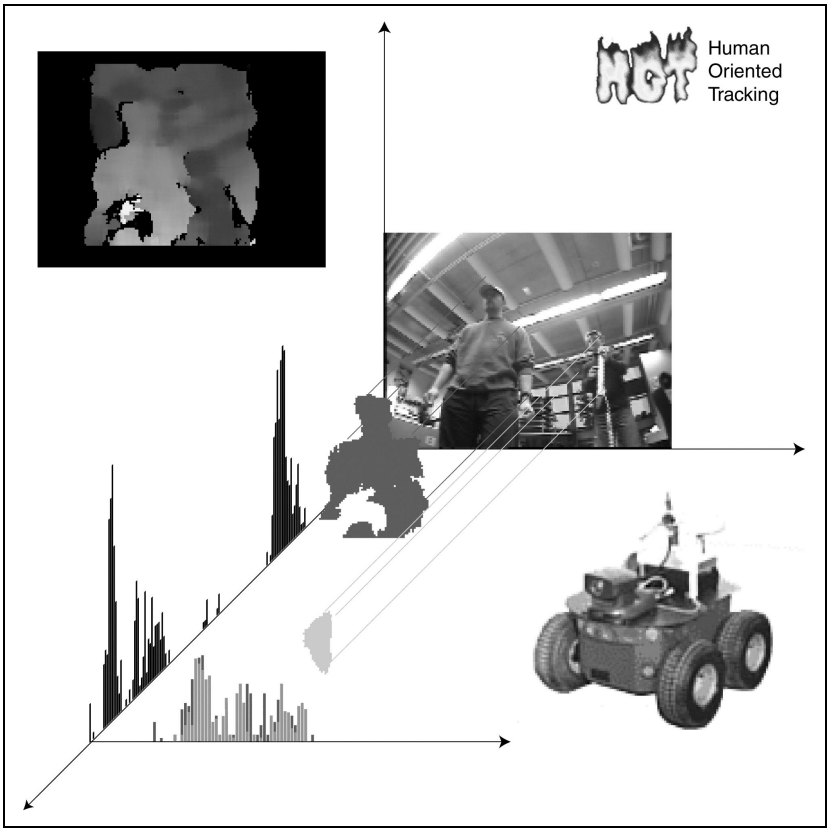
EIDGENÖSSISCHE TECHNISCHE HOCHSCHULE LAUSANNE
POLITECNICO FEDERALE LOSANNA
SWISS FEDERALE INSTITUTTE OF TECHNOLOGY

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Christian Wengert, Björn Poëll
{ christian.wengert | bjoern.poell } @epfl.ch
Assistants: Sébastien Grange, Terrence Fong
Professor: Roland Siegwart

Human Oriented Tracking and Mobile Robot Gesture Driving



Semester Project Winter 2001/2002

Virtual Reality and Active Interfaces Group

Contents

List of figures

I. Introduction

I.1. Goal of the project

Our goal is to develop an active interface to interact with a mobile robot using gestures. Gestures are a natural way of communication between human beings without the need to know the complexity of the machine. To achieve this goal, the robot must detect people in front of it and should identify and recognize their gestures. The main task is to implement a real-time tracking system on a low-processing power, mobile platform. Several applications can benefit from this technology.
As many conditions are not invariant, this is a very challenging task. We have to deal with varying lighting and background conditions. In a first step, the robot has to be aware of the presence of a potential operator. This is done by processing color, motion and depth information from a camera pair mounted on the mobile robot. Even though processing power has evolved rapidly over the last years, a “slow” processor is used and therefore efficient algorithms have to be designed. These algorithms could also be used for portable devices such as PDAs, which begin to spread very fast. Many applications can be thought of for these devices, especially vision systems, which may have a promising future in teleoperation or visualization of geographical or other 3D data. The PDAs still have and will have in the next few years a very limited processing power because of their size and minimal energy consumption requirement.
Robots also often require small, cheap and low-power controllers in order to make them small (sewer inspections) and lightweight (humanoid robot, mars rovers).
Another interesting application lies in augmented reality or wearable computing where an extra set of eyes may provide additional information to a human. Here the electronic device must be as small and as light as possible too.

I.2. People tracking

People tracking is done only by using the depth information from a stereovision camera pair for two reasons:

- Color invariance (changing background and illumination does not affect stereo-vision).
- Efficiency (the used stereovision algorithm is very fast)

This can be advantageous when no color information in the scene can be exploited (many person tracking projects use skin color filtering in order to find humans, but in space or in a nuclear reactor where humans wear protective clothes, no skin color will be visible). In addition, if the robot tries to follow a person, no skin color might be visible. To make the system more robust, color-based identification is used to ensure that only one person is tracked at a time.

I.2.1. Problems

- Our robot is equipped with limited processing power. Therefore, low-resolution images are used and timesaving algorithms have to be designed.

- The cameras are mounted on a mobile robot, so illumination can change dramatically. In addition, background subtraction is not possible, because of the moving cameras.
- The robot must be able to track the person even if there is more than one person in sight or if the person is partially occluded.

I.2.2. Hypotheses

- The distance between the robot and its operator is limited to a certain range due to stereovision geometry.
- Skin color filtering cannot be used while following a person, because skin color may not be visible.
- Only a very simple robot controller has been implemented (no autonomous navigation). We focused our effort on the vision system.

I.3. Gesture recognition

In order to demonstrate how gestures can be used as a control modality, we have defined a discrete set of postures. These are based on a simple geometrical model, including only head and hands in order to have efficient recognition. Sensor fusion of depth and color information is used in order to extract the operator's head and hands in a robust manner.

I.3.1. Problems

- Defining a simple set of postures. Only the head and the hands are used for that.
- For simplicity reasons, static gestures are implemented. At a later stage, the speed of the moving hands can be used as further information in order to control the robot's speed as well (dynamic gestures).

I.3.2. Hypotheses

- Only a limited set of commands is implemented in order to control the robot (stop, follow me, go forward, go backward).
- These commands let the robot move only on a straight line (otherwise it looses sight of its operator).
- The user must stand in front of the robot.
- The robot only receives commands from people that stand still.
- The person who wants to command the robot must be the closest one in the scene.
- The system must be person independent. Everybody, experts and novices alike, have to be able to control the robot by simple gestures without training (whether of the user or of the robot)
- The robot must not perform an unwanted action that could endanger itself. Therefore, gesture recognition must be very conservative. An ignored command is safer than a false interpretation.

I.4. People Identification

To track a person, we use the geometrical properties of a human more than colors. This works fine for finding persons, but it is hard to differentiate two objects that are identified as human. In this case, we use the color scheme (a histogram) of a human

to discriminate. It is important that, in the case where more than one person are seen by the robot, the program will be able to find the right person to follow and doesn't start to track somebody else. To create a color scheme we use either the whole human, separated as much as possible from the background (called the mask of the human), or just a portion, such as a rectangle.

I.4.1. Problems

- As the robot is quite limited in its capacities, the routine to compare people has to be simple and fast
- Color intensity changes very fast and is not a very reliable value for color detection
- Clothing may be similar so it is not sufficient to uniquely check the color of the pullover or the trousers. The comparison must base on the whole color scheme of the human in question including skin and hair.
- The information received from the cameras is limited, i.e. a resolution of 160x120 pixels. If the human in question moves away from the robot, he becomes small very quickly. This means that we have to compare objects that have very different sizes i.e. some color scheme may have more than 5 times more pixels than an other to compare to.
- For the same reason (bad resolution) we may not use other typical attributes of a human like it's height.
- When comparing two people, it is not given that the two color schemes compared to each other represent the humans. Is the entire human represented? Does the scheme include parts of the background?

I.4.2. Hypotheses

- The color information and the mask received include a major part of the human although it is not necessary to have a perfect mask
- The lighting conditions do not change too dramatically.
- People do not wear too similar clothes (i.e. no twin look).

I.5. Robot Control

The robot is running on a Pentium 233Mhz processor, with Windows 98. It has ultra sonic range sensors that can be used for obstacle avoidance. To control the robot, we make use of the ARIA 0.7 C++ library [1]. It's quite a complete library that let's you control the robot in depth and implement very specific behavior. But one is not only well in control of the robots behavior, but also of exception handling, the way the connection has to be made and the way the robot should receive commands.

I.5.1. Hypotheses

- We assume that we don't have many obstacles, as our goal is to follow a human and not to avoid obstacles.
- We run the robot on a flat ground so that the camera always receives more or less the same portion of the human to follow. The tracking routines wouldn't work well if the robot's camera experienced vertical movement.

II. Related research

II.1. Human tracking

Much research has already been performed in the field of human tracking, using many different techniques. Most of them rely on static cameras permitting background subtraction and / or on skin color filtering. Konolige and Beymer [2] only used depth information from a stereo camera pair. They performed template matching in order to extract people from a disparity map. Templates describing the form of a standing person are matched in the disparity map and positive results are extracted.

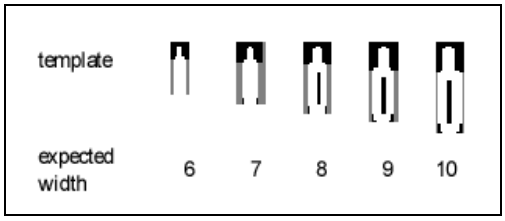


Figure 1: Templates used for person detection. The larger templates have zeros in the center since stereo may fail in the center region if the person is wearing clothing with little texture.

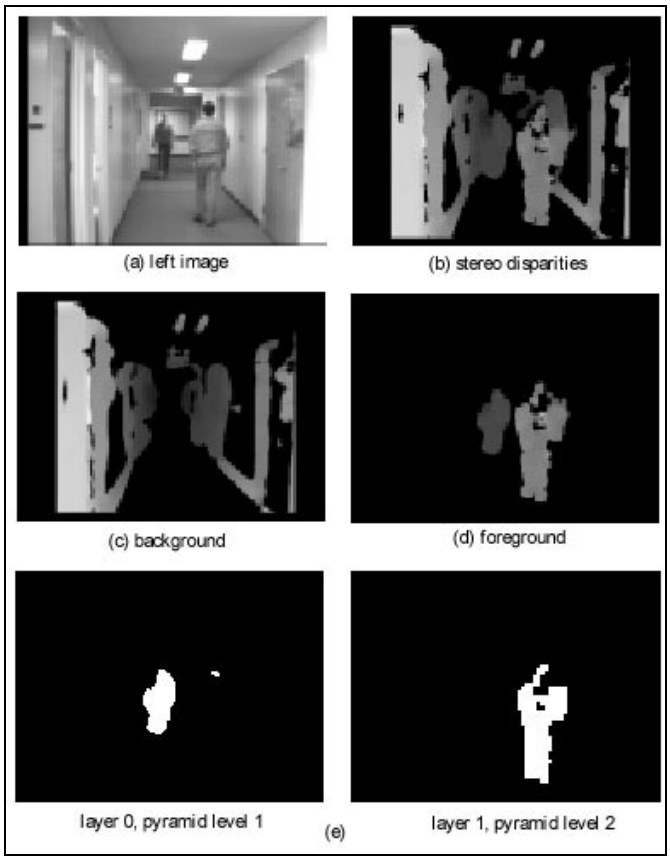


Figure 2: Results from template matching

Advantages of this method are that it is adaptable to any object where the shape is known a priori and that no color filtering is used. It can handle partial occlusions of objects. It also segments the space in depth, which is very useful information for

tracking. The real world size and speed may be used to determine whether the object might be a human or not. However, it is very demanding in terms of processor power. In addition, they use static cameras and background subtraction in order to extract interesting objects, which cannot be used on a mobile robot. The background subtraction could be omitted, but would demand even more processing power and would be a noisier process. Nevertheless, even with background subtraction, this process is too slow for our application.

Rawasek Tanawongsu, Alexander Stoytchev, Irfan Essa [3] use the combination of color, motion and depth information in order to extract people. From a set of pictures, they derived a skin color locus in the normalized red / green plane:

(1) $$r = \frac{R}{R+G+B} \text{ and } g = \frac{G}{R+G+B}$$

Images are filtered with the knowledge of the skin color locus in order to detect human faces and hands. The color filtering method is a very noisy process because wood and leather have similar colors in the normalized red / green plane as human skin. However, the filter works for every skin color from people from different continents. Also, in order to find the skin locus (which differs for every camera, lens and lighting configuration), it has to be found empirically. Because the skin locus changes if illumination changes, an adaptive skin locus should be implemented.
Gi-jeong Jang and In-So Kweon [4] propose such an adaptive skin locus algorithm. Others use statistical models in order to find the best skin locus. Our approach is to define quite a large skin locus and to use depth information to filter the noise that is generated by this procedure.

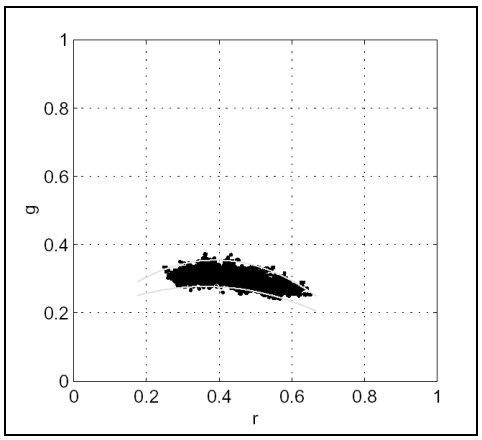


Figure 3: Example of a skin locus in the normalized red / green plane: Nogatech camera skin locus [5]

Another possibility that is a very easy and simple to implement are the use of markers (colored gloves or similar). However, wearing these markers is considered too inconvenient. On the other hand, this method can be very robust and fast tracking can be achieved (see Aibo [6] going after its pink ball).

Detecting people with sound is not always possible, especially in crowded places, in space or underwater (everywhere humans must wear heavy protective clothes with helmets/masks). However, under some circumstances, it can help to find the user

faster, because his position relative to the robot can be estimated quite accurately. Using only sound is certainly not sufficient, but it may be a very useful additional sensor [7].

II.2. Gesture Recognition

Jochen Triesch and Christoph von der Malsburg [8] implemented a single hand tracking based on a static camera, in order to command a robot's hand (grasp an object, put the object somewhere). This is done for a robot that is fixed so they can use background subtraction for simpler object detection. They use skin color detection and motion cues to extract the human's hand and to determine the meaning of the hand posture. In order to detect the fingers, which are very important to determine the posture's meaning, they have to use a better resolution than we have. It is also person independent, requiring no training and using no markers, which is very important in order to have a natural way of communication between the robot and (any) human.

Stefan Waldherr, Sebastian Thrun and Roseli Remero [9] implemented an arm gesture recognition system on a mobile robot using neural networks. While following a person, they adapt their skin locus to the changing illumination. This requires continuous tracking of the head and hands and therefore treating the whole image at any time, which takes much more time than processing only a small band of the image, as we will do.
Sébastien Grange [10] uses sensor fusion of depth and color information in order to extract the user's hands and head. The sensor fusion permits to filter to noise of the color filtering process. Only a simple skin locus has been implemented (a rectangle in the normalized red / green plane). The model of the human is very simple (only head and hands, ignoring the shoulders and the chest). For our application where only a limited set of possible postures is needed we will implement the same model.

Yuanxin Zhu [11] tracks only one hand with motion and color cues. He implemented motion templates in order to extract dynamic gestures. However, this system has to be trained for every gesture. Higher resolution images than in our system has been used as well in order to detect enough details (fingers).

David Kortenkamp, Eric Huber, R. Peter Bonass [12] propose the proximity space method: They implemented a model of a human defining shoulder, arm and head of one side only (due to limited processing power) where every part of the modeled body is represented by a proximity space and the links between them by stiff springs. The joints are similar to the human joints in regard to the limitations, permitting the model to ignore impossible configurations and augment robustness. In our model, impossible configurations are also ignored, but only regarding the angles between head and hands.

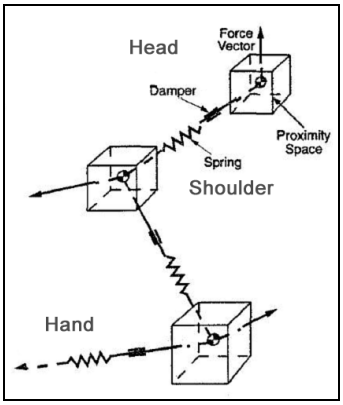


Figure 4: Proximity spaces with links representing head, shoulder, arm

Guangyou Xu, Yuanxin Zhu, Xueyin Lin, Haibing Ren, Xiaoping Zhang [13] use only monocular vision and color detection. Because the lighting conditions and background changes too much on a mobile robot, the robustness of this system is not sufficient for our application. Therefore, stereovision must be added.

II.3. People identification

II.3.1. Color Histograms

It is partly believed that color and surface characteristics of an object play only secondary roles in object identification, because geometric algorithms (i.e. stereo vision, motion) do give more information about an object, than its color.
In an article of Michael J. Swain and Dana H. Ballard [14], the role of color in vision is being discussed and they show that colors can be a robust criteria to detect and identify objects.
For that purpose, they use multidimensional color histograms. A histogram is obtained by discretizing the image colors and counting the number of times each discrete color occurs in the image array. To define a color, three values of “red”, ”green” and ”blue” are used. The color of each pixel is defined with these three values.
Histograms are invariant to translation and rotation of the object and they vary slowly with the change of angle of view, change in scale and occlusion of the object.
Point by point matching of histograms may be one approach to compare histograms but does not work very well, because of shading and camera noise.
To make a histogram insensitive to light intensity changes it is common to normalize it. To do a red-green normalization, each red and green value of a pixel is divided by the sum of the red, green and blue value, as shown by the formula (1). The 3-dimensional histogram becomes 2-dimensional.
To be able to judge the similarity between histograms Swain and Ballard introduce the method Histogram intersection, which tells how many of the pixels of one picture are found in the one to compare to.

II.3.2. Histogram Intersection

As a histogram can be seen as a function, the histogram intersection can be interpreted as the minima of two histogram functions. For a given pair of histograms A and B, each containing n values, the intersection of the histograms is defined as follows:

(2) $$\sum_{j=1}^{n} \min(A_j, B_j)$$

The result of the intersection of two histograms is the number of pixels that have the same color in both histograms. To obtain a match value between zero and one the intersection is normalized by the number of pixels in one of the two histograms (the one we would like to refer to, also called the model). The match value becomes then:

(3) $$H(A,B) = \frac{\sum_{j=1}^{n} \min(A_j, B_j)}{\sum_{j=1}^{n} A_j}$$

Distracting pixels in the background do not reduce the normalized histogram intersection match value. It is only increased by a pixel in the background if

- The pixel has the same color as one of the colors in the model, and
- The number of pixels of that color in the histogram to compare (also called object) is less than the number of pixels of that color in the model.

III. System overview

The following figure gives an overview of the system and its dataflow characteristics showing some performances of our system. The following paragraphs introduce the single sub-systems.

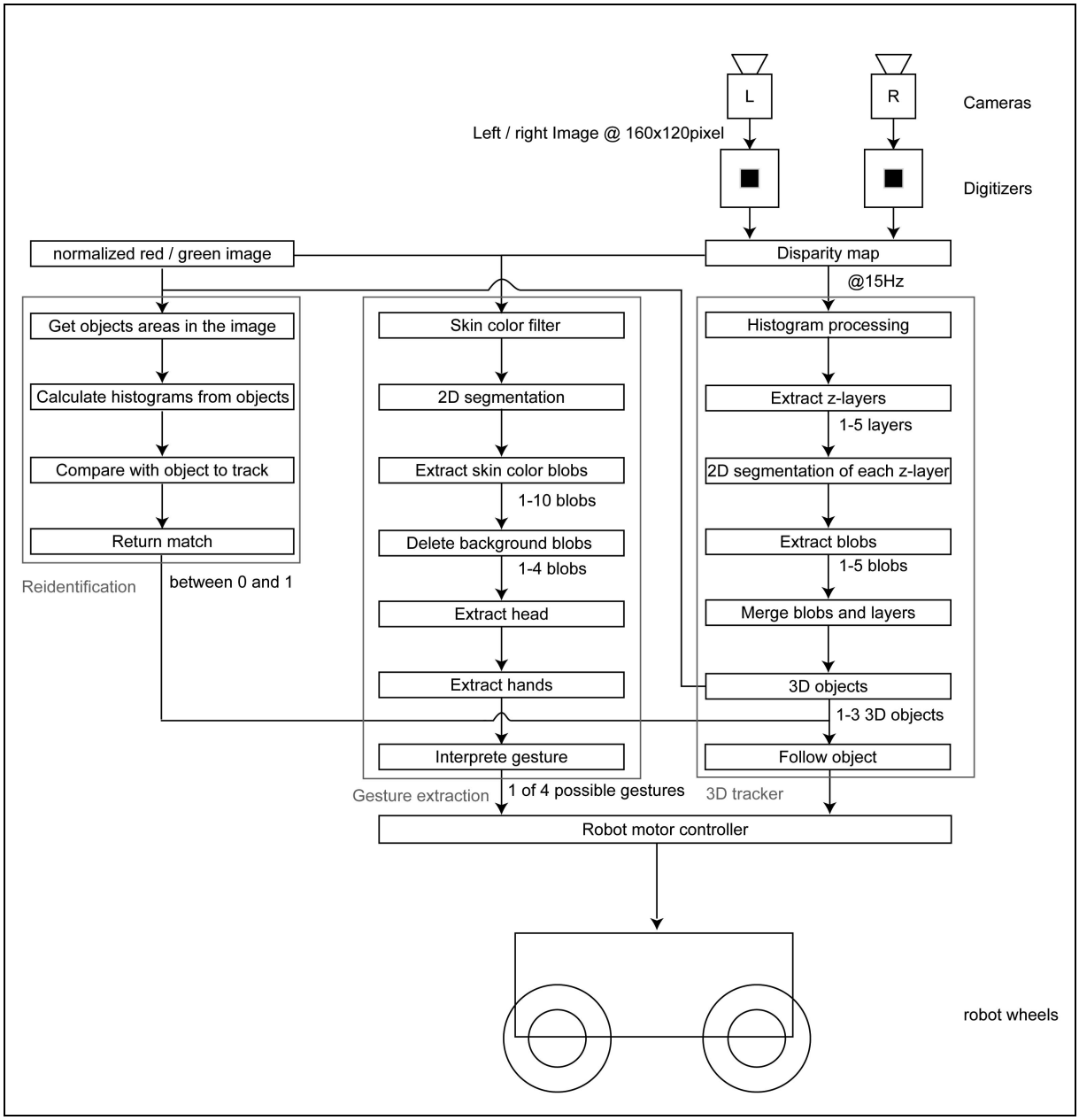


Figure 5: Dataflow chart

III.1. Hardware

The hardware of our system is presented in the following subsections:

III.1.1. Robot

Our system is implemented on an all-terrain mobile robot from Active Media [15] (Pioneer 2-AT).

- Skid-steer platform.
- Autonomy up to 4 to 8 hours on three fully charged batteries.
- Embedded PC104+ Pentium MMX @233MHz (see appendix IX.4.3).
- 128 MB RAM.

- The operating system is Windows98 (which can easily be programmed using Microsoft Visual C++).
- The robot is equipped with 16 sonar sensors for obstacle avoidance (8 in front, 8 behind).

Figure 6: Pioneer II from Active Media

III.1.2. Imaging system

Two cameras from PacificCam (see appendix IX.4.1) and two frame-grabbers PXC200 (see appendix IX.4.2) from ImageNation are mounted on the mobile robot. For efficiency the image resolution is 160 by 120 pixels. Fifteen frames per second range image acquisition and computation can be achieved with the SRI small vision system [16] with the given image resolution.

III.1.3. Geometry

We need a wide field of view so that the robot sees humans on a wide range (as well in x and y as in z). In addition, the robot, which is much smaller than a human, must be able to see the head and raised hands of the user for gesture extraction even if the user is very close to the robot. In order to save processor time, we only process a part of the image. However, we must be sure that the human is always inside this small area in order to track continuously. Using wide-angle lenses and tilted cameras fulfills all these requirements. See Figure 7 for details. The disadvantage of this system is that distant objects tend to be warped, which could be solved by a software dewarp function. As the image resolution is small and we mostly track close objects, and the shape of hands or the head do not change dramatically, this need not be done. Also, when using the disparity map of the whole image, objects in the upper part of the image seems to be further away than they actually are. Therefore a software correction is performed (see Figure 8). Note that the hands, which are the closest objects to the robot, have the brightest values after processing.

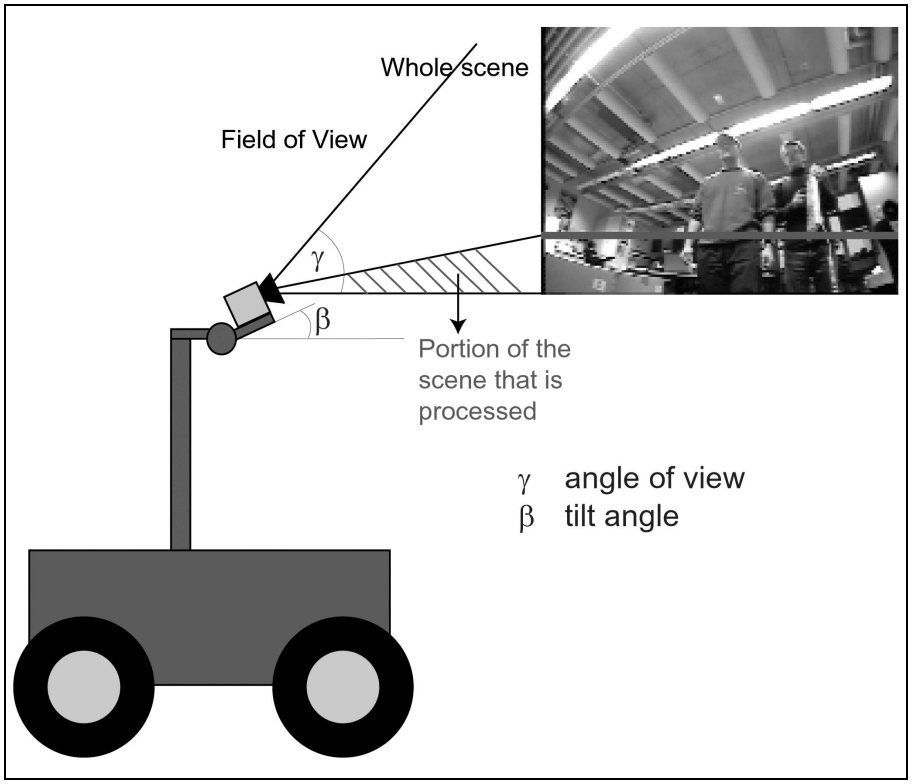


Figure 7: Geometry of the vision system on the mobile robot

Figure 8: Original image (left), disparity map unprocessed (middle) and processed (right) to correct disparity

III.1.4. Overview of techniques

As mentioned above, we only use stereo information in order to detect objects (in our case humans) in front of the robot. But this information is treated in several ways in order to obtain a robust object tracker from the different results.

Stereo delivers many advantages:

- Even partially occluded objects can be detected, which is major problem for other techniques.
- Gives information about the real size of an object, the real-world velocity and the real distance between object and robot.
- The presence of distracting shadows, lighting changes, and camera dynamics has little effects [17].

However, there are some limitations as well:

- Any object is detected. Tracking is not human-specific.
- If two objects are very close to each other, they might not appear as separate objects.

- Depth measurement is limited.
- Depth measurement is not linear. It follows an 1/x relationship (which is implemented in all functions and calculations in our system).
- Stereovision is a noisy process. Especially if there are not enough textures, some parts of the range image cannot be calculated. However an interpolation technique can be used in order to estimate if two objects belong together (if there is a non calculated region between two regions with same disparity, they will be connected).

Histogram processing of the stereo image also delivers useful information:

- Information about the zones of interest in depth (z-slices or layers) by searching peaks in the histogram. Every peak in the disparity histogram represents one or several objects.
- Background/foreground detection: As the background is also processed from the stereovision, we may say that the peak with the lowest disparity value represents the background. Then all values below that threshold are considered as background and are ignored.

III.1.5. Stereo geometry

Some geometrical information is needed in order to work on the disparity image. The disparity is inversely related to depth, and we have to be aware of confusing pixel values (width and height) and real world lengths. In the entire program, real world lengths (centimeters) have been implemented in order to estimate the object's width, height and especially the distance between objects in the image.

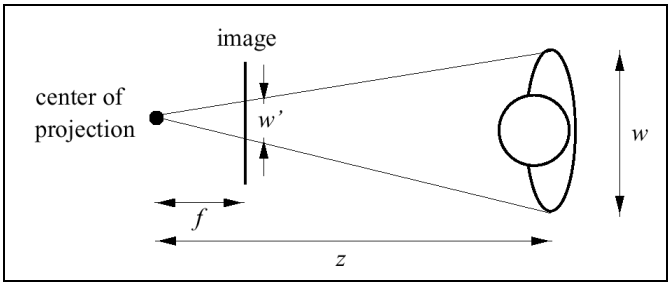


Figure 9: Person width w' in the image plane is proportional to disparity

Mathematically this is expressed as follows:

(4) $\frac{z}{w} = \frac{f}{w'}$ and $d = \frac{bf}{z}$ and finally $w = K\frac{w'}{d}$

With d the disparity value of object, b the stereo head baseline and f the camera focal length, and K a constant that has to be determined for every vision system through calibration.

III.2. People tracking

III.2.1. Histogram processing

A histogram is processed from the small portion of the stereo image (Figure 10).

Peak extraction

From this histogram, all peaks are extracted. Any object is visible as a peak in this histogram (very close objects may be represented by one peak only). A peak is defined around its maximum value, its area, its width and its distances between the maximum and the start / end point. If the area of a peak is smaller than a predefined minimum area, this peak is considered to be noise and will be ignored. Also, the peak that is the furthest away (having the lowest disparity values) is considered as background and will not be processed. Then we segment the 3D space into layers (z-slices) containing all disparity values defined by a peak. Each layer is then 2D segmented.

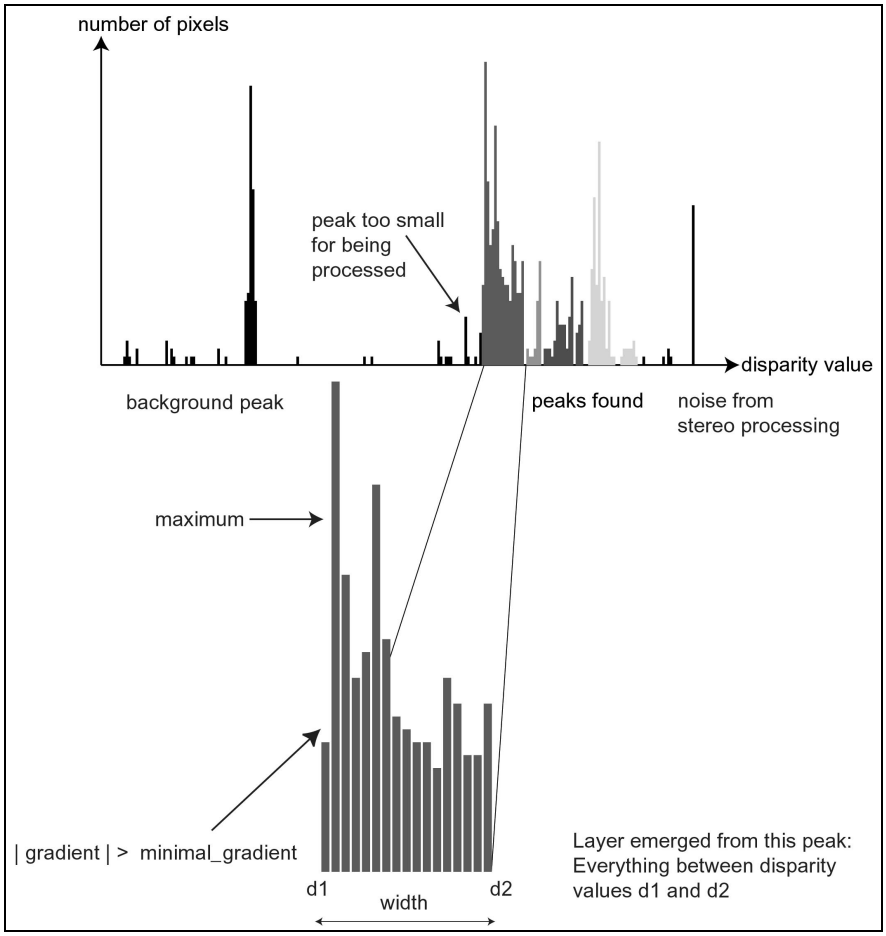


Figure 10: Histogram peak definition and peak extraction

(5) $g = \frac{f(x_1) - f(x_2)}{h}$ Gradient calculation [18]

Peak connection

Because of the noise in the image, too many peaks are extracted. Therefore, another test is performed in order to try to connect two close peaks. The distance, which defines if two peaks can be connected, is inversely proportional to the disparity value.

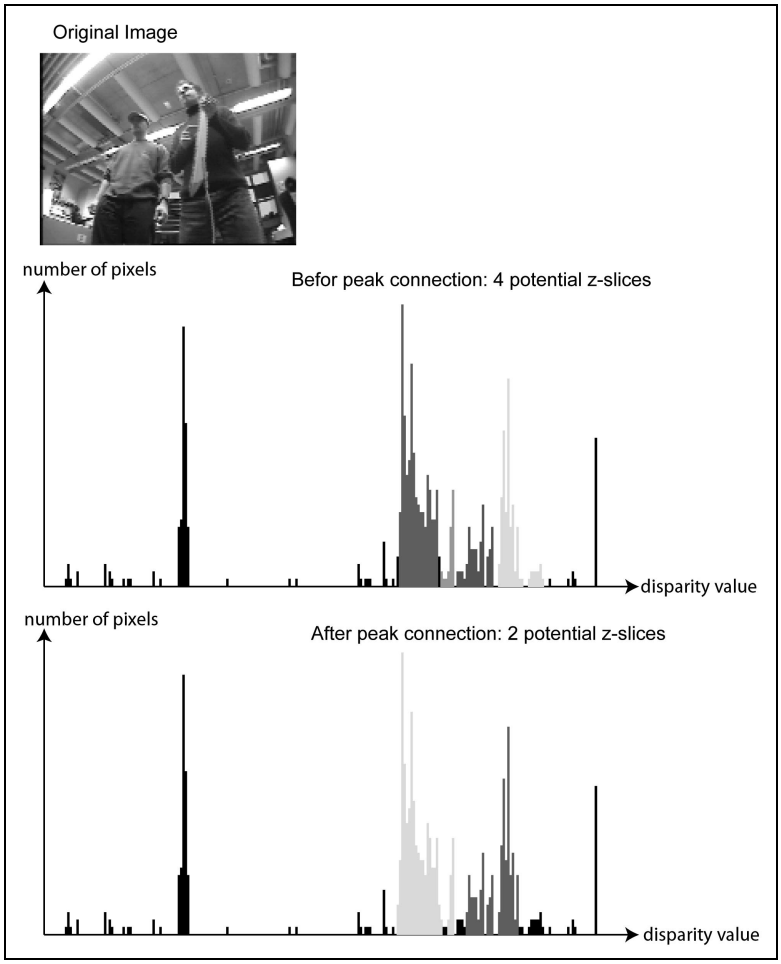


Figure 11: Histogram peak extraction and connection

III.2.2. 2D Segmentation

Once an interesting z-slice has been extracted, a binary image is produced from this slice. This image is 2D segmented using the well-known recursive Connected Compound Labeling algorithm. A list of objects (blobs) is then created containing the information about the object's distribution in the x-y plane, the center of gravity and the position in the image plane. No classic image filters are used in this process, but the blobs are filtered by area (removes single pixel noise) and blobs that are very close to each other will be connected to one blob. This had to be added because the blobs can be un-connected due to the stereo effect of missing textures of a body (see figure below).

Figure 12: Original image (left) and disparity map of processed image portion (right), which shows unconnected blobs

III.2.3. 3D Segmentation and 3D Object Extraction

By putting the histogram processing and 2D segmentation together and performing it for all peaks, we obtain layers of the real world having each at least one object inside. From this information, we try to extract all 3D objects from the scene that are potentially human beings.

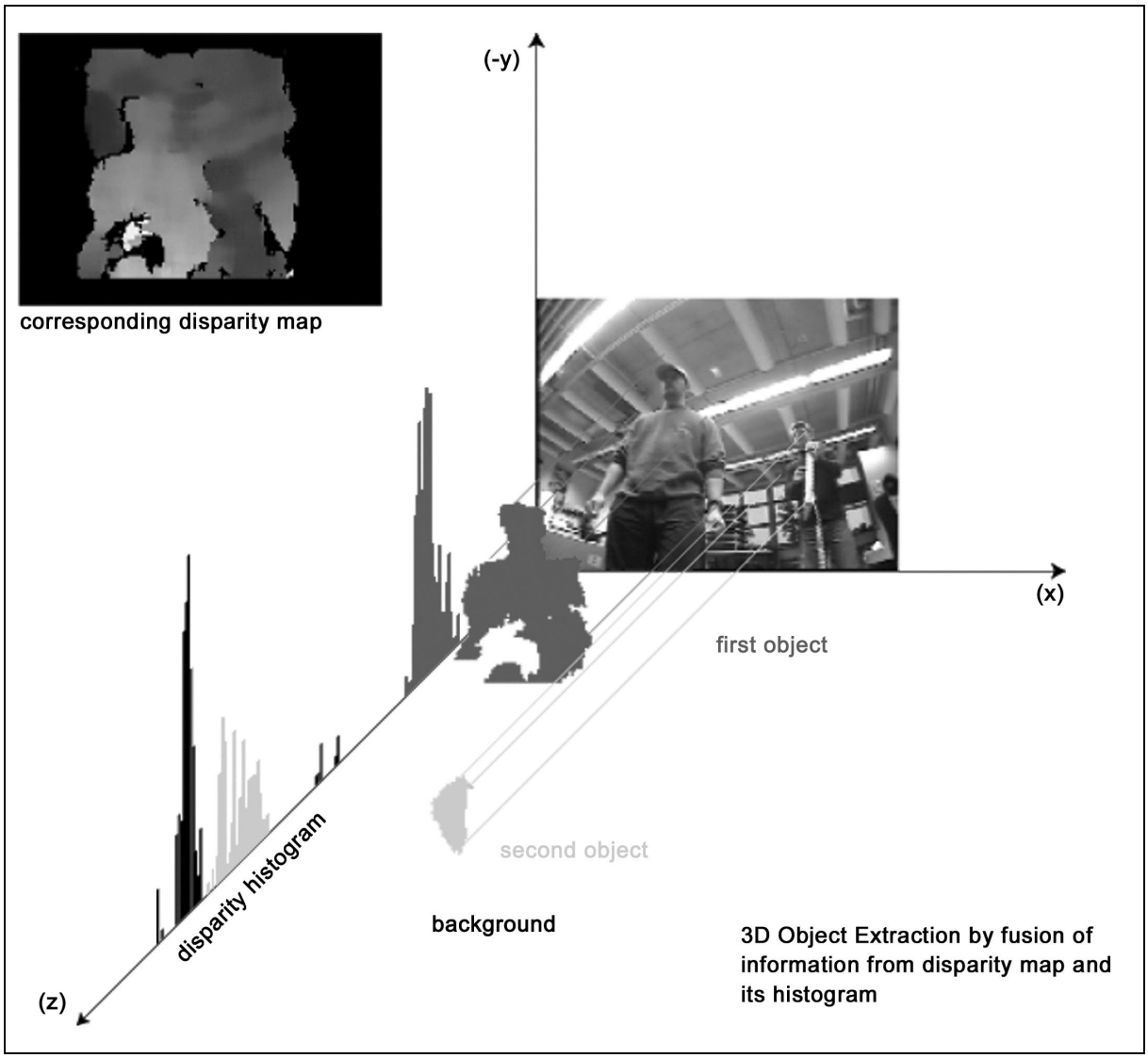


Figure 13: Schema of 3D Object Extraction

For each z-slice or layer, all 3D objects are extracted containing each its distribution of pixel in the x-y plane (blob), the disparity information, the real world and image coordinates and an identification number (ID). Even though much filtering and connection has already been performed on the blobs and on the peaks, errors may still be present. Therefore, the 3D objects are compared to each other and, in case of an overlap (in x, y and z), they are merged.

Advantages of our system are:

- No image processing filters (binary, morphology, matrix) are used, only pixel operators, which are very fast. Filtering is done within the peak-, blob- and object-space and is therefore very fast. Noise can be very easily eliminated in this high-level approach.

- Any object can be tracked. No limitation to humans. Filtering of the objects can be done with respect to the object's real world size.

III.2.4. Continuous tracking

Once an object to follow has been detected, it is better practice to use the information from the previous step in order to try to forecast the objects new position calculating its speed and its direction. This is done using Kalman filtering [19] for least square estimate of real-world velocity and direction of travel of the object. This helps to minimize the tracking area in order to accelerate the tracking sequence. In order to minimize errors, we need a smooth robot trajectory and therefore an improved robot control algorithm.

III.2.5. Color based tracking

Another variant for the tracking is using the re-identifying algorithm in order to keep track on the same object all the time. This is especially useful when the robot is not moving smoothly but follows a person in a zigzag way. Then the forecast algorithm is not robust enough and prevents the system from erroneous tracking. Also, as only a small part of the image is processed to compare the object's color, this is not significant in terms of processing power.

III.3. Gesture recognition

If the operator stops for a while, the robot turns into the gesture recognition mode. Then the whole image is processed (stereo as well as skin color) in order to extract head and hands. This is with respect to the band processing used for tracking a much slower process, but gestures do not need to be updated as fast as tracking information and the resulting lower refresh rate still permits to command the robot at an acceptable speed.
For representing a human being, a very simple model has been developed. A small set of postures has been defined for the basic control of a robot:

- Stop
- Move forward/backward
- Follow me

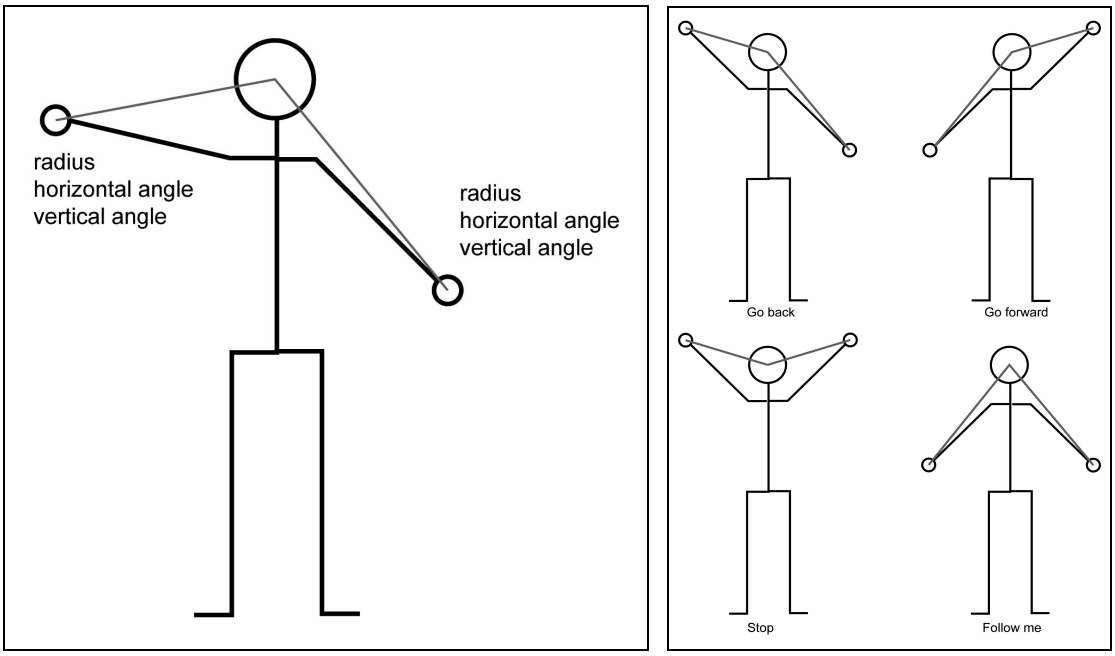


Figure 14: Geometrical model of a human being (left) and possible postures (right)

III.4. People identification

As introduced in the section II.3.2, histogram intersection is used to identify the right person in case there is more than one. To compare two people, normalized colors are used to minimize the influence of changing intensity. A rectangle that covers part of the trousers and part of the pullover is used to build the histogram. To find the degree of coincidence of the histograms in question, we first have to assure that they have a comparable size. Therefore, both histograms are normalized, which means that all values of the histogram are multiplied by a certain factor so that the peaks of both histograms have the same value (i.e. 100).
In Figure 15, we see two different views of a person:

Figure 15: Is this the same person?

Let's say the person is tracked and we would like to compare them to each other to know if it's the same person. In Figure 15, the green rectangle shows the parts that will be used for the histograms:

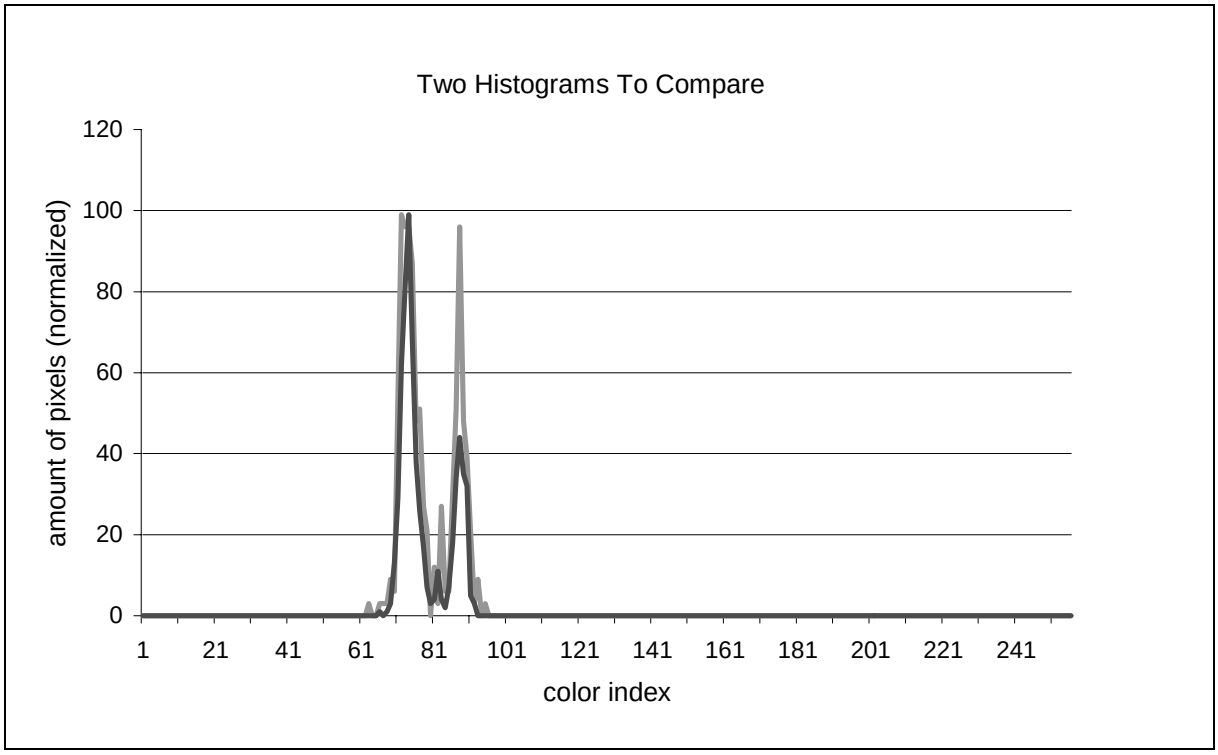


Figure 16: Two normalized histograms of the same person, different views

The green line represents the histogram of the colors in the rectangle of the left image and the red line stands for the right image. Although the right rectangle is obviously greater than the left one, its histogram (red) seems to include fewer values than the green histogram. This is because the histograms had been normalized. Then the histogram intersection is applied:

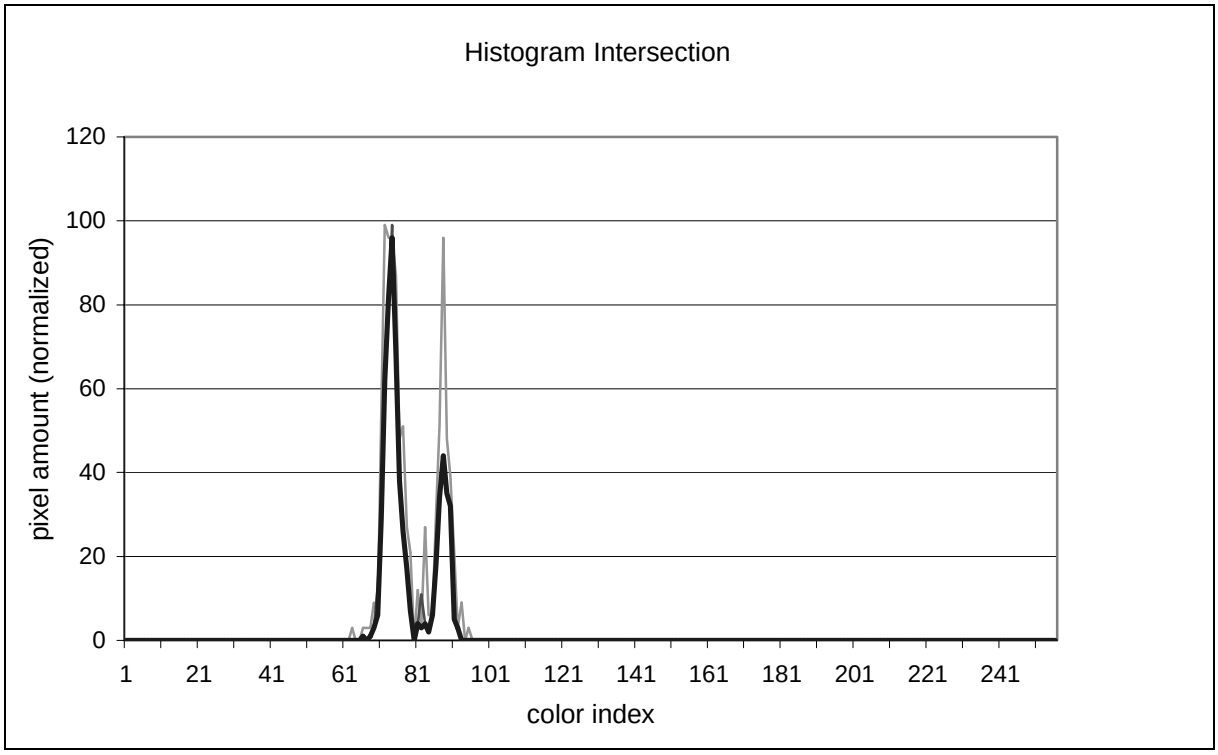


Figure 17: The blue line is the intersection of the two histograms

For each color, the minimum of the two histograms to compare is taken and compared to one of the histograms. For each color, we receive a value between zero and one, where zero means no correspondence and one means complete correspondence. Then we calculate the total correspondence of the whole histogram by averaging all values. The total correspondence (or the match) for the example shown above is 0.95.

This method to compare histograms has some advantages:

- It happens that a histogram to represent an object has peaks that are not typical for its surface, but they are temporarily there because of light reflections, camera or digitizer noise, etc. This (red circle in Figure 18) noise do not have a significant influence on the result of the comparison.

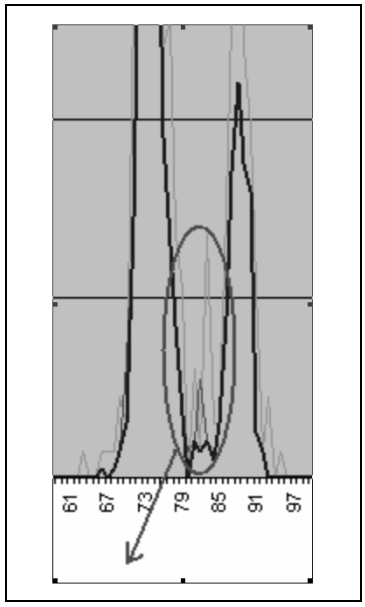


Figure 18: noise does not have much influence on the result

- The quality of the result does not depend on the color or the structure of the clothes the human wears because the whole color spectrum and especially the histograms curve is taken in account. The form of the histogram curve changes little with size or angle of view whereas the peaks may vary quite a bit.

III.5. Robot Control

III.5.1. Overview

Here is a short overview of how to control the robot with ARIA:
Device Connections are the object that ARIA uses to talk to the robot. After creating and opening a device connection, its association has to be set to a robot. After a device connection is set on a robot, one has to connect to the robot.
ArRobot is the heart of ARIA. **ArRobot** is the gateway through which information is read about the robot and through which commands are sent to the robot.
The robot has motors, which must be enabled for the robot to move before any commands can be sent. There are two ways of controlling the robot. One is by Actions (the normal method of controlling a robot) and the other one is with Direct Motion Commands.

- Each action usually embodies one idea of some kind, such as avoiding obstacles in front of the robot, or recovering from a stall. They work together to let the robot move around robustly.
- Direct Motion Commands are the method of giving the robot explicit movement commands, they should not be used in conjunction with actions.

Some more things that should be mentioned:

- State reflection is what the **ArRobot** class is basically for; it takes readings from the robot and reflects them, and takes movement commands and reflects them back down the connection to the robot.
- State reflection can be turned off but be careful with sending too many commands (as they are not being reflected anymore, and the robot receives directly each command sent)

- Aria is highly multithreaded:
  - With threads, multiple operations can happen in the same instant in time. So careful with reading from the same data from two different threads or writing data another thread is reading.
  - Sync tasks are the tasks that are run once per robot cycle, in a specified order. The sync task list is actually a tree, with 5 major branches by default, packet handling, sensor interpretation, action handling, state reflecting, and user tasks
  - An asynchronous task runs in its own thread

III.5.2. Simplified robot control

To simplify the robot control, we use the direct motion commands and we let the robot run in its own thread. First, a serial connection is created and set to the robot. Then the connection is opened. Now we can send commands to the robot and we let it run in its own asynchronous thread. We have defined a set of simple direct motion commands:

- Go (distance in mm, velocity in mm per second)
- Turn (angle in degree, rotation velocity in degree per second)
- Stop
- Beep (different sounds)

This small command collection can easily be extended, for example with a command that sets the acceleration or makes the robot go to a given point etc.

IV. Implementation

IV.1. People Tracking

In order to save a lot of processor time, a different algorithm has been implemented for keeping track of a person once he / she is found. At startup, the robot has no person to track, so it waits until an object appears in its field of view (which is only scanned partially). As soon as an object appears in front of the robot, a full image scan is performed, the object is assigned an ID and its position and size are calculated. Then only the sub band is processed again and the dx, dy, dz is constantly updated in order to estimate the objects trajectory. If the object moves within these boundaries, there is no need to re-process the full image and the object keeps being tracked. Another method to track people is using the color-indexing algorithm in order to keep track of the right object (person). This is especially advantageous, if the robot's trajectory is not smooth.
If the object to be tracked / followed is lost, the robot tries to find the person again (using the color indexing method). In case of lost track, the robot stops and tells the user to slow down, respectively to come back (using a simple beep).

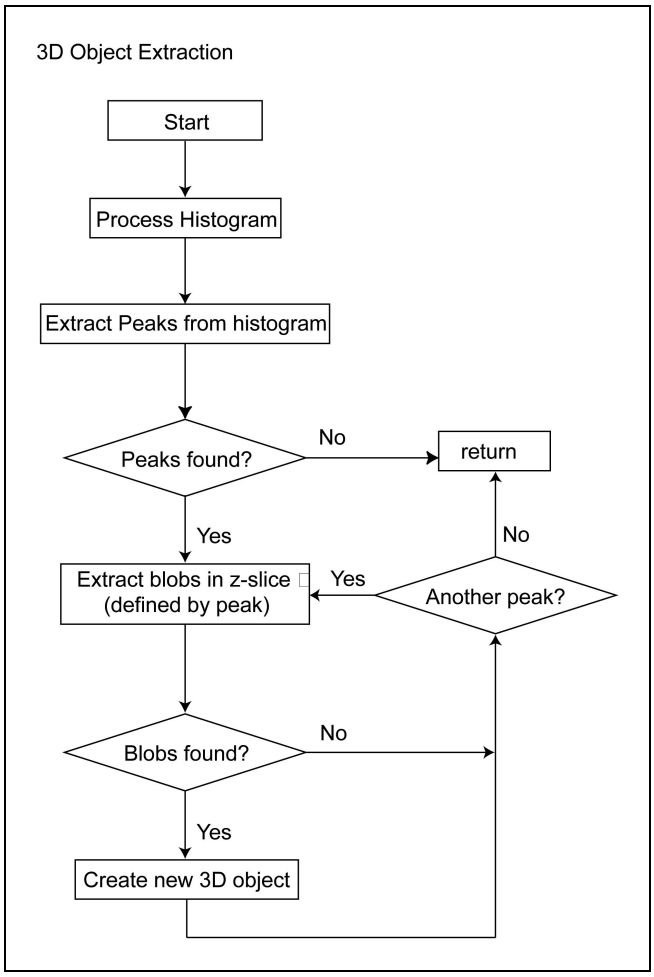


Figure 19: Program schema of 3D object extraction

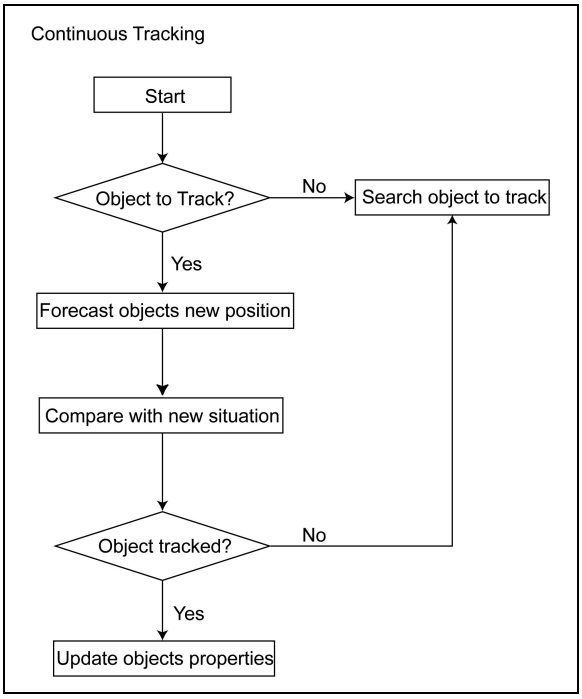


Figure 20: Program schema for continuous tracking

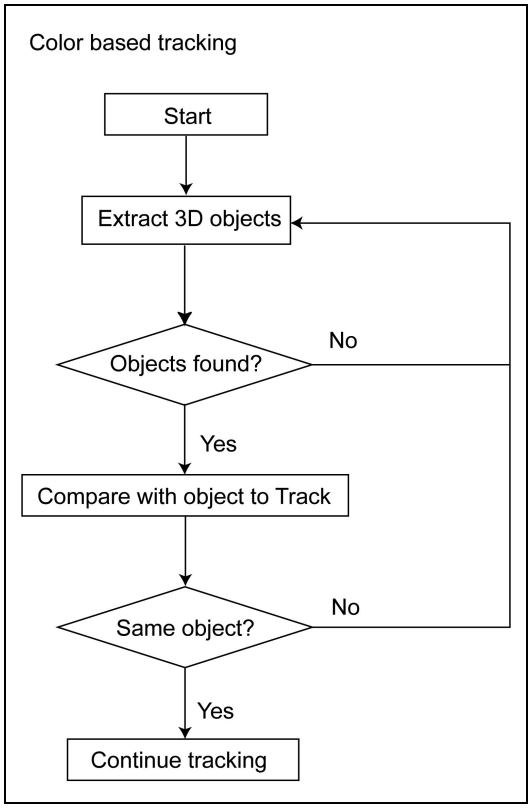


Figure 21: Program schema color tracking

IV.2. Gesture recognition

The gesture recognition part of the program reuses big parts from the code described above for the 3D object extraction. It is however combined with information from the skin color filtering. When the robot expects a gesture, it uses the result from the tracker engine in order to estimate the search area for the head. As the lower part of the body is tracked, we can make the hypothesis that the head is above that area but not further away than approximately one meter. In this search region, the biggest and closest skin color blob is searched and, if found, stored in a class tlSimpleGesture. If no head can be found using this method, the rest of the image is scanned and the biggest object (in the real world) is supposed to be the head. As we now know the position of the head, we try to localize the hands on the left (respectively on the right) of the head's x-position. In this area also, the biggest and closest skin colored object is searched. An additional test is performed which checks the angle between head and hand and rejects the invalid hand objects.

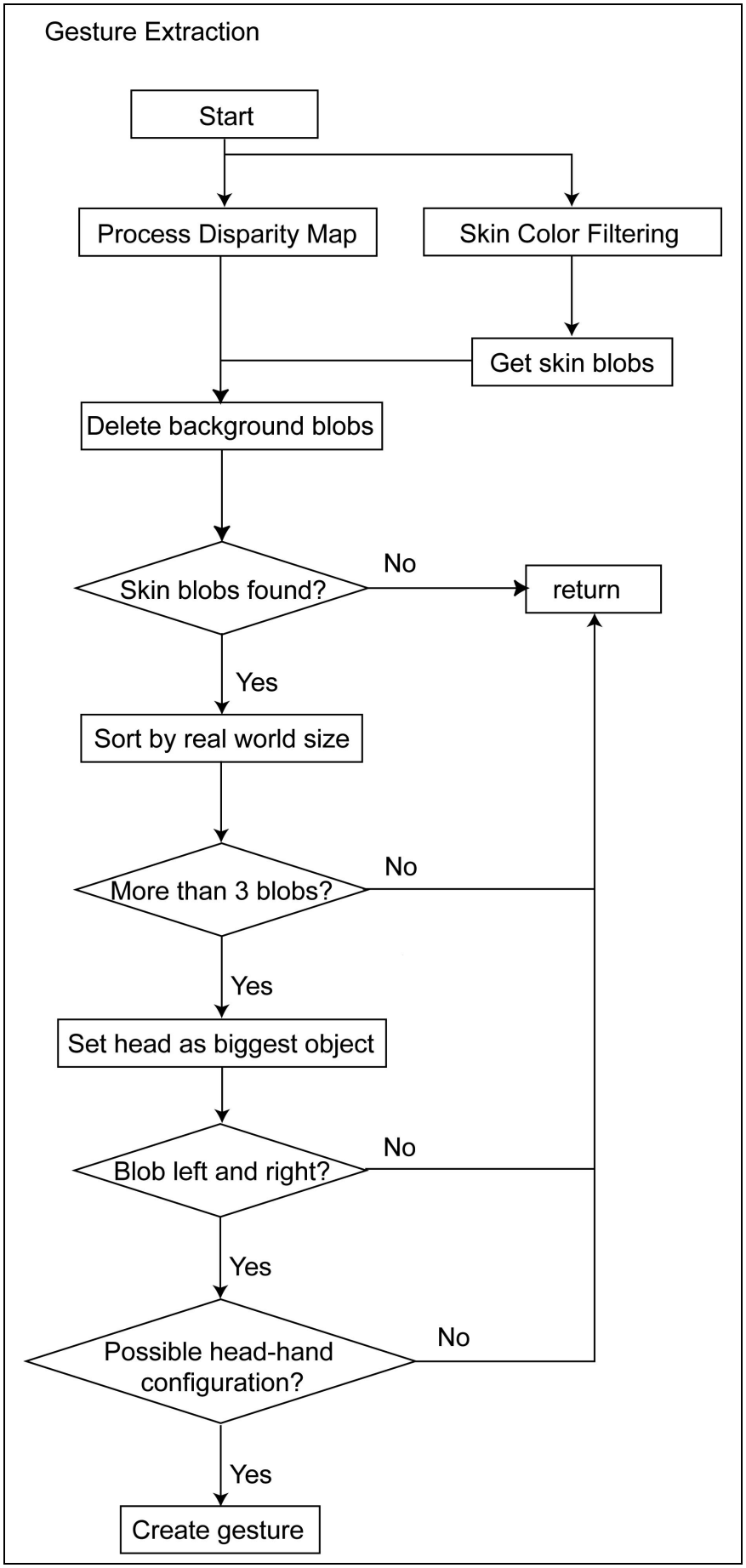

Gesture Extraction
Start
Process Disparity Map
Skin Color Filtering
Get skin blobs
Delete background blobs
Skin blobs found?
No
return
Yes
Sort by real world size
More than 3 blobs?
No
Yes
Set head as biggest object
Blob left and right?
No
Yes
Possible head-hand configuration?
No
Yes
Create gesture

Figure 22: Program schema Gesture Extraction

IV.3. People Identification

People identification is done with the class tlbPerson2. An instance of this class represents a person with the normalized color histograms as attributes. To define an instance of the class tlbPerson2 one can both use the constructor or the method setPerson (tlRect *anyRect, tlImage *anyImage) and pass the rectangle as well as the image out of which the rectangle should be extracted from as argument. Then the normalized red-green histograms will be created.
The method float comparePerson (tlbPerson2 *person) takes another person as argument and gives back the matching value between zero (no match at all) and one (complete match).

IV.4. Robot Control

The class to control the robot is called tlbRobot3. Its attributes include a connection ArSerialConnection and a robot ArRobot. The constructor first enable the robots motors, turn off its ultra sonic sensors (not needed) and turn on the sounds (so the robot can beep). Then it will open the device connection ArSerialConnection (return if failed) and set it to the robot (and shut it down in case of failure). If connection to the robot is established, the robot will be run in its own thread with the command runAsync().
Once the constructor has been called, the simple commands go, turn, beep, stop, exit can be used. The state reflection has been turned off because it causes weird behavior with direct motion commands. The commands are sent with the comInt(ArCommands) method, where ArCommands is a useful enumeration (enum) of commands.

IV.5. Coding

The whole program has been written in C++. Classes have been extensively used. There is one tlTracker3D[1] class, which performs 3D object extraction and tracking. The tlSimpleGesture class performs the correct transformation from a gesture into an executable command for the robot. In order to process a histogram a tlPeak class has been implemented. For re-identification, the tlPerson class is available which stores the color information of the people to track. Finally, a tlObject3D class (used by tlSimpleGesture and tlTracker3D) implements a 3D object.

[1] Note: As this program is based on the tracking library from Sébastien Grange (tLib), the prefix tl has been used.

IV.5.1. Class overview

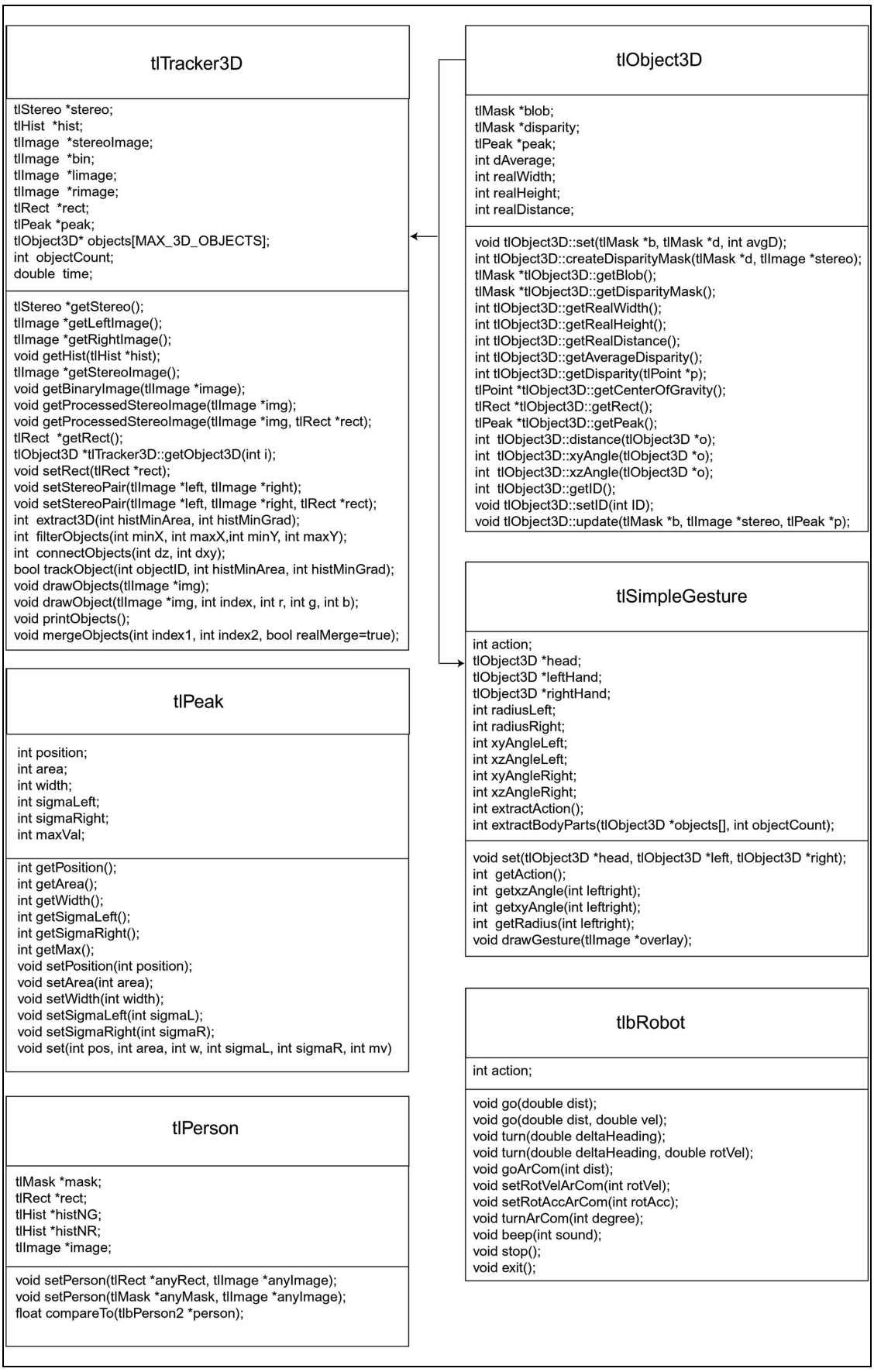


Figure 23: Classes overview

V. Results

V.1. People Tracking

Figure 24 and Figure 25 illustrate the performance of the people detection and tracking algorithm in typical configurations.

Figure 24: Successful tracking of two people (partial overlap in the middle and right image)

Figure 25: Successful object extraction with overlap (left) and unsuccessful (right) due to objects overlapping and same disparity values

V.1.1. Problems

The tracking performance depends on many parameters, which have to be determined empirically, and which depends on the vision system. First, there are parameters for the histogram processing:

- A peak's minimal area (HIST_MIN_AREA[2]).
- The gradient, respectively the speed with which the peak is falling (HIST_MIN_GRADIENT).
- The distance between two peaks in order to connect them (which follows a 1/x law, but which also needs empirically determined correction factors: a/x+b) (_tlDisparityThreshold()).

For the 2D segmentation we need:

- The minimal distance between two blobs in order to connect them (given in centimeters, so this also follows the 1/x law). (_tlConnectDistance())
- The minimal area of a blob in order to distinguish between noise and relevant blobs (BLOB_MINIMAL_AREA)

The same occurs for the 3D object processing. We need again a connection distance and a minimal area (all in real world lengths, therefore again 1/x). Finally we need many camera / system specific constants such as the baseline, the focal length, the angle between the camera's optical axis and the floor, the height of the cameras from

[2] Note: The constant's name from the source code is between parenthesis

the floor and a constant for doing the conversion from disparity to real world lengths (used in all 1/x calculations). At last the skin locus has to be defined as a part of the normalized red / green plane. Tests only have been performed on static cameras, because our robot‘s power supply failed and could not be repaired before finishing this report. In order to see all our constants refer to our definition file in the appendix IX.3.

V.2. Gesture recognition

As mentioned before, it is very important that the robot interprets the gestures very conservatively. This is achieved by using sensor-fusion of depth and color information and by defining some limits to the degrees of freedom of the human’s arms.

Figure 26: Successful gesture extraction: go back, follow me, stop, go forward (from top left to bottom right)

Figure 27: No gesture could be extracted because hand has no skin color (due to saturation effects around the hand)

Problems

- Better no action perform then perform a false action
- Very noisy

V.3. People Identification

Below we see some examples where more than one person appears in front of the robot. The robot should only track the person with white trousers and ignore the other people appearing. Each person that appears is compared to the person to track and in case of a correspondence less than 0.5, the person found is considered as different from the one to track.

Figure 28: The person to track, standing alone

Figure 29: Second person appearing (left), coming closer but stays untracked (right)

Figure 30: Even closer (left), third person appearing (right)

The lightning conditions influence the robustness of re-identification even when using normalized colors. The results under strong illumination are superior to those with less lighting, because the difference between darker colored clothes and pure black tends towards zero. Also the direction of illumination influences the system, because shadows can be interpreted as darker colors and thus changes the histogram.

Another problem is that the tracker does not always detect the center of a person correctly so the wrong area is chosen as a reference for identification. Maybe a higher but thinner rectangle would give better results.

V.3.1. Problems

Result depends

- On the difference of the clothes being used
- On the light conditions
- On the result of the tracking (Is the person where the tracker thinks it is?)

VI. Future Work

Our work already covers the bases for a successful people tracking and gesture recognition system. Although a lot of work still can be performed in order to improve this system:

- Adding an infrared camera could be very useful to distinguish between humans and dead objects.
- Adding more processor power (a portable computer) would make it possible to use higher resolution images, having more details, especially for skin color detection. Higher resolution images would result in more z-slices, and therefore more detailed 3D objects could be extracted.
- With a better stereovision algorithm, more details would be visible and less noise would be present (see [20]).
- Implement a more complex model of human beings for more gesture recognition.
- VRML on PDA. For collection and visualization of 3D data, PDAs can be used. In order to acquire a model of objects in VRML a stereovision system and the presented 3D tracker may be used. Applications: geographic information systems, distant visualization.
- Adding a more sophisticated robot controller for smoother following, allowing this way better forecasting of the objects trajectories (in fact, this has not been tested yet).
- Adding pan / tilt / zoom cameras to the robot. Therefore, it could track the operator even when he is giving a command like “go there”. The zoom would help to command the robot in a larger zone. An auto-focus finally would result in more sharpen images, showing much more details, but would bring up significant calibration problems that would have to be solved.
- Add voice feedback. If the robot could tell the user that it has lost him, or that the user is going too fast it would improve the communication between man and machine. A simple beep or so is too annoying in long terms.
- Add voice processing. The robot should also understand spoken commands.

VII. Conclusion

Even with limited processing power, it was possible to design very efficient algorithms in order to

- Track people,
- (Re-)identify them
- Understand their (static) gestures
- Control a robot

These algorithms were implemented on a mobile platform where changing background and lighting conditions made it more difficult to extract meaningful information from the images. Stereovision has proved to be a very robust and fast method for a tracking engine. As mentioned before, the conditions may change rapidly, being a problem for our color based methods such as gesture recognition and (re-)identification. The color indexing process updates the color profile of a person from time to time, so the changing illumination poses no problem. For the skin color filtering, we simply set our skin color locus to be very tolerant, though producing more noise, which is eliminated by sensor-fusion with the depth information. With these settings, our system worked in office locations.
Our software has been designed to be reusable and many behaviors that are more complex may be added to our work. Because we limited ourselves to low processing power, our work could easily be made more performing by adding a state-of-the-art processor. The use of end-user operating systems such as Windows 98 made

software development easier, but a real embedded OS could also improve our system in terms of speed and stability.
In addition, implementing more sensor modalities would improve robustness even in very complex scenes. Our system has shown the possibility that interaction with machines through gestures is a feasible task and the set of detected gestures could be enhanced to more commands by implementing a more complex model of a human being. In the future, service robots executing many different tasks from house-maid work to nuclear power plant services might arise and become a common part of everyday live normal as computers nowadays.

Lausanne, 11th of February 7, 2002

Christian Wengert Björn Poëll

VIII. References

[1] ARIA by *Activ*Media Robotics, LLC
44 Concord Street, Peterborough, NH 03458, USA
www.ActivMedia.com/robots

[2] Real-time Tracking of Multiple People Using Continuous Detection
David Beymer and Kurt Konolige
Artificial Intelligence Center
SRI International, Menlo Park, CA 94025, USA

[3] Robust Tracking of People by a Mobile Robotic Agent
Rawasek Tanawongsu, Alexander Stoytchev, Irfan Essa
College of Computing, GVU Center
Georgia Institute of Technology
Atlanta, GA 30332-0280 USA

[4] Robust Real-time Face Tracking Using Adaptive Color Model
Gi-jeong Jang and In-So Kweon
Department of Electrical Engineering & Computer Science
Korea Advanced Institute of Science and Technology
373-1 Kuseong-dong, Yuseong-gu, Taejon, Korea

[5] Using the skin locus to cope with changing illumination conditions in color-based face tracking
Maricor Soriano, Birgitta Martinkauppi, Sami Huovinen, and Mika Laaksonen
Machine Vision and Media Processing Unit, Dept. of Elec. Eng. and Infotech Oulu
P.O.Box 4500, FIN-90014 University of Oulu, Finland

[6] http://www.aibo.sony.com

[7] Real-Time Face Tracking Using Audio and Image Data
Prof. Michael Brandstein
Harvard Intelligent Multi-Media Environment Laboratory (HIMMEL)

[8] A gesture Interface for Human-Robot-Interaction
Jochen Triesch and Christoph von der Malsburg
Institut für Neuroinformatik

Ruhr-Universität Bochum,
44780 Bochum, Germany

[9] A Neural-Network Based Approach for Recognition of Pose and Motion Gestures On a Mobile Robot
Stefan Waldherr, Sebastian Thrun and Roseli Remero
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA, USA

[10] Vision-based Sensor Fusion for Active Interfaces: H.O.T. – Human Oriented Tracking
Diploma Thesis 1999/2000
Sébastien Grange
Computer Human Interaction Center
SRI International
Stanford, CA, USA

[11] Toward Real-time Human-Computer-Interaction with Continuous Dynamic Hand Gestures
Yuanxin Zhu
Department of Computer Engineering and Computer Science
University of Missouri-Columbia
Columbia, MO 65211, USA

[12] Recognizing and interpreting gestures on a mobile robot
David Kortenkamp, Eric Huber, R. Peter Bonasso
Metrica Inc.
Robotics and Automation Group
NASA Johnson Space Center – ER2
Houston, TX 77058

[13] Toward Robot Guidance by Hand Gestures Using Monocular Vision
Guangyou Xu, Yuanxin Zhu, Xueyin Lin, Haibing Ren, Xiaoping Zhang
Department of Computer Science and Technology
Tsinghua University
Beijing 100084, China

[14] Color Indexing.
Michael J. Swain and Dana H. Ballard.
International Journal of Computer Vision, 7(1):11--32, 1991

[15] http://www.activrobots.com

[16] SRI International
Menlo Park, CA, USA

[17] Background modeling for segmentation of video-rate stereo sequences.
Christopher Eveland, Kurt Konolige and Robert C. Bolles.
In proceedings IEEE Conf. On Computer Vision and Pattern Recognition, pages 266-271, June 1998

[18] Introduction à l'analyse numérique
Jacques Rappaz, Marco Picasso
Presses polytechniques et universitaires romandes, 1998, ISBN 2-88074-363-X

[19] An Introduction to the Kalman Filter, Greg Welch and Gary Bishop, TR 95-041, Department of Computer Science, University of North Carolina at Chapel Hill, NC 27599-3175

[20] Global Matching Criterion and Color Segmentation Based Stereo
Hai Tao and Harpreet S. Sawhney

Sarnoff Corporation
201 Washington Rd., Princeton, NJ 08543

IX. Appendix

IX.1. Class documentations

Every piece of source code is extensively documented. Some fixed rules have been used in order to extract the documentation from the source code into a HTML help file using doxygen[3]. This permits easier reuse of these classes.
The help files as well as all the source code can be found on the CD going with this report.

IX.2. File overview

Header	Implementation	Description
defines.h	-	Constants, Macros, empirical values
tlObject3D.h	tlObject3D.cpp	3D object
tlPeak.h	tlPeak.cpp	Histogram peak
tlTracker3D.h	tlTracker3D.cpp	Tracking engine
tlSimpleGesture.h	tlSimpleGesture.cpp	Simple gesture extraction
tlbPerson.h	tlbPerson.cpp	Color indexing and re-identification
tlbRobot.h	robot3.cpp	Robot control
tlProcessBlobs.h	tlProcessBlobs.cpp	2D segmentation
tlProcessObjects3D.h	tlProcessObjects3D.cpp	3D objects extracting and tracking
tlProcessHistogram.h	tlProcessHistogram.cpp	Histogram processing
tlProcessGestures.h	tlProcessGestures.cpp	Gesture preprocessing

Table 1: File overview

[3] http://www.doxygen.org

IX.3. Header files

tlObject3D.h

```
#include "tlVision.h"
#include "tlStereo.h"
#include "defines.h"
#include "tlPeak.h"
/**
  * class tlObject3D
  * An object describing a 3D object.
  *
  * @Author:      Christian Wengert
  * @Date:        7.01.2002
  * @Project:     TLIB
  * @Version:     1.0
  * @ToDo:        -
  *
  * @see          method descriptions
  *
  */
class tlObject3D {
private:
      //fields
      tlMask *innerBlob;      ///x,y,area,width,height,cx,cy,pixels;
      tlMask *disparity;      ///contains the disparity map of the object
      tlPeak *peak;           ///corresponding peak in the histogram
      int dAverage;           ///Average disparity of this blob
      int realWidth;          ///width in the real world
      int realHeight;         ///height in the real world
      int realDistance;       ///distance in the real world
      int ID;                 ///the ID of this objects, is given by tracker
      bool tracked;           ///true if already tracked, false otherwise
      int lastDisparity;      ///lastDisparity from being tracked (z)
      tlPoint *lastPos;       ///the last position in (x,y)
      int deltaX;             ///the speed of this object (disparity speed)
      int deltaY;             ///the speed of this object (disparity speed)
      int deltaZ;             ///the speed of this object (disparity speed)
      int realdeltaX;         ///the real world speed
      int realdeltaY;         ///the real world speed
      int realdeltaZ;         ///the real world speed
      //methods
public:
      //constructors
      tlObject3D::tlObject3D();
      tlObject3D::tlObject3D(tlObject3D *o);
      tlObject3D::tlObject3D(tlMask *b, int objID);
      tlObject3D::tlObject3D(tlMask *b, tlImage *stereo, int objID);
      tlObject3D::tlObject3D(tlMask *b, tlImage *stereo, int avgD, int ID);
      tlObject3D::tlObject3D(tlMask *b, tlMask *d, int avgD, int ID);
      tlObject3D::tlObject3D(tlMask *b, tlImage *stereo, tlPeak *p,
                             int ID);
      tlObject3D::~tlObject3D();

      //methods
      int tlObject3D::distance(tlObject3D *o);
      ///measures distance between two objects in real world coords [cm]
      int tlObject3D::xyAngle(tlObject3D *o);
      ///measures angle between two objects in real world coordinates [°]
      int tlObject3D::xzAngle(tlObject3D *o);
      ///measures angle between two objects in real world coordinates [°]
      void tlObject3D::set(tlMask *b, tlMask *d, int avgD,int id);
      ///sets the object
      int tlObject3D::createDisparityMask(tlMask *d, tlImage *stereo);
      ///creates disparity mask from stereo image
```

```
      tlMask *tlObject3D::getBlob();
      ///returns a pointer to this objects blob
      tlMask *tlObject3D::getDisparityMask();
      ///returns a pointer to this objects disparity mask
      int tlObject3D::getRealWidth();
      ///returns the real world width
      int tlObject3D::getRealHeight();
      ///returns the real world height
      int tlObject3D::getRealDistance();
      ///returns the real world distance between robot and object
      int tlObject3D::getAverageDisparity();
      ///returns the average disparity from this objet
      int tlObject3D::getDisparity(tlPoint *p);
      ///returns the disparity at point p from this objet
      tlPoint *tlObject3D::getCenterOfGravity();
      ///returns the the center of gravity
      tlRect *tlObject3D::getRect();
      ///returns the boundaries of the object
      tlPeak *tlObject3D::getPeak();
      ///returns the peak
      int tlObject3D::getID();
      ///gets the ID of this object
      void tlObject3D::setID(int ID);
      ///sets to ID of the object
      tlPoint *tlObject3D::getLastPosition();
      ///returns the last position
      int tlObject3D::getLastDisparity();
      ///returns the last disparity value (distance)
      int tlObject3D::getRealdeltaY();
      ///returns the real world speed
      int tlObject3D::getdeltaY();
      ///returns the speed
      int tlObject3D::getRealdeltaX();
      ///returns the real world speed
      int tlObject3D::getdeltaX();
      ///returns the speed
      int tlObject3D::getRealdeltaZ();
      ///returns the real world speed
      int tlObject3D::getdeltaZ();
      void tlObject3D::update(tlMask *b, tlImage *stereo, tlPeak *peak);
      ///updates an already tracked object
};
```

tlPeak.h

```
#include "tlVision.h"
#include "defines.h"
/**
  * class tlPeak
  *
  * Describes a peak in a histogram
  *
  * @Author:      Christian Wengert
  * @Date:        7.01.2002
  * @Project:     TLIB
  * @Version:     1.0
  * @ToDo:        -
  *
  * @see          method descriptions
  */
class tlPeak {
private:
      int position;          ///position in the histogram
      int area;              ///area that peak and its neighbors occupy
      int width;             ///width of the peak
      int sigmaLeft;         ///distance to the left to next zero position
      int sigmaRight;        ///distance to the right to next zero position
      int maxVal;            ///maximum value
```

```
public:
      //constructors
      tlPeak();
      tlPeak(tlPeak *p);
      tlPeak(int position);
      tlPeak(int position, int area, int width, int sl, int sr, int mv);
      ~tlPeak();
      int getPosition();
      ///returns the position of the peak
      int getArea();
      ///returns the area of this peak
      int getWidth();
      ///returns the width of this peak
      int getSigmaLeft();
      ///returns distance to next zero to the left
      int getSigmaRight();
      ///returns distance to next zero to the right
      int getMax();
      ///returns the max Val of this peak
      void setPosition(int position);
      ///sets this peaks position
      void setArea(int area);
      ///sets this peaks area
      void setWidth(int width);
      ///sets this peaks width
      void setSigmaLeft(int sigmaLeft);
      ///sets this peaks sigma left
      void setSigmaRight(int sigmaRight);
      ///sets this sigma right
      void set(int pos, int area, int width, int sl, int sr, int mv);
      ///sets all properties
};
```

tlTracker3D.h

```
/**
  * class tlTracker3D
  *
  * Contains a full 3D-objects processing and extracting engine.
  *
  * @Author:      Christian Wengert
  * @Date:        7.01.2002
  * @Project:     TLIB
  * @Version:     1.0
  * @ToDo:        -
  *
  * @see          method descriptions
  */

//include files
#include "tlVision.h"
#include "tlDisplay.h"
#include "tlStereo.h"
#include "tlObject3D.h"
#include "tlProcessBlobs.h"
#include "defines.h"
//defines
class tlTracker3D {
private:
      //fields
      tlStereo *stereo;       ///the stereo processor
      tlHist *hist;           ///the stereo histogram for 3D segmentation
      tlImage *stereoImage;   ///the processed stereo image
      tlImage *bin;           ///the binary for the 2D segmentation (x,y)
      tlImage *limage;        ///the left eye image
      tlImage *rimage;        ///the right eye image
      tlRect *rect;           ///the portion of the image to process
      tlObject3D* objects[MAX_3D_OBJECTS];
```

```
	///a list of all objects,
	int objectCount;	///the number of objects
	int time;		///used to calc the speed of objects
	//methods
	void sortObjects(int left, int right);
	void swapObjects(int i, int j);
public:
	//methods
	tlStereo *getStereo();
	///returns the tlStereo
	tlImage *getLeftImage();
	///returns the left image
	tlImage *getRightImage();
	///returns the right image
	void getHist(tlHist *hist);
	///returns the tlHist
	tlImage *getStereoImage();
	///returns the stereoImage
	void  getBinaryImage(tlImage *image);
	///returns the processed binary Image
	void  getProcessedStereoImage(tlImage *img);
	///returns the processed stereoImage
	void  getProcessedStereoImage(tlImage *img, tlRect *rect);
	///returns the processed stereoImage
	tlRect *getRect();
	///returns the tlRect
	tlObject3D *tlTracker3D::getObject3D(int i);
	///returns a 3D object
	void setRect(tlRect *rect);
	///sets the working rectangle
	void setStereoPair(tlImage *left, tlImage *right);
	///sets the stereoEngine
	void  setStereoPair(tlImage *left, tlImage *right, tlRect *rect);
	///sets the stereoEngine
	int extract3D(int histMinArea, int histMinGrad);
	///extracts the 3D objects and returns the objectCount
	int filterObjects(int minW, int maxW, int minH, int maxH);
	///deletes smaller then minW/minH [cm] and bigger then maxW/maxH [cm]
	int connectObjects(int dz, int dxy);
	///connects objects if disparity value is +/- the same
	///deltaDisparity) of both objects and if distance between them
	///below the parameter
	bool trackObject(int objectID, int histMinArea, int histMinGrad);
	///tracks an object by ID
	void drawObjects(tlImage *img);
	///draws the object in different colors on img
	void  drawObject(tlImage *img, int index, int r, int g, int b);
	///draws the object i color color
	void  printObjects();
	///prints the main properties of the objects
	void  mergeObjects(int index1, int index2, bool realMerge=true);
	///merges two objects
	//constructors
	tlTracker3D::tlTracker3D();
	tlTracker3D::tlTracker3D(tlImage *left, tlImage *right);
	tlTracker3D::tlTracker3D(tlImage *left, tlImage *right, tlRect *r);
	tlTracker3D::~tlTracker3D();
};
```

tlSimpleGesture.h

```
#include "tlVision.h"
#include "tlObject3D.h"
#include "defines.h"

#define		NO_ACTION_FOUND		0
#define		STOP_ACTION		1
#define		FORWARD_ACTION		2
```

```
#define           BACKWARD_ACTION         3
#define           FOLLOW_ACTION           4
#define           TURN_LEFT_ACTION        5
#define           TURN_RIGHT_ACTION       6
#define           STAY_HERE_ACTION        7
/**
  * class tlSimpleGesture
  *
  * Desribes a simple gesture
  * Can be adjusted to a amore complex human model than just head and hands
  * by deriving a complexer class from this one
  *
  * @Author:      Christian Wengert
  * @Date:        1.02.2002
  * @Project:     TLIB
  * @Version:     1.0
  * @ToDo:        -
  *
  * @see          method descriptions
  */
#define LEFT      1
#define RIGHT     2
class tlSimpleGesture {
private:
      int action;             /// meaning of a posture
      tlObject3D *head;       ///a object representing the head
      tlObject3D *leftHand;   ///a object representing the left hand
      tlObject3D *rightHand;  ///a object representing the right hand
      int radiusLeft;         ///distance head left hand
      int radiusRight;        ///distance head right hand
      int xyAngleLeft;        ///angle in xy plane head left hand
      int xzAngleLeft;        ///angle in xz plane head left hand
      int xyAngleRight;       ///angle in xy plane head right hand
      int xzAngleRight;       ///angle in xz plane head right hand
      int extractAction();    ///calculates from the above 6 parameters
      int extractBodyParts(tlObject3D *objects[], int objectCount);
      ///extracts the body parts from a list of 3D objects
public:
      //constructors
      tlSimpleGesture();
      ///standard constructor
      tlSimpleGesture(tlObject3D *head, tlObject3D *l, tlObject3D *r);
      ///constructor with initial posture
      tlSimpleGesture::tlSimpleGesture(tlObject3D *objects[], int count);
      ///extracts itself the objects!!
      ~tlSimpleGesture();
      ///destructor
      void set(tlObject3D *head, tlObject3D *left, tlObject3D *right);
      ///sets a new posture
      int getAction();
      ///returns the command of this gesture
      int getxzAngle(int leftright);
      ///returns the vertical angle between head and left/right arm
      int getxyAngle(int leftright);
      ///returns the horizontal angle between head and left/right arm
      int getRadius(int leftright);
      ///returns the distance between head and left/right arm
      void drawGesture(tlImage *overlay);
      ///draws the posture on the image overlay
};
```

tlbPerson.h

```
/**
  * class tlbPerson
  * Used for color indexing and re-identification
  *
  * @Author:      Bjoern Poell
```

```
  * @Date:        7.01.2002
  * @Project:     TLIB
  * @Version:     1.0
  * @ToDo:        -
  *
  * @see          method descriptions
  *
  */

class tlbPerson2
{
private:
public:
      tlMask *mask;
      tlRect *rect;
      tlHist *histNR;
      tlHist *histNG;
      tlImage *image;
      //constructors
      tlbPerson2();
      tlbPerson2(tlMask *anyMask, tlImage *anyImage);
      tlbPerson2(tlRect *anyRect, tlImage *anyImage);
      ~tlbPerson2();
      //person functions...
      void setPerson(tlRect *anyRect, tlImage *anyImage);
      ///sets person properties
      float compareTo(tlbPerson2 *person);
      ///compares two persons
};
```

tlbRobot.h

```
#include "Aria.h"

/**
  * class tlbRobot
  * Used for robot control
  *
  * @Author:      Bjoern Poell
  * @Date:        7.01.2002
  * @Project:     TLIB
  * @Version:     1.0
  * @ToDo:        -
  *
  * @see          method descriptions
  *
  */

class tlbRobot {

public:
      //constructor
      tlbRobot();
      //destructor
      ~tlbRobot();

      //a pointer to the robot,"new ArRobot()" will be called in tlbRobot()
      ArRobot *robot;
      ArTcpConnection con;
      //methods
      //arg: distance in mm
      void go(double dist);
      //arg: distance in mm velocity in mm per second
      void go(double dist, double vel);
      //arg: delta heading in degree
      void turn(double deltaHeading);
      //arg: delta heading in degree,rotation velocity in degree per second
      void turn(double deltaHeading, double rotVel);
```

```
      //arg: dist in mm
      void goArCom(int dist);
      //arg: deg per sec
      void setRotVelArCom(int rotVel);
      //arg: acc in deg per sec per sec
      void setRotAccArCom(int rotAcc);
      //arg: delta heading in degree, this is with the comInt command,
      void turnArCom(int degree);
      //arg: each int makes a diffrent sounds
      void beep(int sound);
      void stop();
      void exit();
};
```

tlProcessBlobs.h

```
/**
  * tlProcessBlobs.h
  *
  * @Author:      Christian Wengert
  * @Date:        7.01.2002
  * @Project:     TLIB
  * @Version:     1.0
  * @ToDo:        -
  *
  * 2D segmentation from binary image
  * Blob processing routines
  */

#ifndef __TLPROCESSBLOBS_H__
#define __TLPROCESSBLOBS_H__
#include <string.h>
#include "tlVision.h"
#include "defines.h"
#define _TL_CAMERA_EXTRA_BLOB_WIDTH  10
#define _TL_CAMERA_EXTRA_BLOB_HEIGHT 10

int _tlFindBlobs(tlImage *image, tlRect *rect);
///low level extraction method
int _tlLabelBlobs(tlPixel *src, tlPixel *dest, int w, int h, int x, int y);
///low level extraction method
int _tlExtractBlobs(tlImage *image, tlRect *rect);
///low level extraction
int _tlFilterBlobs(int min_size, int max_size, int ratio);
///filters blobs by size and ratio
int _tlFilterBlobs(int area, int ratio);
///filters blobs by area and ratio
int _tlFilterBlobs(int area);
///filters blobs by area
int _tlProcessBlobs (tlImage *bin, tlRect *rect);
///extracts blobs from binary image
int _tlProcessBlobs (tlImage *bin, tlRect *rect, int connectDist);
///extracts blobs from binary image and connects those which inter-distance
///is below connect_distance
int _tlProcessBlobs (tlImage *bin, tlRect *rect, int connectDis, int area);
int _tlProcessBlobs (tlImage *bin, tlRect *rect, int minBlobArea,
                     int connectDist, tlImage *stereo, int diparity);
///extracts blobs from binary image and connects those which inter-distance
///is below connect_distance and filters those which area is below min_area
tlMask *_tlGetBlob(int index);
///returns a blob from the list
int _tlGetBlobCount();
///returns the blob count
void _tlSwapBlobs(int i, int j);
///swaps two blobs in the list
void _tlSortBlobs(int left, int right);
///sorts the blobs in the list by area
int _tlConnectBlobs(int distance);
```

```
///connects blobs which inter distance is below distance
int _tlConnectBlobs(int distance, tlImage *image, int disp);
///connects blobs which inter distance is below distance. But also checks
///(dis-)continuities in the disparity image
void _tlMergeBlobs(int index1, int index2, bool realMerge=true);
///merges two blobs. Called by connect blobs
void _tlPrintBlobs();
///prints all relevant infos about the blobs in the list
int  tlTrackBlobs(tlImage *bin, tlImage *stereo, tlRect *rect, int area,
                  int threshold, int distance);
///high level blob tracking routine
int _tlTrackBlobs(tlImage *bin, tlImage *stereo, int area, int threshold,
                  int distance);
///high level blob tracking routine
int _tlDrawBlobs(tlImage *refImage, int obj, int r, int g, int b);
///draws all blobs
float _tlConnectDistance(int dist, int disp);
///returns the connect distance by 1/x law
```

tlProcessObjects3D.h

```
/**
  * tlProcessObjects3D.h
  *
  * @Author:      Christian Wengert
  * @Date:        7.01.2002
  * @Project:     TLIB
  * @Version:     1.0
  * @ToDo:        -
  *
  * 3D segmentation from 2D segmentation and histogram and stereo data
  * tlObjects3D processing routines
  */

#include "tlVision.h"
#include "tlObject3D.h"
#include "defines.h"

int _tlExtractObjects3D(tlImage *stereo, tlRect *rect, tlHist *hist,
                        tlObject3D *objects[], int histMinArea,
                        int histMinGrad);
///extracts all 3D objects
void _tlDrawObjects3D(tlImage *temp,int numberOfObjects,
                      tlObject3D *objects[]);
///draws all 3D objects
bool _tlTrackObjects3D(tlImage *stereo, tlImage *bin, tlRect *rect,
                       tlHist *hist, tlObject3D *objects[],
                       int histMinArea, int histMinGrad,
                       int oldObjectCount, int objectID);
///tracks all 3D objects
tlObject3D *getObject3DByID(int id, int objectCount,
                            tlObject3D *objects[]);
///returns the object which corresponds to the ID
void _tlMergeObjects3D(tlObject3D *objects[], int index1, int index2);
///merges two objects and also all properties
int _tlConnectObjects3D(tlObject3D *objects[], int objectCount, int dz,
                        int distance, tlImage *stereo);
///connects objects thar are overlapped
void _tlSortObjects(int left, int right, tlObject3D *objects[]);
void _tlSortObjectsByX(int left, int right, tlObject3D *objects[]);
void _tlSortObjectsByRealSize(int left, int right, tlObject3D *objects[]);
void _tlSwapObjects(int i, int j, tlObject3D *objects[]);
int tlcmToPixel(int cm, int disparity);
int tlPixelTocm(int pixel, int disparity) ;
int _tlFilterObjects3DBySize(int wmin, int wmax, int hmin, int hmax, int
                             objectCount, tlObject3D *objects[]);
```

tlProcessHistogram.h

```
/**
  * tlProcessHistogram.h
  *
  * @Author:      Christian Wengert
  * @Date:        7.01.2002
  * @Project:     TLIB
  * @Version:     1.0
  * @ToDo:        -
  *
  * Histogram processing routines
  * extracts peaks and permits to process them
  *
  */
#include "defines.h"

int _tlProcessHist(tlImage *stereo, tlRect *rect, tlHist *hist, int
                   min_area, int min_gradient);
///processes the histogram and creates a list of the peaks of this
///histogram
void _tlSortPeaks(int left, int right);
///sorts peaks in the list by disparity
void _tlSwapPeaks(int i, int j);
///swaps two peaks in the list
void _tlPrintPeaks(int peakCount);
///prints relevant informations about the peaks in the list
void  _tlDrawHist(tlImage *imageIn, tlImage *imageOut, tlHist *hist,
                  tlRect *rect);
///draws the histogram
tlPeak *_tlGetPeak(int index);
///returns a peak from the list
int tlGetForegroundStart();   1
///returns the disparity value where the foreground starts
int _tlMergePeaks(int index1, int index2,int peakCount);
///merges the properties of peaks into one
void _tlDrawPeaks(tlImage *imageOut, int peakCount);
///draws the peaks
double _tlDisparityThreshold(int d);
///calculates the distance, two peaks must have between them in order to be
///considered as one peak (follows a 1/x law)
int _tlConnectPeaks(int peakCount);
///connects peaks
```

tlProcessGestures.h

```
#include "tlVision.h"
#include "defines.h"
#include "tlTracker3D.h"
#include "tlSimpleGesture.h"
/**
  * tlProcessGestures.h
  *
  * @Author:      Christian Wengert
  * @Date:        7.01.2002
  * @Project:     TLIB
  * @Version:     1.0
  * @ToDo:        -
  *
  * Gesture processing routines
  *
  *
  */
tlSimpleGesture *tlExtractGestures(tlTracker3D *tracker);
///extracts the gesture from the image and returns the action code
void tlFlattenDisparity(tlImage *stereo, tlRect *rect, float angle);
///corrects the disparity, if cameras are tilted
void tlStereoProjection(tlImage *stereo, tlRect *rect, tlImage *output,
```

```
                        tlHist *hist);
///makes a projection of the stereo image->view from top
```

IX.4. Datasheets and specs

IX.4.1. Camera

Owner's Manual
for CS-Mount COLOR CCD Camera
~ CSC - 730 / 735 / 740 / 745 ~

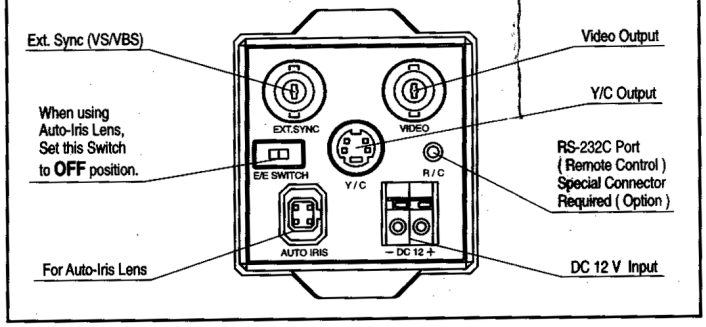
SPECIFICATIONS

Signal System	NTSC	PAL	NTSC	PAL
Effective Picture Elements	510(H) × 492(V)	500(H) × 582(V)	768(H) × 494(V)	752(H) × 582(V)
Lens Mount	CS mount			
Video Output	VBS - 1.0Vpp / 75Ω Synchronous Negative Polarity , Y/C (S-Video) Separate Output			
Power Supply Voltage	DC 7 V~15 V			
Power Consumption	Max 2.4W (Vin = DC 12V)			
Weight	Approx. 160g			

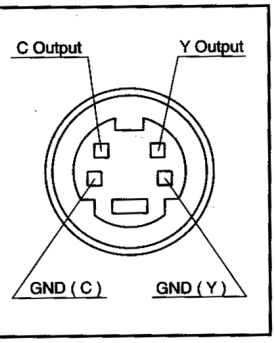
ACCESSORIES

	L Wrench	C-Mount Adaptor Ring	Auto-Iris Connector	Y/C Output Connector
	●	●	●	–
	●	●	●	●

CONNECTION

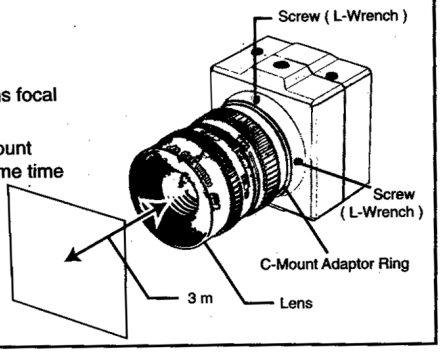


Y/C Output

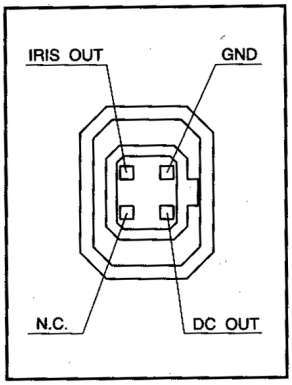


BACK FOCUS ADJUSTMENT

1. Loosen the screws
2. Adjust the backfocus.
   a. Set the lens Iris Open
   b. Set the index of the lens focal length to FAR (∞)
   c. Turn the lens and C-Mount Adapter Ring at the same time to adjust the focus. Choose an aim which locates at more than 3 m away and bring it into focus.
3. Tighten the screws.



Auto Iris Lens



MADE IN JAPAN

Manufactured by
PACIFIC CORPORATION
TOKYO , JAPAN

Owner's Manual
for CS-Mount B&W CCD Camera
~ CSB - 460B / 465B ~

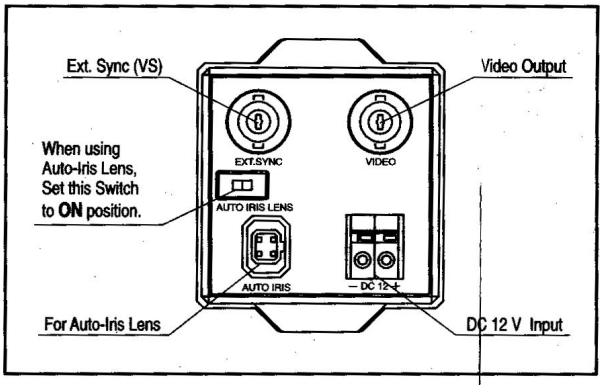
SPECIFICATIONS

Signal System	EIA	CCIR
Effective Picture Elements	510(H) × 492(V)	500(H) × 582(V)
Lens Mount	CS mount	
Video Output	1.0Vpp / 75Ω synchronous negative polarity	
Power Supply Voltage	DC 12V ± 10%	
Power Consumption	Max 1.5W (Vin = DC 12V)	
Weight	Approx. 160g	

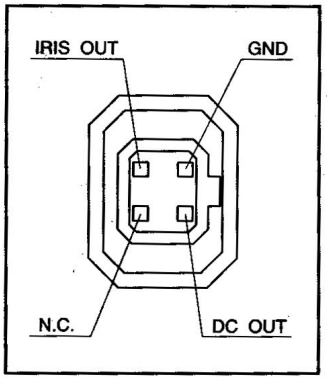
ACCESSORIES

L Wrench	C-Mount Adaptor Ring	Auto-Iris Connector
●	●	●

CONNECTION

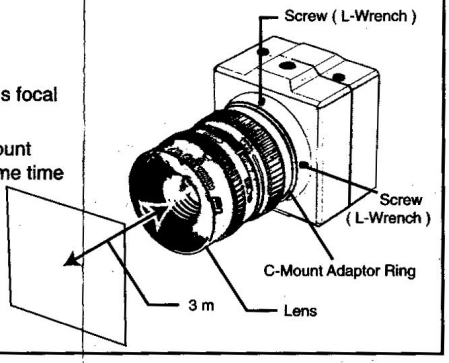


Auto Iris Lens



BACK FOCUS ADJUSTMENT

1. Loosen the screws
2. Adjust the backfocus.
   a. Set the lens Iris Open
   b. Set the index of the lens focal length to FAR (∞)
   c. Turn the lens and C-Mount Adapter Ring at the same time to adjust the focus. Choose an aim which locates at more than 3 m away and bring it into focus.
3. Tighten the screws.



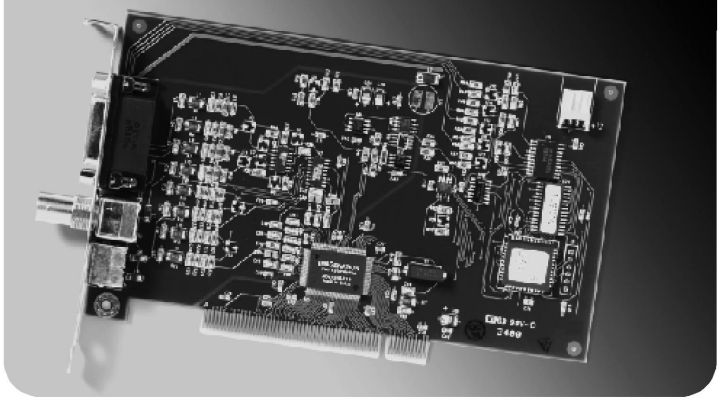
MADE IN JAPAN

Manufactured by
PACIFIC CORPORATION
TOKYO , JAPAN

IX.4.2. Frame grabber

CYBEROPTICS
HAMA SENSORS • IMAGENATION VISION

Imagenation PXC200A Color Frame Grabber



Now you can get the features, flexibility and accuracy you need for high-performance color and monochrome video capture, at a surprisingly affordable price.

Affordable color solution for standard PCI formats

Built on robust features found in much more expensive products. The Imagenation PXC200A Color Frame Grabber combines high quality color and monochrome video capture with a Peripheral Component Interface (PCI) at an unusually affordable price. High accuracy, low pixel jitter and other leading features offer solid support for the most demanding industrial and commercial applications.

Acting as a PCI bus master, the PXC200A achieves real-time video capture to system memory. It handles data transfers while the main CPU is free to run other parts of your application or other applications. Image data can be transferred to a buffer in main memory or directly to another PCI device.

Four multiplexed video inputs can accept color video from NTSC, PAL and SECAM video sources. One of these can also be reserved for S-video. Color formats supported include YCrCb and RGB, while Y8 is supported for monochrome applications. Also included are real-time image scaling with interpolation, plus horizontal and vertical cropping to minimize memory and bus bandwidth requirements.

The PXC200A is easy to integrate into your application, providing support for commercial machine vision software including XCaliper, Halcon and CVB.

PXC200A options contribute higher performance and flexibility. The following features are included at a reasonable price with the PXC200A's optional control package (part number PXC200AF).

- Eight general purpose I/Os – programmable as four separate triggers and four strobes.
- Vertical and horizontal sync out for gen-locking a video source, providing a more stable image.
- Strobe inhibit during CCD transfer time for reliable image capture with strobes.
- A total of four video imputs – any or all of which can be S-video.
- DC restore on all 4 video inputs.

Comprehensive software support for Imagenation products. Demo programs for Windows and DOS make it easy to get the PXC200A up and running right out of the box. They let you capture images, save images to disk and adjust image capture features of the card without writing code.

Software support is also included for Borland and Microsoft C/C ++ compilers and other languages that can call standard Windows DLLs.

Our technical support engineers are on hand to provide assistance throughout your development process. Current software and examples are also available at the Technical Support page of the Imagenation product web site at www.imagenation.com

PXC200A Specifications

Input video formats	NTSC, PAL, SECAM, S-video
Input video	1V peak to peak, 75Ω
Resolution	NTSC: 640 x 480; PAL/SECAM: 768 x 576
Sampling jitter	Maximum of ± 4ns relative to horizontal synchronization (for a stable source)
Output formats	Color: YCrCb 4:2:2 and 4:1:1; RGB 32,24,16,15; Monochrome: Y8
External trigger	Input pulled by 10k Ω to 5V, trigger requires a TTL pulse of 100 ns (minimum); software programmable edge or level sensitivity; software programmable polarity
ESD protection	All inputs and outputs are diode protected
Form factor	PCI short card, 174.6 x 106.7 mm (6.875 x 4.2 inches)
Video noise	≤ 1 LSB (least significant bit)
Camera power	+ 12 VDC output
Video multiplexer	Standard board: four video inputs (only one can be S-video); Optional Control Package: four video inputs, any or all of which can be S-video
Power requirements	+ 5 VDC, 500 mA
Operating temperature	0° C to 60° C
Warranty	One year limited parts and labor

PXC200AF–Optional Control Package Specifications

- Video expanded to 4 S-video inputs
- Independent DC restore on four video inputs
- Fast field detect for unsynchronized sources
- Four TTL inputs, four TTL outputs
- Horizontal and vertical drive outputs
- Inputs and outputs can be configured as triggers and strobes
  - – Programmable trigger edges, debounce
  - – Programmable strobe durations, polarity
  - – Video synchronized strobes programmable to any line
  - – Programmable association between triggers/strobes/grabs/video channel
- Strobes can be inhibited during CCD transfer time (programmable inhibit zone)

PXC200A Ordering Information

PXC200AL	Standard product
PXC200AF	Standard product plus optional control package
CB-008	Video cable for PXC200A (L or F) - 4 Composite cameras, enables basic I/O
CB-009	Video cable for PXC200AF - 4 S-video cameras, enables basic or extended I/O
CB-020	Video cable for PXC200AF - 4 Composite cameras, enables extended I/O

Built-in software protection for the PXC200A is also available.

Contact us for more information or to discuss your application.

Toll free:(800) 366-9131 Phone:(503) 495-2200
Fax:(503) 495-2201
Email:info@imagenation.com Web site:www.imagenation.com

Key Features

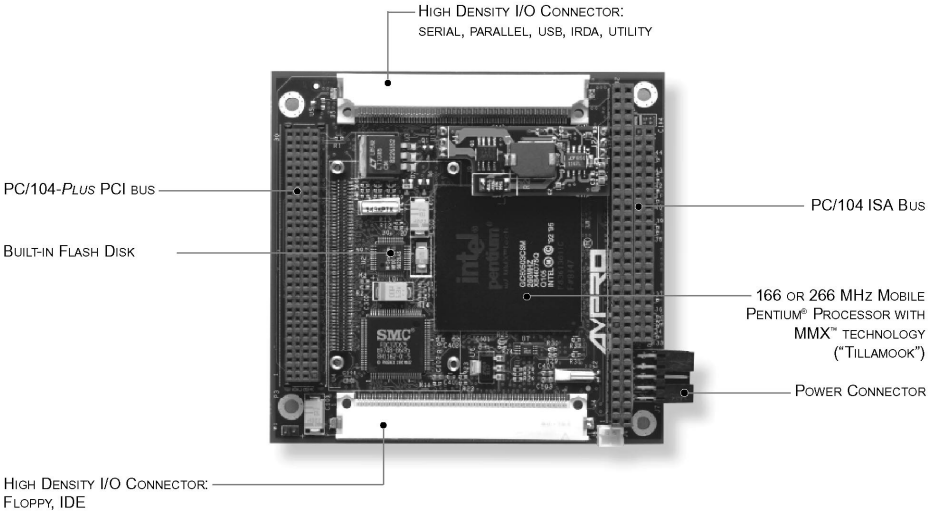
- PCI bus master design for real-time image capture
- Support for YcrCb, RGB and Y8 (gray scale) output formats
- Low pixel jitter
- Standard capture resolutions of 640 x 480 (NTSC) and 768 x 576 (PAL/SECAM)
- Four multiplexed video inputs (NTSC/PAL/SECAM/S-video)
- Real-time image scaling with interpolation, plus horizontal and vertical cropping
- Continuous, software-initiated and triggered capture of frames
- External TTL-level trigger
- + 12 VDC camera power supply
- Simple software interface
- Software development support for DOS, Watcom DOS/4GW extender, Windows 95/98/2000/NT/ME/XP
- Support for C/C++ Visual Basic, Delphi and Direct Draw
- SDK included
- DAC reference generator eliminates inconsistency of AGC
- Verified to FCC Part 15 Class A requirements. Full compliance to CE EMC standards (EN-55022, EN-55024, CISPR-22)

CYBEROPTICS®
HAMA SENSORS • IMAGENATION VISION

IX.4.3. On-board computer

CM3-P5*e*

CoreModule™/P5*e* • *Pentium® performance PC/104-Plus™ compliant CPU module*



This Ampro Pentium® CoreModule is a perfect 'macrocomponent' for developers who want to embed true Pentium performance in their application, but who need to focus their design resources on their application's unique hardware, software, and packaging requirements.

Extensive power management, in both hardware and software, results in extremely low power operation. Intel's "ball grid aray" (BGA) Mobile Pentium processor ("Tillamook") provides superior thermal characteristics and reduces cooling requirements, while allowing the module to operate at extended temperature ranges.

To support the low voltage (3.3 and 1.9 VDC) requirements of the Pentium processor and core logic, Ampro built a highly efficient DC-to-DC converter into the CoreModule/P5*e*, resulting in single-supply (+5 VDC) system operation and minimal power consumption. The module also features a temperature sensor which can trigger clock reduction for proper CPU thermal conditioning.

Two system expansion busses—PCI and ISA—offer the ultimate in system integration flexibility.

Ampro Computers, Inc. • 4757 Hellyer Ave., San Jose, CA 95138 • Tel. (408)360-0200 • Fax (408)360-0222 • sales@ampro.com

CM3-P5e

SPECIFICATIONS

PC MOTHERBOARD FUNCTIONS

CPU	• 166 or 266 MHz Intel Mobile Pentium® processor with MMX™ technology ("Tillamook")
CHIPSET	• Intel 82439TX North Bridge/82371AB South Bridge
MEMORY	• Up to 256 MBytes via socketed DRAM module
SYSTEM CONTROLLERS	• 7 DMA channels (8237 equivalent) • 15 interrupt channels (8259 equivalent) • 3 programmable counter/timers (8254 equivalent)
KEYBOARD AND MOUSE	• Standard PS/2 (PC/AT) keyboard and mouse ports • Speaker port with 0.1 watt output
REAL TIME CLOCK	• Real time clock with CMOS setup; requires external 3.0 - 3.6V battery
BIOS	• Award ROM-BIOS with Ampro enhancements
POWER MANAGEMENT	• Compliant with Intel/Microsoft APM specification (version 1.1 or higher)

ADDITIONAL ONBOARD FUNCTIONS

SERIAL	• Two RS232C serial ports with full handshaking (16550 equivalent) • Serial port 2 supports RS232, TTL, or IRDA
PARALLEL	• IEEE-1284 compatible enhanced parallel printer port with bidirectional data lines
EIDE	• Supports 1 or 2 IDE hard disk drives • Supports Ultra DMA/33 mode transfers (33 MB/sec burst)
FLOPPY	• Supports 1 or 2 drives
USB	• Two USB ports
IRDA	• Infrared interface port • Normal mode supports up to 115.2K Baud • Fast IR mode supports up to 4M bits per second
ONBOARD DISKONCHIP™ OPTION	• Hardware and firmware compatible with M-Systems DiskOnChip2000™ and Millenium • 8 MB storage capacity • Real-time operating system support
CONFIG EEPROM	• Supports battery-free boot capability • 512 bits available for OEM use
WATCHDOG TIMER	• Timeout triggers system management mode interrupt
LOW VOLTAGE RESET	• Triggers when +5V power drops below +4.7V

MECHANICAL

SIZE	• 3.6 x 3.8 x 0.9 in. (90 x 96 x 23mm); PC/104-*Plus* form-factor compliant (Includes stackthrough pins. Please refer to PC/104-*Plus* specification for stacking and other dimensions.)
BUS	• 16-bit PC/104 ISA bus • 32-bit PC/104-*Plus* PCI bus
POWER	• Requirements (mA typical, with 64 Mbytes DRAM and 8 Mbytes onboard DiskOnChip): +5V ±5%

	normal	doze	suspend
166 MHz	1240	660	460
266 MHz	1520	780	520

ENVIRONMENTAL	• Operating temperature: 0° to 70° C standard; –40° to +85° C extended (special order) Note: additional airflow or heatsinking required to maintain 95° C maximum CPU case temperature. • 5% to 95% relative humidity, non-condensing • Storage temperature: –55° to +85° C • Weight: 4.1 oz. (116 gm)

DS-CMP5e 2.5K 0300

THE EMBEDDED SOLUTIONS PROVIDER™

Ampro Headquarters: 1 (800)966-5200 • www.ampro.com