimized for use on the AIBO. To provide handles to hardware components, an Software Development Kit (SDK) is added. This SDK is called OPEN-R and is built to expand the capabilities of entertainment robots. The interface is layered and optimized to enable efficient development of hardware and software for robots.

The OPEN-R SDK discloses the specifications of the interface between the ‘system layer’ and the ‘application layer’.

The SDK has the following features:

- Modularized software and inter-object communication

OPEN-R software is object-oriented and modular. Software modules are called "objects" (specifically, "OPEN-R objects").

In OPEN-R, robot software is implemented so that processing is performed by multiple objects with various functionality running concurrently and communicating each other via inter-object communication.

Connections between objects are defined in an external description file. When the system software boots, the description file is loaded and used to allocate and configure the communication paths for inter-object communication. Connection ports in objects are identified by the service name, which enables objects to be highly modular and easily replaceable as software components.

- Layered structure of the software and services provided by the system layer

The OPEN-R system layer provides a set of services (input of sound data, output of sound data, input of image data, output of control data to joints, and input of data from various sensors) as the interface to the application layer.

This interface is also implemented by inter-object communication.

These services enable application objects to utilize the robot's underlying functionality, without requiring detailed knowledge of the hardware devices that comprise the robot.

The system layer also provides the interface to the TCP/IP protocol stack, which enables programmers to create networking applications utilizing the wireless LAN.

Open-R objects

OPEN-R application software consists of several OPEN-R objects. The concept of an object is similar to one of a process in the UNIX or Windows operating systems with regard to the following points of view:

1. An object corresponds to one executable file

An object is a concept that only exists at run-time. Each object has a counterpart in the form of an executable file, created at compile-time. Source code is compiled and linked to create this executable file. Then, the file is put on an AIBO Programming Memory Stick. When AIBO boots, the system software loads the file from the AIBO Programming Memory Stick and executes it as an object. (An executable file has a filename with an extension of ".bin".)

2. Each object runs concurrently with other objects.

Each object has its own thread of execution and runs concurrently with other objects in the system.

3. Objects exchange information using message passing

An object can send messages to other objects. A message contains some data and a selector, which is an integer that specifies a task to be done by the receiver of the message. When an object receives a message, the function corresponding to the selector is invoked, with the data in the message as its argument. An important feature of objects is that they are single-threaded. This means an object can process only one message at a time. If an object receives a message while it is processing another message, the second message is put into the message queue and processed later.

Below is the typical life cycle of an object:

- 1 Loaded by the system
- 2 Wait for a message
- 3 When a message arrives, execute the method corresponding to the selector specified in the message. Possibly send some messages to other objects.
- 4 When the method finishes execution, go to step 2.

Note that this is an infinite-loop: an object cannot terminate itself. It exists while the system is activated.

4. An object has multiple entry points

Unlike an ordinary programming environment in which a program has a single entry point "main()", OPEN-R allows program to have multiple entry points. Each entry point corresponds to a selector as explained above. Some entry points have purposes that are determined by the system, e.g. initialization and termination. Other entry points have purposes specific to the object.

Inter-object communication

Software controlling entertainment robots typically consists of various objects, each with its own tasks, such as image recognition, speech recognition, motion control and motion generation. They communicate with each other while they perform their tasks. In OPEN-R, this communication between objects is called “inter-object communication”.

The use of inter-object communication enables each object to be created separately and later be connected to other objects. This results in a very efficient development lifecycle.

When two objects communicate, the side that sends data is called the “subject,” and the side that receives data is called the “observer”. The subject sends a ‘NotifyEvent’ to the observer. NotifyEvent includes the data that the subject wants to send to the observer. The observer sends a ‘ReadyEvent’ to the subject. The purpose of ReadyEvent is to inform the subject that the observer is ready to receive data or not. If the observer is not ready to receive data, the subject does not send any data to the observer.

Fig2 shows a case where the subject of object A communicates with the observer of object B.

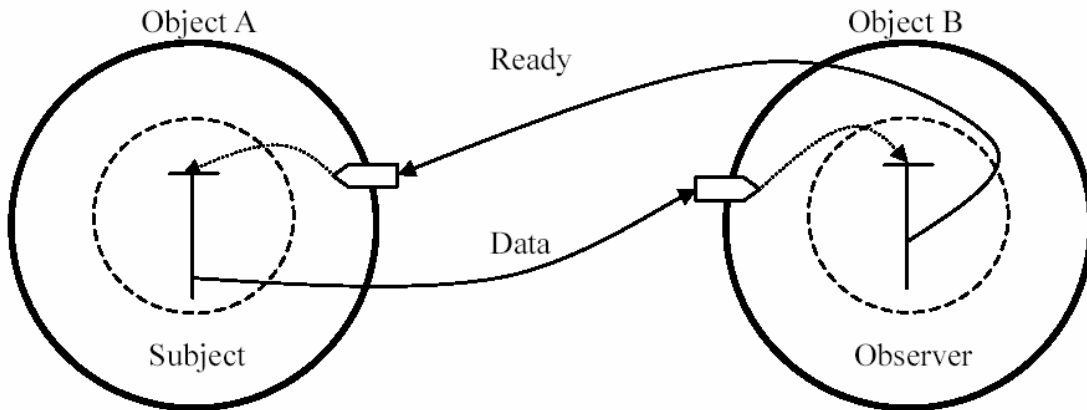


Fig2 Inter-object communication

Before the observer receives data from the subject, the observer must inform the subject of its current state. When the observer is in a state ready to receive data, the observer sends ‘ASSERT-READY’ to the subject. When the observer is in a state not ready to receive data, the observer sends ‘DEASSERT-READY’ to the subject.

When the subject receives ASSERT-READY from the observer, the subject starts to send data to the observer. After the observer receives this data and is ready to receive the next data, the subject sends ASSERT-READY again.

Shared memory

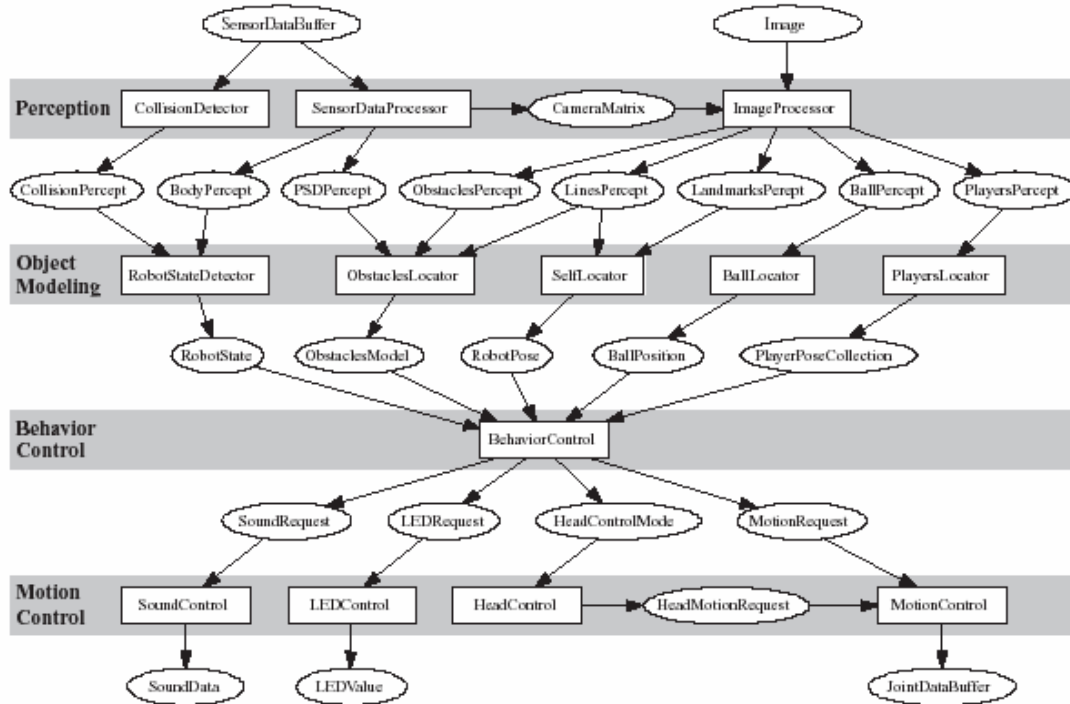
To speed up the process of object communication, messages can be placed in the shared memory region. Instead of sending the entire data structure from object A to B, the data structure is placed in a special region of the robots memory which can be accessed by every object. The sending object creates an structure (RCRegion) that contains a pointer to the allocated memory and sends this structure to the receiving object. This object retrieves the

data structure from the designated memory address and processes it according to the rules defined in stub.cfg (this file defines the method that is to be called when a message arrives from a certain connection).

The disadvantage of this method is that it cannot be used with objects that run on different platforms (i.e. different robots / computers). In other words, it cannot be used to send messages over the network.

III. Soccer software framework (German Team 2003 code)

Every program that's built to run on any robot needs to comply to a model that takes sensor data as input and sends output data to the actor (e.g. joints, leds etc.) interfaces. The GT package is described by the following schematic.



As can be seen, this model is highly modulated. This means that it is relatively easy to dedicate one module to a project team and have it alter it without having to edit the entire system, or a large chunk of it.

Furthermore, we can conclude that it is not possible to delete any module from the design, since the centralized behaviour control (the module that computes raw output on basis of pre-processed sensor data) needs all the data concerning the robot to compute the appropriate action.

Main modules

As describe above, the German Team has split the robot's information processing into *modules*. Each module has a specific task and well-defined interfaces. Different exchangeable *solutions* exist for many of the modules. This allows the four universities in the team to test different approaches for each task. In addition, existing and working module solutions can remain in the source code while new solutions can be developed in parallel. Only if the new version is better than the existing ones (which can be tested at runtime), it becomes the *default solution*.

This chapter describes most of the modules that were implemented. All modules can be put in one of the super model categories, being:

- Detection
- Cognition
- Motion

Since all technical methods of the GT package are described in their own paper (German Team – Robocup 2003), this paper only describes basic techniques on Cognition and Motion. Because the goal of this graduation project is to improve image processing methods, a larger explanation on Vision methods (one of the larger modules in Detection) is included.

Cognition

The concept of cognition is particularly easy. As can be seen in the schematics drawn above, the cognition module (behaviour control) takes five input structures and walks through a decision tree, a large pile of if-statements. If the tree reaches a final branch, a concurrent request is called. There are two main advantages to centralized computing:

- easy to adapt; *if there are some required changes in actions to be taken on the basis of the some set of input data, a developer only needs to follow the tree and change the concurrent action branch.*
- no collision control; *when thinking is de-centralized (or agent based), there is always a chance that the separate modules have different conclusions, leading to different actions and thereby crashing the robot. In such cases an extra module (collision control module) needs to compare module actions and take the one that is most likely the right one.*

Disadvantage of this model are:

- computation time; *the whole tree needs to be walked through to come to a conclusion while the most appropriate action is mostly based on outputs of one or two modules.*
- no rooms for smart shortcuts; *in human thinking and reacting, certainly when it comes to fast sport such as soccer, decisions are often based on reflexes. If reflexes were to be implemented in the code, they would be rather useless since the whole decision tree is walked through anyways.*

Motion

The basics of motion are to move a joint from position A to position B (e.g. define required joint angles for position B on time x and interpolated all required angles for the frames laying in between). If you look at motion in such a low level form, it is a lot of work to create fluent movements and to switch between clusters of time-angles, therefore there's a need for a higher level system on top of the low-level system to provide handles for other modules (in this case: behaviour control).

The German Team built a XABSL engine to describe complex motions. The *Extensible Agent Behaviour Specification Language* XABSL is an XML based behaviour description language. XABSL can be used to describe behaviours of autonomous agents. The runtime system XabslEngine executes the behaviours on a target platform.

Specific behaviour description languages prove to be suitable replacements to native programming language like C++ when the number and complexity of behaviour patterns of an agent increases.

XABSL simplifies the process of specifying complex behaviours and supports the design of both very reactive and long term oriented behaviours. XABSL uses hierarchies of behaviour modules called options that contain state machines for decision making.

The Architecture behind XABSL

The German team paper described the following on XABSL architecture:

In XABSL, an agent consists of a number of behaviour modules called options. The options are ordered in a rooted directed acyclic graph, the option graph. The terminal nodes of that graph are called basic behaviours. They generate the actions of the agent and are associated with basic skills.

The task of the option graph is to activate and parameterize one of the basic behaviours, which is then executed. Beginning from the root option, each active option has to activate and parameterize another option on a lower level in the graph or a basic behaviour.

Within options, the activation of behaviours on lower levels is done by state machines. Each state has a subsequent option or a subsequent basic behaviour. Note that there can be several states that have the same subsequent option or basic behaviour.

Each option has an initial state. This state becomes activated when the option was not active during the last execution of the option graph. Additionally, states can be declared as target states.

In the options above it can be queried if the subsequent option reached such a target state. This helps to check if a behaviour was successful. Additionally, each state can set special requests (output symbols), that influence the information processing besides the actions that are generated from the basic behaviours.

Each state has a decision tree with transitions to other states at the leaves. For the decisions the agent's world state, other sensory information and messages from other agents can be used. As timing is often important, the time how long the state is already active and the time how long the option is already active can be taken into account.

The execution of the option graph starts from the root option of the agent. For each option the state machine is carried out one times, the decision tree of the active state is executed to determine the next active state. This is continued for the subsequent option of the active state and so on until a basic behaviour is reached and executed.

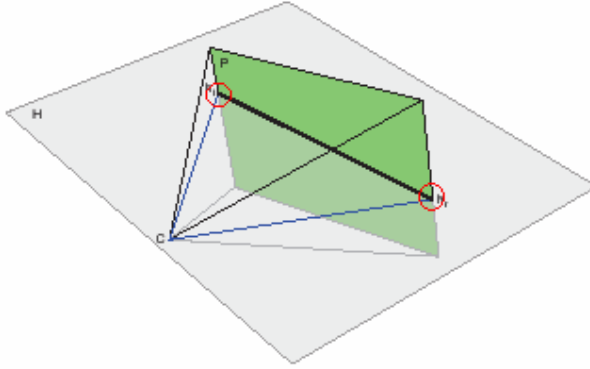
From this we can conclude that each motion has it's own XABSL agent which describes a certain amount of states. These states are chained to the low level angle definition, by doing this, states are interchangeable between motion objects.

Vision

The Vision module provides the technique that analyses the raw image data that comes from the robot's camera. This module is the most important one in the entire robot, since it is the basis for most calculations, such as self-localization or obstacle-localization. The output of the vision module fills the data structure *PerceptCollection*. A percept collection contains information about the relative position of the ball, the field lines, the goals, the flags, the other players, and the obstacles. Positions and angles in the percept collection are stored relative to the robot.

Because the camera of the robot is mounted on its head, the vision module requires some additional data concerning the head odometry. Furthermore, it needs data on the current camera settings (which can be altered through the camera interfaces). This data is provided in the CameraMatrix object (see GT schematics).

The most important feature of this object is the horizon, this is the line that describes the angle of the head relative to the ground. The following image describes the virtual projection of the camera matrix against the ground plane.



The two red circles are the start- and endpoints of the horizon. This line is used as a base for the direction of the scan lines. In ordinary image processing, an image is scanned from 0,0 to x,y, taking every pixel into account. To save valuable processing time, only every other pixel is scanned according to a line function (scan line). The direction of the scan lines are either parallel or perpendicular to the horizon. This is done to correct for the angle of the head. Each line is scanned pixel by pixel from top to bottom. During the scan each pixel is classified by colour. A characteristic series of colours or a pattern of colours is an indication of an object of interest, e. g., a sequence of some orange pixels is an indication of a ball; a sequence or an interrupted sequence of pink pixels followed by a green, sky-blue, yellow, or white pixel is an indication of a flag; an (interrupted) sequence of sky-blue or yellow pixels followed by a green pixel is an indication of a goal, a sequence of white to green or green to white is an indication of an edge between the field and the border or a field line, and a sequence of red or blue pixels is an indication of a player. All this scanning is done using a state machine; mostly counting the number of pixels of a certain colour class and the number of pixels since a certain colour class was detected last. That way, beginning and end of certain object types can still be determined, although some pixels of the wrong class are detected in between.

To classify a pixel by its colour, there's need for a determination of which colour is which. In other words, what range of RGB (or YUV) values determines as specific colour (for instance: orange). To do this, the Robot Control program provides the developer with a tool to set the colour ranges for each specific colour. Due to lighting changes, the colour table (the set of all colour settings) has to be changed accordingly.

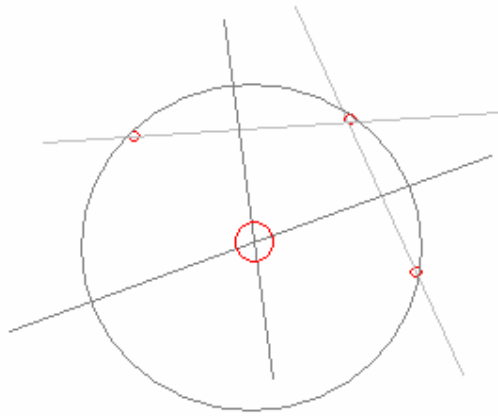
The detection of objects classes in the field are quite different, each of the following subchapters describes how this detection is done.

Ball detection

For balls, upper and lower points on their boundaries are detected during scanning. Points on the border of the image are ignored. During scanning, red pixels below a reasonable number of orange pixels are also treated as orange pixels, because shaded orange often appears as red. Although only a single ball exists in the game, the points are clustered before the actual ball position is detected, because some of them may be outliers on the tricot of a red robot. To remove outliers in vertical direction, upper points are ignored if they are below many other lower points in the same cluster, and lower points are ignored if they are above many other upper points in the same cluster.

The actual calculation of the position of the ball depends on the number of points detected:

- If at least three points have been detected, these can be used to calculate the centre of the ball by intersecting the middle perpendiculars. So, three points are selected to construct two middle perpendiculars.



Considering that the three small red circles are the found ballpoints, the dark grey lines are the middle perpendiculars and the large red circle the assumed centre of the ball. From there on, the distance to any ballpoint can be taken as the radius of the circle.

Points that are close to green are preferred, because it is assumed that they are actually located on the border of the ball. In contrast, points close to white can result from a high spot on the ball, but also from the real border of a ball in front of the border of the field. First, two points are selected that have the largest distance from each other. Then a third point is chosen that is furthest away from the two other points.

If the three points do not lie on a straight line, the centre of the ball in the image can be calculated, even if the ball is only partially visible. If the ball is not below the horizon or if the camera matrix is not valid because the robot is currently kicking, the distance to the ball is determined from its radius. Otherwise, the distance is determined from the intersection of the ray that starts in the camera and points to the centre of the ball with a plane that is parallel to the field, but on the height of the ball centre.

- If there is at least a single point on the border of the ball, it is assumed to either be the highest or the lowest point of the ball. The point is projected to the field plane and the distance to the ball is determined from this projection.
- If all orange points lie on the border of the image, it is assumed that the ball fills the whole image and the middle between all orange border points is assumed to be centre of the ball. The position on the field is again determined by intersecting the view ray with the field plane in the height of the ball centre.

Finally, the position of the ball in field coordinates is projected back into the image, and a disk around this position is sampled for orange pixels. If enough orange pixels are found, the ball is assumed to be valid.

Flag (field corner) detection

All indications for flags found during scanning the grid are clustered. In each cluster there can actually be indications for different flags, but only if one flag got more indications than the others, it is actually used. The centre of a cluster is used as a starting point for the *flag specialist*. It measures the height and the width of a flag. From the initialization pixel the image is scanned for the border of the flag to the top, right, down, and left where top/down means perpendicular to the horizon and left/right means parallel to the horizon. This leads to a first approximation of the size of the flag. Two more horizontal lines are scanned in the pink part and if the flag has a yellow or a sky-blue part, two more horizontal lines are also scanned there. The width of the green part of the pink/green flags is not used, because it is not always possible to distinguish it from the background. To determine the height of the flag, three additional vertical lines are scanned. The leftmost, rightmost, topmost, and lowest points found by these scans determine the size of the flag.

To find the border of a flag, the flag specialist searches the last pixel having one of the colours of the current flag. Smaller gaps with no colour are accepted. This requires the colour table to be very accurate for pink, yellow, and sky-blue.

Goal detection

A goal *specialist* measures the height and the width of a goal. The image is scanned for the borders of the goal from the left to the right and from the top bottom, where again top/down means perpendicular to the horizon and left/right parallel to the horizon. To find the border of the goal the specialist searches the last pixel having the colour of the goal.

Smaller gaps with unclassified colour are accepted. The maximal size in each direction determines the size of the goal.

Robot detection

To determine the indications for other robots, the scan lines are searched for the colours of the tricots of the robots. If a reasonably number of pixels with such a colour is found on a scan line, it is distinguished between two cases:

- If the number of pixels in tricot colour found on a scan line is above a certain threshold, it is assumed that the other robot is close. In that case, the upper border of its tricot (ignoring the head) is used to determine the distance to that robot. As with many other percepts, this is achieved by intersecting the view ray through this pixel with a plane that is parallel to the field, but on the “typical” height of a robot tricot. As the other robot is close, a misjudgement of the “typical” tricot height does not change the result of the distance calculation very much. As a result, the distance to close robots can be determined.
- If the number of pixels in tricot colour found is smaller, it is assumed that the other robot is further away. In that case, the scan lines are followed until the green of the field appears. Thus the *foot points* of the robot are detected. From these foot points, the distance to the robot can be determined by intersecting the view ray with the field plane. As not all foot points will actually be below the robot’s feet (some will be below the body), they are clustered and the smallest distance is used.

Debugging

One of the basic ideas of the German Team architecture is that multiple solutions exist for a single task, and that the developer can switch between them at runtime. In addition, it is also possible to include additional switches into the code that can also be triggered at runtime. The realization consist of two methods: *debug requests* and *solution requests*. The system manages two sets of information, the current state of all *debug keys*, and the currently active solutions.

A special infrastructure called *message queues* is employed to transmit requests to all processes on a robot to change this information at runtime, i. e. to activate and to deactivate debug keys and to switch between different solutions. The message queues are also used to transmit other kinds of data between the robot(s) and the debugging tool on the PC. For example, motion requests can directly be sent to the robot, images, text messages, and even drawings can be sent to the PC. This allows visualizing the state of a certain module, textually and even graphically.

IV. Concepts of edge detection

Introduction

This paper describes a number of concepts in edge detection. It consists of an explanation of what edge detectors are, what edge detectors are used for and the way to implement one. Furthermore, this study explains the different techniques which are used to compute edges, and how the advantages and disadvantages of each technique. Last, there is some sample pseudo code included to help future programmers get on their way.

Goals

The main goal of edge detection is to simplify an image without losing information. This is based on the assumption that the information of every image is stored in the colour intensity slopes. In other words, every shape in an image can be replaced by its boundary without losing precious information.

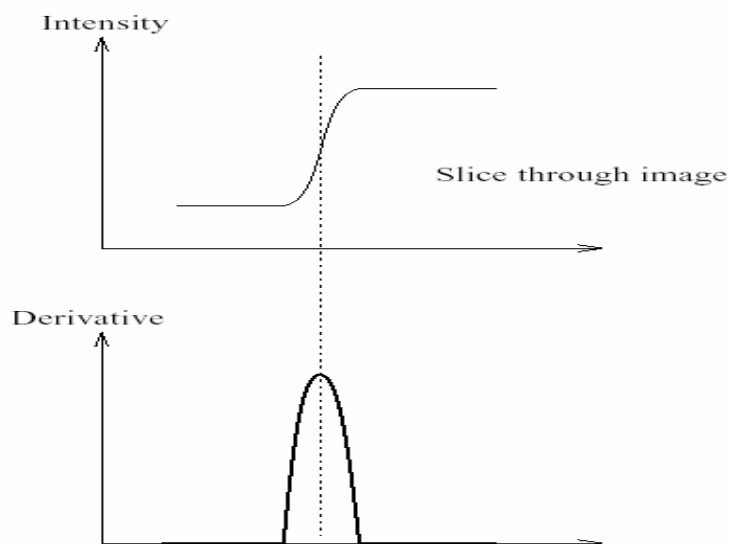
The reason to simplify the image is the fact that feature extraction (the common name for all methods which simplify images) is mostly used as a pre-processing step to something more intelligent, such as shape detection. In that case, the next processing steps do not have to cope with huge amounts of data (e.g. the entire image), but only with the pre selected, high energy, parts of the image, which of course saves a lot of processing time.

Methods & Techniques

Mathematical concept

Most edge detection methods work on the assumption that an edge occurs where there is discontinuity in the intensity function or a very steep intensity gradient in the image. Using this assumption, if we take the derivative of the intensity values across the image and find points where the derivative is a maximum, we will have marked our edges.

The following images illustrate this method:



The gradient of the image function I is given by the vector :

$$\nabla I = \left[\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y} \right].$$

The magnitude of this gradient is given by :

$$\sqrt{\left(\frac{\partial I}{\partial x}\right)^2 + \left(\frac{\partial I}{\partial y}\right)^2}$$

Although typically, an approximate magnitude is computed using:

$$|G| = |G_x| + |G_y|$$

Which is much faster to compute.

And the angle of the gradient:

$$\tan^{-1}\left(\frac{\partial I}{\partial y} / \frac{\partial I}{\partial x}\right).$$

To find the points where the first derivative has it maximum, we need to take the second derivative of the image function I , being:

$$\nabla^2 I = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}.$$

and calculate the zero crossings.

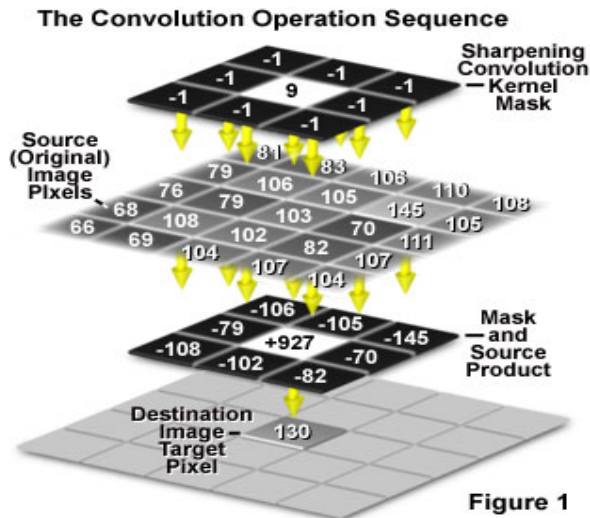
Convolution

Since an image doesn't consist of a 2 – dimensional intensity function, but of adjacent pixels, we need to find a way to get the gradient of every pixel in a different, programmable way.

There are several methods to detect edges in an image; however, all these methods are based on 1 single concept called convolution.

Convolution is a concept that uses map overlays to compute the spatial gradient for every pixel. A map overlay is a matrix (kernel) of x by y pixels. The matrix is multiplied by all the pixels it covers. The sum of all products is used as the convolved pixel value, representing the result of the chosen image process (for instance, sharpening, edge detection, smoothing). This way, all pixels are to be convolved by the kernel as to create a new image map.

The next image illustrates how this method is used with a sharpening kernel:



In edge detection there's need for horizontal and vertical edge detection since edges can be found in varying angles. For this we take two kernels (one vertical / one horizontal), after convolving the two kernels, the mean square of the convolved pixel values is as the magnitude of the convolution (see chapter : *Mathematical concept*). The angle of the edge is determined by taking the arctangent of the relation between the vertical convolution and the horizontal convolution (dy/dx).

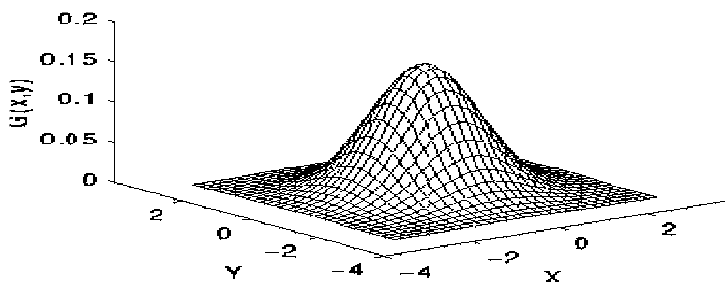
Pre-processing

Since most images consist of 3 layers (RGB , YUV..) of pixel intensities and usually contain large amounts of noise, there's need for a couple of pre-processing steps before any edge detection can be applied.

First, a one-layer image (greyscale) has to be created by taking the average value of all pixels in the three layers. Next, the image needs to be smoothed to filter the static out of the image. This can be done with a couple of convolution kernels, but the most commonly used kernels are the mean – and Gaussian kernel.

The mean filter is a 3x3 kernel with only 1/9 as values. Like the name says, it takes the sum of all adjacent pixel and itself and divides this by 9.

The Gaussian kernel is represented as follows:



The convolution mask can be made as big as the programmer want, however, the bigger the kernel, the less image boundary can be smoothed (the kernel has to do a complete overlay, so boundary pixels cannot be smoothed).

The concept is quite the same as the mean filter concept, the product of the kernel and the underlying pixels is taken. The sum of these products is divided by the weight of the complete kernel (e.g. the sum of all kernel values).

Both kernels are displayed below.

MEAN

$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$
$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$
$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$

GAUSSIAN

$\frac{1}{161}$	2	4	5	4	2
	4	9	12	9	4
	5	12	15	12	5
	4	9	12	9	4
	2	4	5	4	2

After these pre-processing steps, the edge detection kernel can be applied to the image. The next chapter describes different kernels to accomplish this goal.

Kernels & Methods

There are several operators to find the spatial intensity changes of an image. Some are designed for speed, some for integrity. This chapter describes the most commonly operators, being Roberts-Cross and Sobel. The last described method is the Canny edge detection method, which includes all pre and post processing steps.

Roberts-Cross

The Roberts-Cross operator uses two convolution masks, size 2x2 pixels, one for vertical and one for horizontal convolution.

+1	0
0	-1

G_x





0	+1
-1	0

G_y

The main reason for using the Roberts-Cross operator is that it is very quick to compute. Only four input pixels need to be examined to determine the value of each output pixel, and only subtractions and additions are used in the calculation. In addition there are no parameters to set. Its main disadvantages are that since it uses such a small mask, it is very sensitive to noise. It also produces very weak responses to genuine edges unless they are very sharp.

Sobel

The Sobel operator consists of a pair of 3x3 convolution masks as shown in the next picture. One mask is simply the other rotated by 90°. This is very similar to the Roberts-Cross operator.

    +1		
-2	0	+2
-1	0	+1

G_x

+1	+2	+1
0	0	0
-1	-2	-1

G_y

These masks are designed to respond maximally to edges running vertically and horizontally relative to the pixel grid, one mask for each of the two perpendicular orientations. The masks can be applied separately to the input image, to produce separate measurements of the gradient component in each orientation (call these G_x and G_y). These can then be combined

together to find the absolute magnitude of the gradient at each point and the orientation of that gradient.

The Sobel operator is slower to compute than the Roberts Cross operator, but its larger convolution mask smoothes the input image to a greater extent and so makes the operator less sensitive to noise. The operator also generally produces considerably higher output values for similar edges compared with the Roberts Cross.

Canny

The Canny operator was designed to be an optimal edge detector (according to particular criteria, there are other detectors around that also claim to be optimal with respect to slightly different criteria). It takes as input a grey scale image, and produces as output an image showing the positions of tracked intensity discontinuities.

The Canny operator works in a multi-stage process. First of all the image is smoothed by Gaussian convolution. Then a simple 2-D first derivative operator (somewhat like the Roberts-Cross) is applied to the smoothed image to highlight regions of the image with high first spatial derivatives. Edges give rise to ridges in the gradient magnitude image. The algorithm then tracks along the top of these ridges and sets to zero all pixels that are not actually on the ridge top so as to give a thin line in the output, a process known as *non-maximal suppression*. The tracking process exhibits hysteresis controlled by two thresholds: $T1$ and $T2$ with $T1 > T2$. Tracking can only begin at a point on a ridge higher than $T1$. Tracking then continues in both directions out from that point until the height of the ridge falls below $T2$. This hysteresis helps to ensure that noisy edges are not broken up into multiple edge fragments.

The effect of the Canny operator is determined by three parameters; the width of the Gaussian mask used in the smoothing phase, and the upper and lower thresholds used by the tracker. Increasing the width of the Gaussian mask reduces the detector's sensitivity to noise, at the expense of losing some of the finer detail in the image. The localization error in the detected edges also increases slightly as the Gaussian width is increased.

Usually, the upper tracking threshold can be set quite high, and the lower threshold quite low for good results. Setting the lower threshold too high will cause noisy edges to break up. Setting the upper threshold too low increases the number of spurious and undesirable edge fragments appearing in the output.

One problem with the basic Canny operator is to do with Y-junctions *i.e.* places where three ridges meet in the gradient magnitude image. Such junctions can occur where an edge is partially occluded by another object. The tracker will treat two of the ridges as a single line segment, and the third one as a line that approaches, but doesn't quite connect to, that line segment.

Conclusion

If you are looking for a way to compress images in order to make next processing steps easier and faster to compute, edge detection is a great method to accomplish this. As described there are several ways to compute discontinuities in the image intensity curve, so you can choose one that fits your needs. Of course, programmers can always take existing methods and tweak them to meet their goals.