PROJECTE FINAL DE CARRERA

Stereo vision based person detector for rail work environments

Estudis: Enginyeria de Telecomunicació

Autor: Javier Rodríguez Frías

Director/a: Albert Orpella García

Any: 2016

Table of Contents

Chapter 3 Collaborations

1. GMF (COMSA group)
2. Electronics department

Chapter 4 Abstract

Rail work environments present a specially hazardous environment for workers. A system using stereo vision has been devised in order to prevent a machine from starting up when a person is too close in front of the machine.

Chapter 5 Introduction

Railway construction and maintenance works present a special challenge from a safety point of view. The nature of the environment causes workers to be exposed to many different hazards. Workers can suffer injuries from falling objects, tripping, electrocution and so on. Of every possible kind of accident, the ones with the highest fatality rate are those related to the operation of rail work specialized machines.

There are a number of highly specialized machines that are used only in railway environments, such as tamping and profiling machines. These machines often act as a team, but it is often the case that the operations are carried out one at a time, sequentially. That makes the situation specially prone to accidents, as a machine that has been standing by for quite some time can become suddenly operational, potentially catching on foot operators off-guard.

There are all kind of precautions to avoid accidents. The most relevant to the situation from a liability point of view is the presence of the Safety Pilot, whose specific task is to keep the infrastructure workers safe by constantly communicating with everyone involved using a variety of means, typically flags, flashlights, megaphones, and horns. Of course, radio and mobile phone are used to constantly be in touch with nearby train stations and other work sites.

As railways are linear, the machines tend to end up closely arranged in a line. As the machines themselves are quite massive and there may be a number of them, that means the workers can be quite a way away from each other. It is quite likely that obstacles of some sort can hinder communication of some member of the crew to the pilot.

There are a number of factors that increase the likelihood of human error:

- Sleep deprivation.

Infrastructure works often take place during railway down time, in the middle of the night, and the same team of workers often transfers from site to site during the week, having odd work shifts.

- Hindered communication between the Safety Pilot and the work crew.

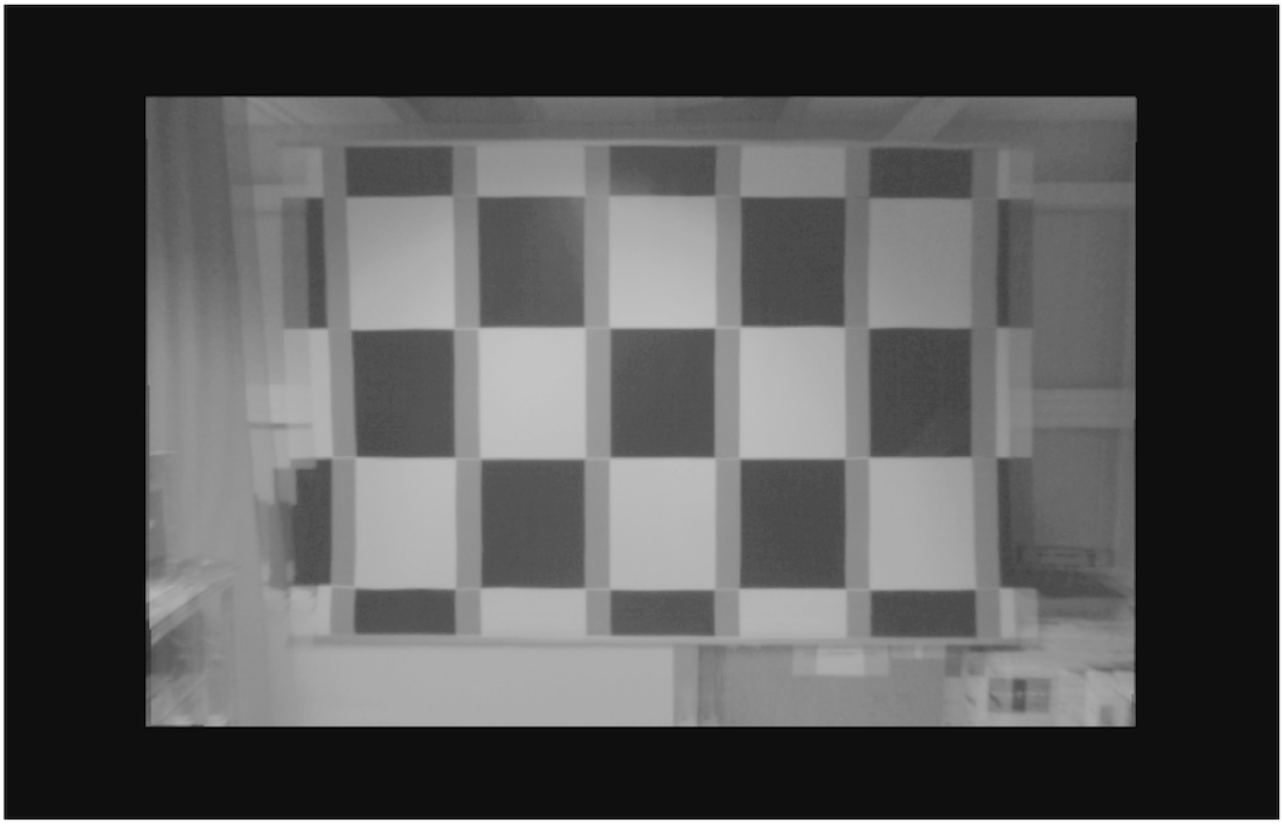
The environment itself is quite noisy while works are taking place. Both the powerful Diesel engines and the moving around of the ballast are incredibly loud and can potentially drown out warning signals.

- No direct vision.

Machine operators often have no direct vision of the track ahead or behind the machines. The driver has to rely on the Safety Pilot to stop or to advance.

The intent of this project is to minimize a certain subset of the risks by minimizing the human error involved.

5.1 What the problem is



We narrow down the scope of the protection system to a particular situation that is both frequent and very likely to cause severe injury to persons. We want to prevent a machine that suddenly starts moving forward or backwards from running over any person that is standing in the immediate vicinity. The kind of machines used in rail works are very heavy but their engines have a lot of horsepower. In stark contrast to their bulky appearance they can suddenly advance forward taking people that are leaning against the stoppers or in the immediate vicinity by surprise. There is great risk of causing irreversible damage to workers. Some instances of this sorry mishap were the motivation for the development of this system.

5.2 Description of the typical environment

The ballast is the name for the bed of rocks that is laid underneath the rails. It supports the weight of the trains and drains wa ter away. Rain and rail traffic wear down the ballast. Insufficient shoulder ballast can cause lateral movement of the rails, and an uneven distribution degrades the quality of the track, making it dangerous for trains to go over a certain speed. To keep tolerances under control the ballast needs to be periodically maintained.

The system can be installed in any kind of machine, but it is originally tailored to meet the requirements of a ballast tamper, as these kind of machines are very commonly used and present special challenges. The task of these machines is to lift the rail and tamp the ballast underneath each sleeper. The structure of the machine is similar to a bridge. Under the bridge's arc several tamping tools are operating. While the machine is working the driver is in charge of controlling both the tamping process and the movement of the machine. The crucial fact is that the driver position is underneath the machine's arc, with no vision whatsoever of anything but the way the ballast is being conditioned.

Illustration 1: Tamping machine

As we can see in figure Illustration 1, there is no visibility of the track at either side of the machine and any person standing there is in immediate danger.

The view from each end of the machine includes machine structural elements such as stoppers or protruding arms, and environment artificial elements, such as the rails, the ballast, the sleepers and oftentimes protective fences.

In a typical environment, another machine may be stationed ahead at a distance, and there may be artificial lightning of the scene and artificial elements. As we shall see, man-made environments pose specific challenges to disparity detection and must be dealt with. Also, some circumstances such as direct sunlight shining down into the objective at sunrise or sunset, or optical reflection in pools of water and the like have to be considered as likely challenges to the system.

There are two clearly delimited modes of operation: work and transport. The transition between those modes is managed by the Pilot. Any kind of alarm system should include the possibility to be bypassed so as to not interfere during transport operations.

Chapter 6 State of the art

When it comes to detecting the presence of human beings, there are a number of alternatives already in use. Following there is a brief summary of the most commonly available technologies.

6.1 RADAR systems

Heavy duty machinery in open mining environments make increasingly use of Radar systems , like Banksman Intelligent Radar System from Vision Techniques, or more recently CatDetect from Caterpillar. These detection systems rely for the most part on displays that are monitored by the machine's operator. It is up to him to judge the kind of obstacle that lies ahead. These systems are pretty effective for these environments but they might be less than ideal for the applications considered in the scope of this project.

Illustration 2: Radar system installed at the back of a vehicle

In Illustration 2 we can see a Vision Techniques system attached to a vehicle.

1. Radar technology is not functional for very short distances, as short as 70cm. This is due to the very nature of the radar system, that disallows simultaneous reception and transmission of a pulse.

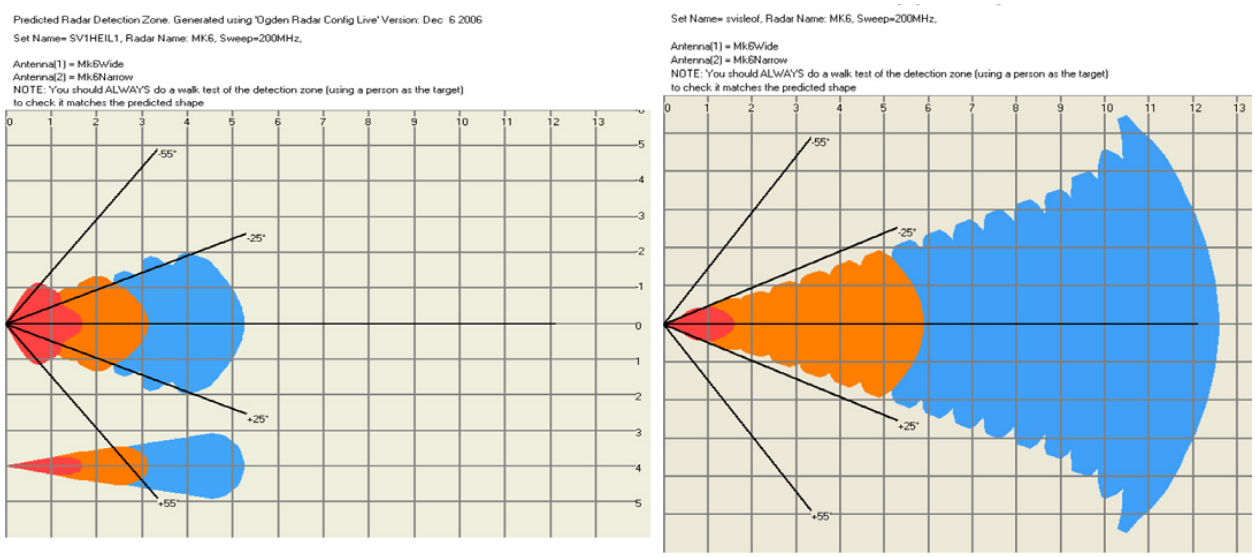


Illustration 3: Emision and detection RADAR beams.

2. Radar detector must be placed so that the beam travels parallel to the ground. Also, the beam spreads like a fan from the sensor position forward. This kind of radiation diagram creates very large blind spots in the vicinity of the vehicle. On the other hand, in the middle distances the beam widens far too much, covering the areas located alongside the rails. This would lead to the apparition of hard to weed out false positives whenever fences, masts or operators are present at either side of the railway.

Illustration 4: Machine structures disallow installation below the stopper line

3. A low height positioning of the detector may hinder the other functional parts of the vehicle that are already in place, like linkage systems, hoses or other sensors As seen in Illustration 4, railway machinery does not offer a wide range of possibilities for the installation of sensors below the stoppers without creating severe functional interference.

6.2 Ultrasound detection

Ultrasound ranging is widely used in the automotive sector as a parking aid. While the sensors are robust and it is relatively easy to adapt them to any structure, they lacks resolution. We went to the extent of building an ultrasound ranging prototype to test the waters, and it didn't look promising at all. While using single sensor data simplifies the processing a lot, in order to discriminate structural obstacles and spurious reflections from people a large array of sensors would be needed, and processing

would be much more complex, without any upfront idea about the final performance of the system.

A qualitative environmental noise level assessment was performed using a simple ultrasound demodulator and it was confirmed that there is in fact a lot of in band ultrasound noise produced by the tamping machine moving the ballast around, and from the ultrasound components of engine noise. While that is in itself not a reason to give up on the possibility of ultrasound detection, there are some practical problems in recreating the works environment in order to test the efficacy of a system. To develop such a system would require constant field testing.

Furthermore, to get a useful representation of the environment using ultrasound it would be necessary to install an array of sensors maybe spanning the width of the machine. That could take up quite a space on each end of the machine. Those areas are already crowded by structural elements, lights and so on, and the different purpose of each machine causes them to have very different geometries. Rounded or angled shapes could affect the necessary calculations. It is unpractical for the purposes of this system to tailor the installation to the extent of modifying noticeably the machines or altering the core algorithms.

6.3 Time of flight detection

Time of flight cameras use the known speed of light to infer the range where objects are at in a given scene. Once the acquisition of the scene is done, the range information is contained in the arrival time of the different parts of the scene. The obvious difficulty lies in the acquisition. Fast and precise gating or sophisticated phase detecting algorithms may be necessary for this approach.

There are a number of available commercial cameras that use this technology. If this approach is chosen then the necessary computations are greatly reduced, because the output of the camera correlates with the disparity matrix, and just some rotations or boundary checks would be necessary after that.

The main drawback of time of flight cameras is the price. It can also be argued that using a ready made device would put the system at the expense of the availability of a particular camera. It would also further complicate the fulfillment of safety and reliability standards, as the core technology of the system would be outside of our control.

6.4 Infrared detection

Human body heat emission sensors (pyroelectric, microbolometer, thermopiles) present also generic detection drawbacks. Works are usually carried out at night and sometimes in very cold weather. In this environment workers wear thick hooded clothes precisely to isolate themselves from ambient temperature. Avoiding body heat loss diminishes FIR emissions, reducing detectability. In hot environments, on the other hand, ambient temperature can reach up to that of the human body.

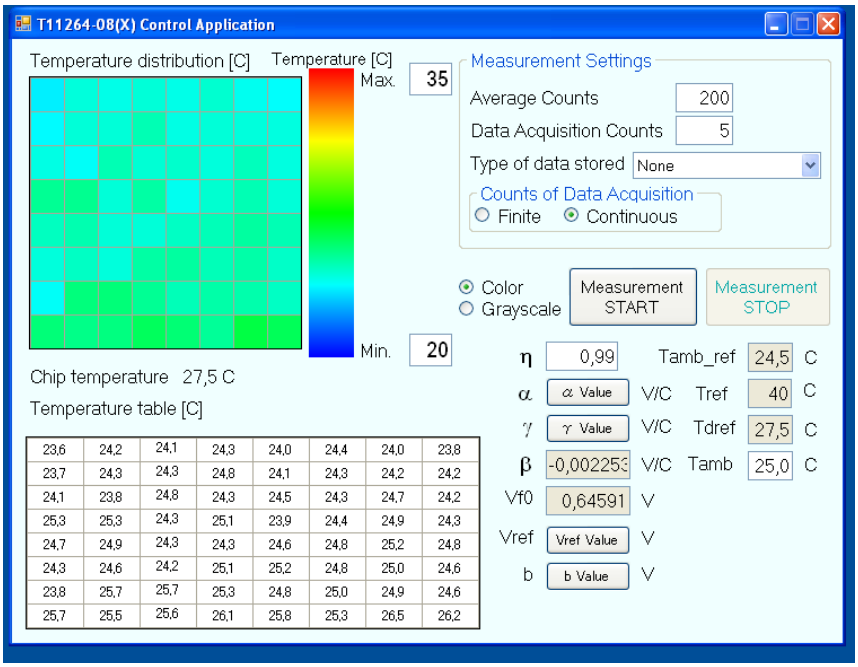


Illustration 5: Infrared picture without any person in view

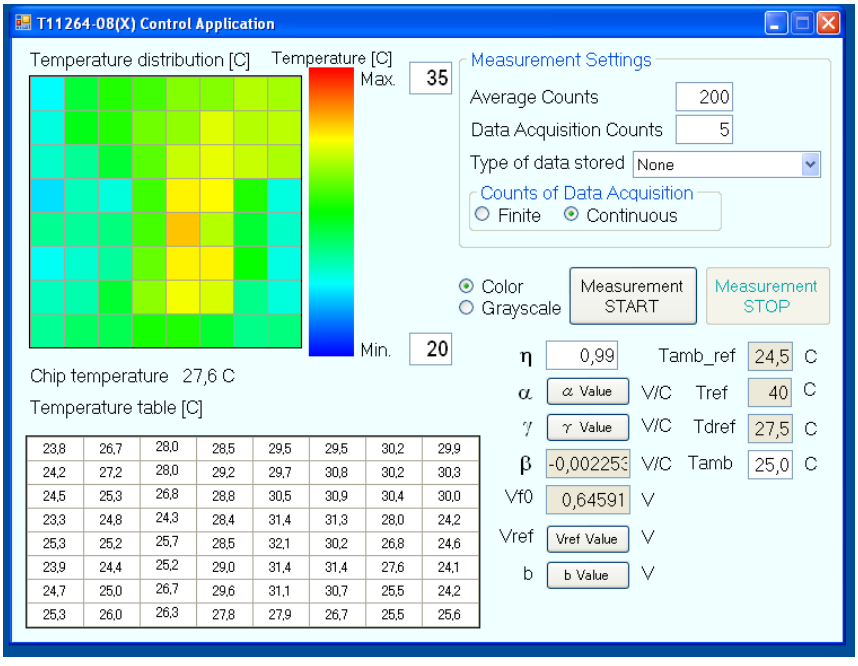


Illustration 6: Infrared picture with a person in view

Some tests have been conducted using a thermopile array. Thermopile arrays are wide spectrum radiation sensors that are often employed in gas detection applications. A Hammamatsu T11264-8 was tentatively evaluated. The resolution, as seen in the pictures, is a 8x8 pixel array.

The test conducted shows that Gaussian noise is very high for a correct discrimination of temperatures at or near human body temperature. Integration times in excess of four seconds are needed in order to filter out the noise and tell apart a human body from its surroundings. This time scale renders this kind of technology unusable for the time being.

6.5 Pattern recognition

Pattern recognition is promising because it would allow for the use of just one camera, doing away with the necessity for accurate alignment and calibration. Using 2D methods also means working with a considerably smaller set of picture elements. As we don't need to consider volumes, everything must be extracted from the features of the original picture instead.

Sophisticated algorithms like those necessary for machine vision get better as more information is known about what to expect. It is easier to inspect a PCB for misplaced components because it is possible to train the system about what to expect. Persons can adopt a variety of stances, crouching, walking, or carrying equipment, and it gets increasingly difficult to detect. In a man made environment of railway machinery, it should also be able to tell people apart from machines or naturally occurring objects like bushes, etc.

Also, even if the system could successfully detect a person, there is still the need to position the person in space. The system should be able to assert that the person is within or without the danger zone with a resolution of a few inches. It would be wrong for the system to trigger the alarm when a person is standing just outside of the danger zone. There is no easy way to infer position in space from a single picture. That in itself would constitute a very complicated problem.

There is also a problem related to the kind of algorithms that are available for 2D pattern recognition. Trained algorithms are often the most effective but do not offer a way to know in advance if the system will be effective. It may very well happen that a person is not detected because the system was trained using a very different set of images as those that happen in a particular instance. The algorithm chosen should make as few assumptions as possible in order not to compromise safety.

As effective as single camera systems are in industrial environments, developing a method to locate objects in space when there is no way to control the environment is extremely problematic.

A nice strategy, which is somehow cheating, would be to assume that every person to be detected is wearing reflective yellow vests. That could greatly facilitate the initial processing of the pictures. Filtering by color or brightness we could reject most of the rest of the picture and achieve great processing speeds.

This idea was rejected at an early stage because we cannot fundamentally guarantee the percentage of rail track workers that consistently don the mandatory individual protection equipment. It has been observed that oftentimes, specially at the beginning or the end of the work shift, protection equipment is neglected. As this is a safety assurance device, there should be no unnecessary assumptions regarding operation circumstances.

Also, even if some special color or pattern was used on the vests and only on the vests, there is still the problem to locate where the persons are. The system should allow normal operation if people are in the field of view but outside of an agreed upon danger area.

6.6 Stereo vision

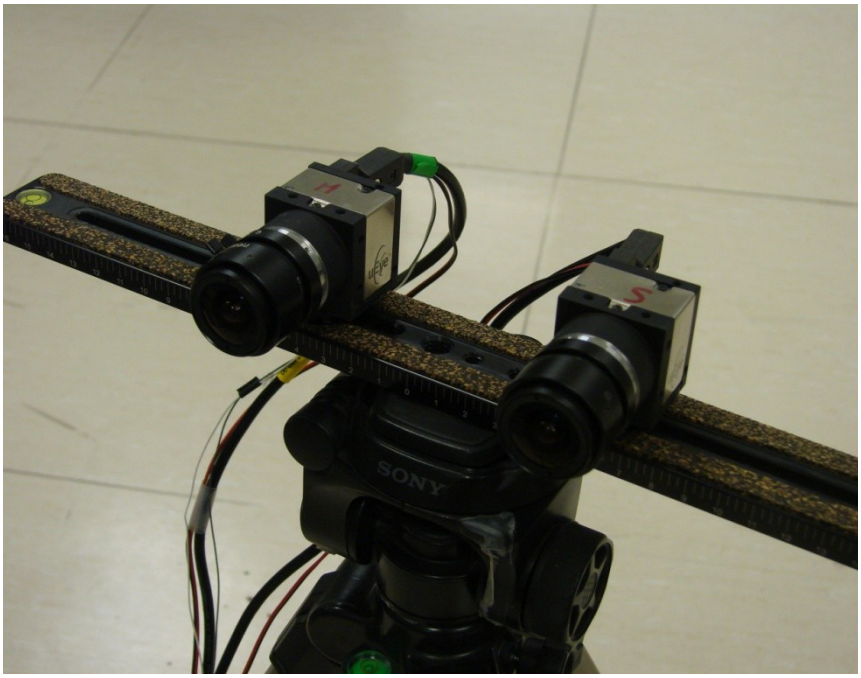


Illustration 7: First camera set prototype

Stereo vision offers both reasonable cost and resolution, with an intuitive map between data acquisition and representation. It does present though some computational challenges.

	Rad	Ultrasou	Time of	Infrar	Pattern	Stereo

	ar	nd	flight	ed	recognition	vision
Installation	NG	NG	OK	OK	OK	OK
Field of view	NG	OK	OK	OK	OK	OK
Data processing	OK	NG	OK	OK	NG	?
Pricing	OK	OK	NG	OK	OK	OK
Responsiveness	OK	OK	OK	NG	OK	?

The table shows a quick summary of available technologies. Stereo vision is in principle promising if the challenges about data processing speed are overcome.

6.7 Mechanical construction

It consists of two units connected by two cables. The camera unit is fixed to the outside of the machine, facing the tracks. The main unit sits at the cabin visible to the driver. Two steel boxes have been developed to encase the units. The requirements of the two boxes are quite different. The outer box is built to withstand outdoor usage. Both boxes should be relatively rugged, and connections need to be watertight. Both container boxes are made of steel.

The outer box is built like a helmet enclosing the camera system, and has a window to allow vision. A rigid polygonal prism has been developed in order to fix the cameras at an angle. Also, the system itself has an infrared luminary. Infrared LED's light up in sync with the shutter, ensuring minimum illumination is present while the system is active. These LED's are fixed between the cameras.

Behind the cameras there is room to fix the serializer and the FPGA board.

The base unit contains the DSP board, the TRACO regulator and necessary connections.

The camera unit box contains a poligonal prism structure to hold the cameras themselves. The outer box is sturdy enough to withstand the environment and is waterproof. A small IR luminary is placed in between the cameras.

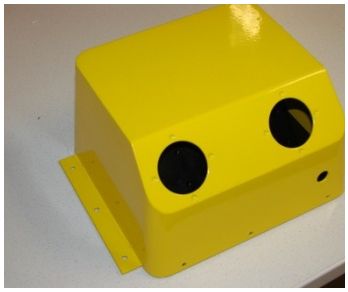


Illustration 8: First iteration of camera enclosure

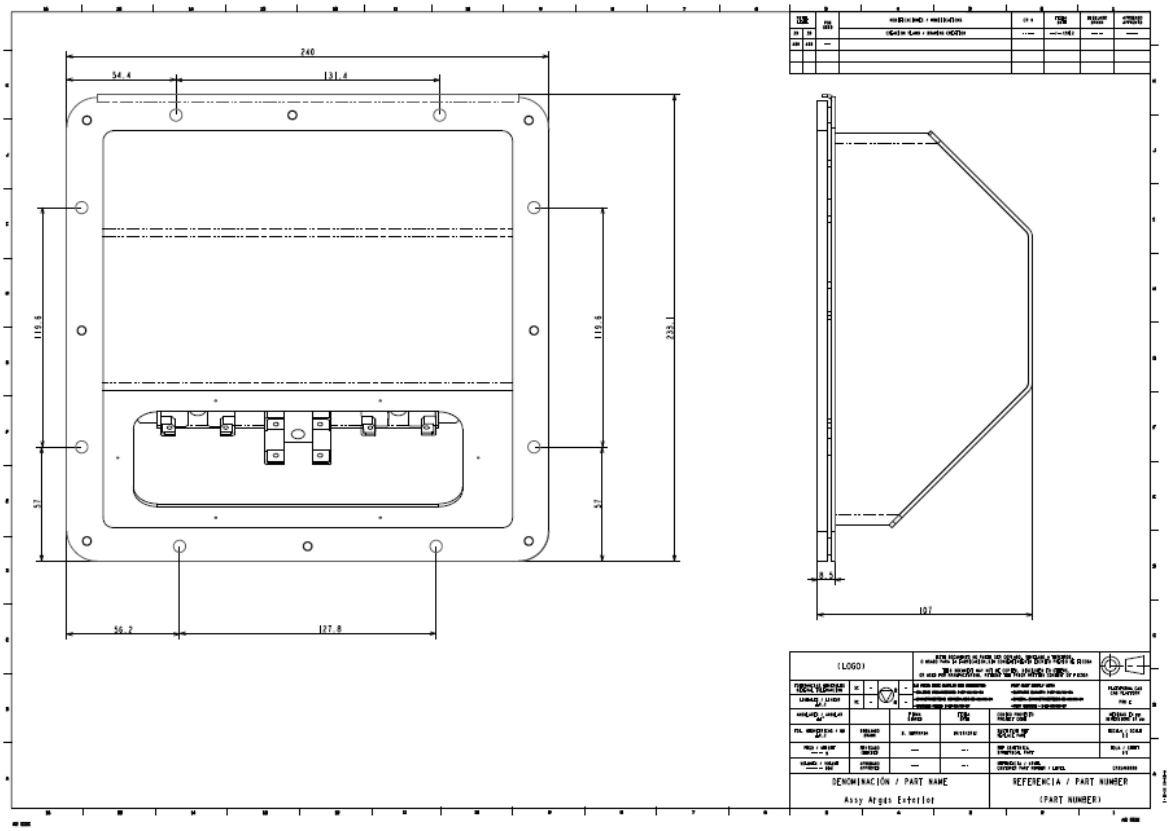


Illustration 9: Secon shot of camera unit blueprint.

6.8 Camera configuration

The camera firmware allows for a number of parameters to be configured during use.

In this application, every time the system is powered up, the correct configuration is sent to both cameras by the operating system. An effort has been made to use camera registers in order to prevent saturation or underexposure on the pictures. Both could be detrimental to the process of extracting information.

6.9 Hardware architecture

The aim of the system is to stop the machine from starting if there is an obstacle in the way. The system is initially thought to act upon the machine transmission system preventing it from being engaged.

A fundamental principle that helps compliance during Failure Mode and Effect Analysis is to design the system in such a way that the default response of the system is inherently safe. For instance, train brakes are normally activated, in a way car brakes are not. Only by applying hydraulic pressure to the circuit can brakes be released.

Following this principle, the default for the system would be to disallow train operation unless there is both the certainty that the system is correctly operating and there is no active alarm.

There are two scenarios for the system to disallow operation: either the system is working correctly and detecting an obstacle, or the system self diagnoses itself as unreliable and then the operation is also not permitted.

The software involved in detecting an obstacle using computer vision requires several layers of software that cannot be fundamentally trusted with critical tasks. The software kernel is not real time, there is code being executed from many different sources, and, at the end of the day, using an operating system based structure means that it is very difficult to know at any given time what the system is doing exactly.

To address this issue, again, the stance that has been taken is to default to non operation unless the system specifically indicates otherwise.

From the hardware point of view a dual micro/supervisor system is in place. A STM8S105K6T3 micro acts as the supervisor of the Texas DSP that carries the bulk of the calculations.

All outputs are routed through the supervisor micro, which is in charge of checking that the main micro has started and checked in OK. As soon as communication is lost, the supervisor micro reverts to the safe mode of disallowing operations.

This approach simplifies compliance with CENELEC 50126/128/129 standards, as it is mainly the supervisor micro that needs to ensure its operation under extreme circumstances.

The system is powered by the 220V AC available at the cabin of the train and it is split in two units. The main unit can produce visible and acoustical warnings. It also has configurable outputs whose purpose is to be wired directly into the circuitry of the train. The camera unit is

connected to the main unit using two wires, one to supply power and one to carry serialized image data from the cameras to the main processor.

The system is built around a Texas Instruments DM8148 device. It includes a Cortex A8 processor and a C674X DSP. This system was chosen both because of the high computation capacity and because of the video input/output interfaces.

The chosen optical sensors are Aptina MT9V022. They are 740x480 automotive sensors, withstanding railway temperature requirements. The optical systems are Sunex 2.1mm, allowing for a 135 degree viewing angle.

At the start of the project there were no commercial available modules incorporating the DM8148 processor. It was necessary to develop a custom circuit and layout for the application. The DSP board integrates I/O interface, deserializer and power supply, as well as the main processor and supporting peripherals (DSP board).

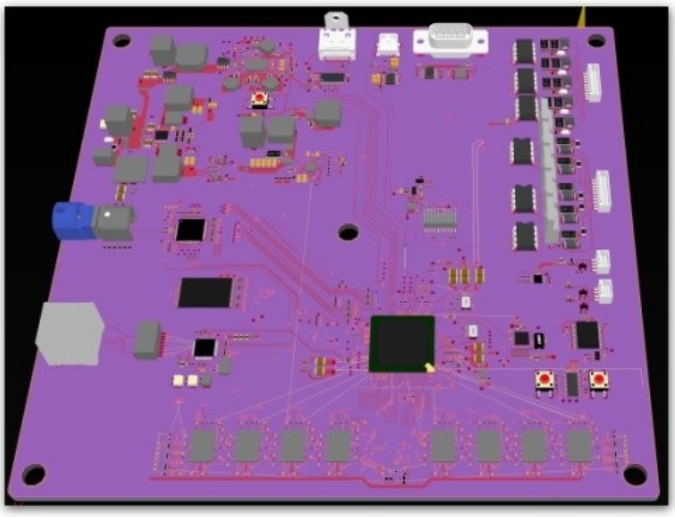


Illustration 10: 3D view of the first trial of DSP board.

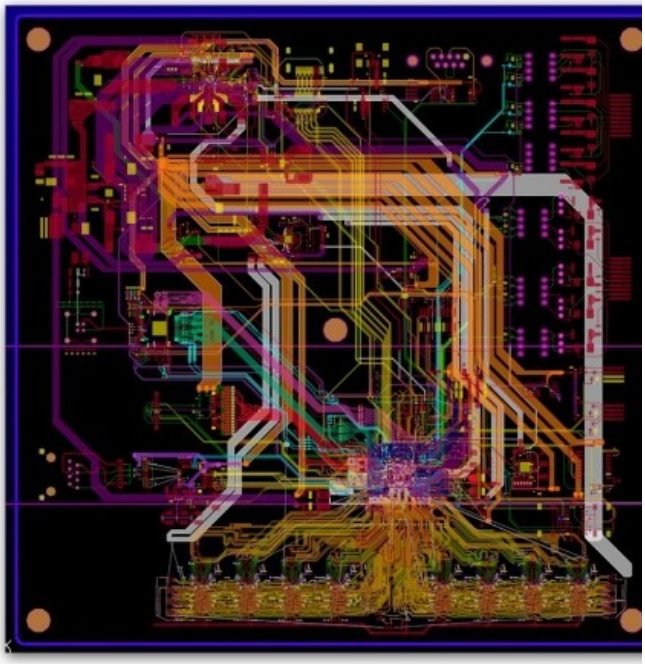


Illustration 11: Image of first trial routing.

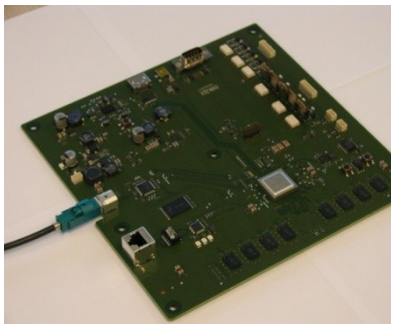


Illustration 12: First trial mounted DSP board.

A second version of the DSP incorporates a small STM8 supervisor micro, a real time clock, a Peltier control and several other changes.

Several cameras were tried in order to get the optimum field of view for the application. The final one is a 2,1 mm Sunex optic with a 135 degrees field of view. Interface electronics were designed for this camera, including serializer.

The first tests with this camera evidenced problems in the acquisition of the video signal by the DSP, originated by software driver discrepancies. This forced the inclusion of a FPGA board next to the cameras in order to adapt the timings.

Illustration 13: FPGA timing investigation

As an important input, the main box has a rugged USB port that is used to input the parameters while on the field.

The hardware allows also for an HDMI monitor to be connected to it. During normal operation it is not in place, but it is very convenient to have while tuning the algorithms to real world conditions as a way to quickly evaluate the algorithm parameters *live* while on the field. With immediate feedback there is no need to go back to the lab as many times just to make sense of the data and perhaps try a different parameter value. Trips back to the lab are only necessary in order to modify the algorithms themselves.

Also it is very nice to have while adjusting installation parameters. A quick visual feedback of the warning and danger areas helps to avoid obvious mistakes during installation.

6.10 Software architecture

To ease development, algorithms have been developed in MATLAB. Later on, they have been ported to C using the MATLAB export tool, and this code has been compiled for the ARM architecture using the standard GCC chain. The scope of the current project is centered around the MATLAB algorithms.

Chapter 7 Stereo vision theory

7.1 Epipolar geometry

Computer stereo vision mimics the depth perception setup found in humans and other animals by using two front facing cameras and combining the images obtained in order to reconstruct the scene and extract useful spatial information.

Computer stereo vision estimates a 3D model from two or more simultaneously taken pictures, using the laws of perspective and the known characteristics and position of a set of cameras.

The simplest stereo vision arrangement uses two cameras. The most basic target of scene reconstruction is to be able to assign depth values to the 2D positions of objects in the images. To do so, we have to establish a correspondence between key features on both pictures whenever we estimate that they originate from the same point in space. This is called the stereo matching problem, and a variety of solutions can be proposed.

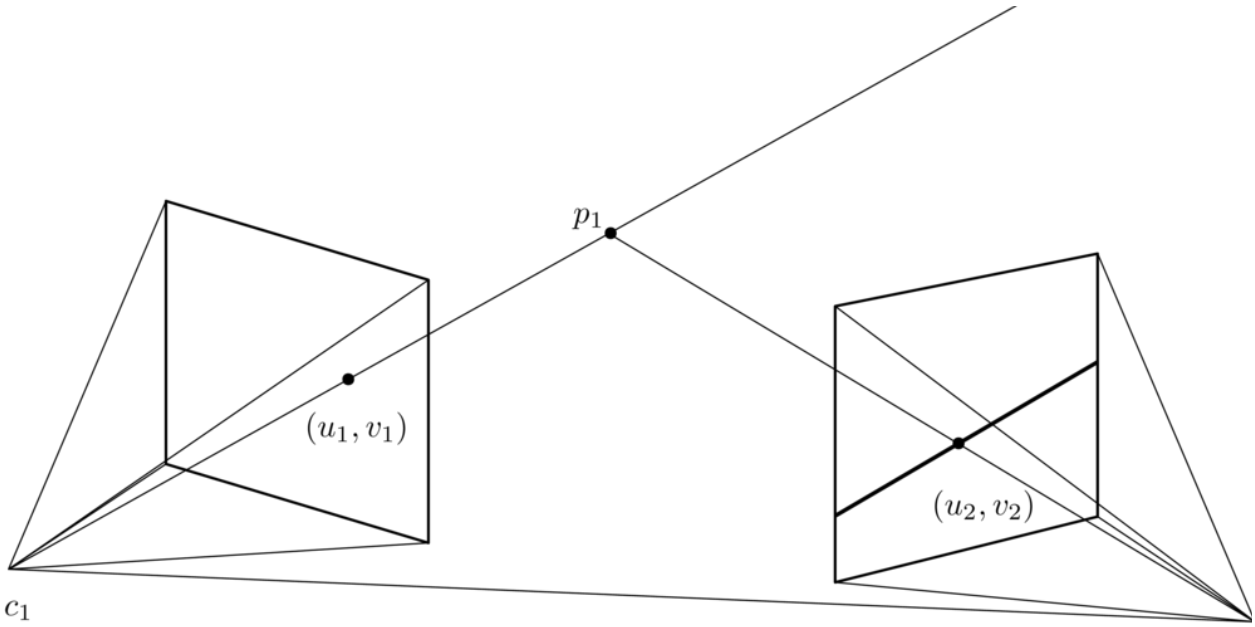


Illustration 14: Search along the epipolar line

Let's consider two cameras that are pointing forward facing a point-like object *p* but that are verged inwards. Both cameras have a projection of the object on their respective optical planes. In the first camera the point is projected at (u_1,v_1) and in the second camera the point is projected at (u_2, v_2). It is easy to see that the point projections actually represent infinite lines where the object could be, between the camera center and infinity.

A plane can be determined using as three defining points the two optical centers of the cameras and the position where the point is at. This plane is called the epipolar plane. The lines where it intersects with the projection planes of the cameras are called epipolar lines.

In order to resolve where a point could be in space we could consider (u_1,v_1) and, given that we know the orientation of the segment between c_1 and the projection (u_1,v_1) and between c_1 and c_2. We can now determine the epipolar plane for the point *p*, because two intersecting lines determine a plane. At this point we can trace the epipolar line on the projection plane of the second camera, and progress along this line trying to determine which (u_2,v_2) corresponds to the point *p.*

Once the coordinates of (u_2,v_2) are known, the actual whereabouts of p can be found out using triangulation.

It is possible to visualize that the set of epipolar lines traced on camera c_2 by the epipolar planes of the points seen by c_1 are convergent towards c_1's position. If c_1 acts as left camera and c_2 as right camera, then the lines on the c_2 plane tend to converge to some point to the left of the visible frame.

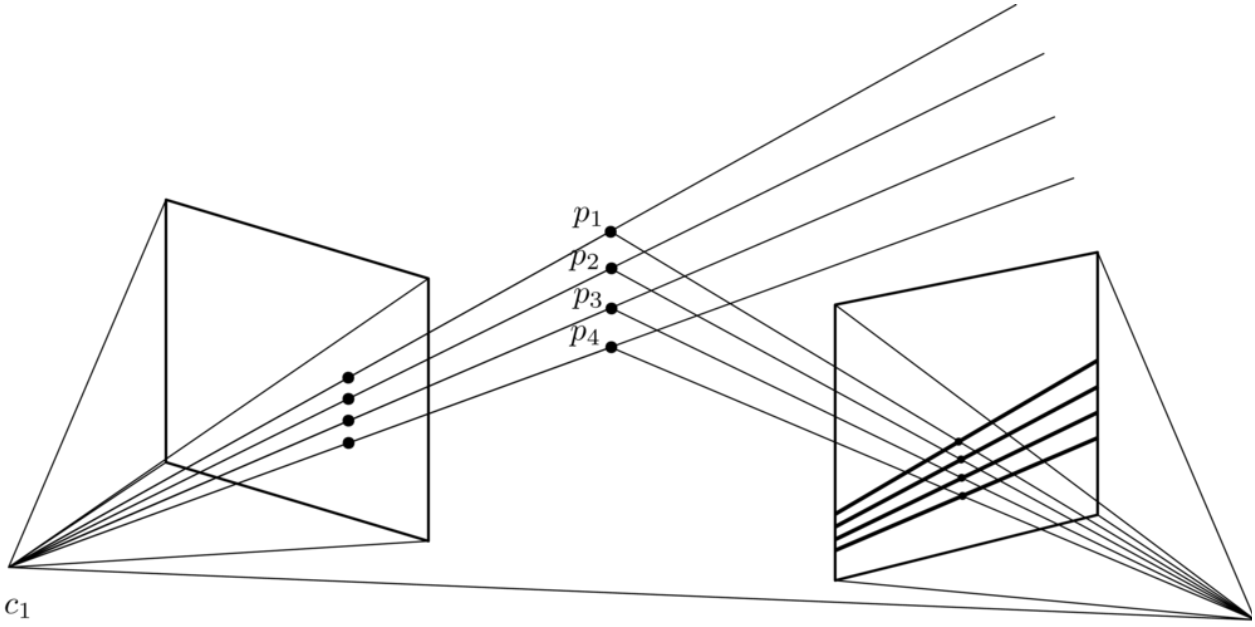


Illustration 15: Epipolar planes

Of course, if c_1 is actually seen by c_2 camera, the epipolar lines will appear to be radiating from c_1's optical center. That may not always be the optimal setup, though.

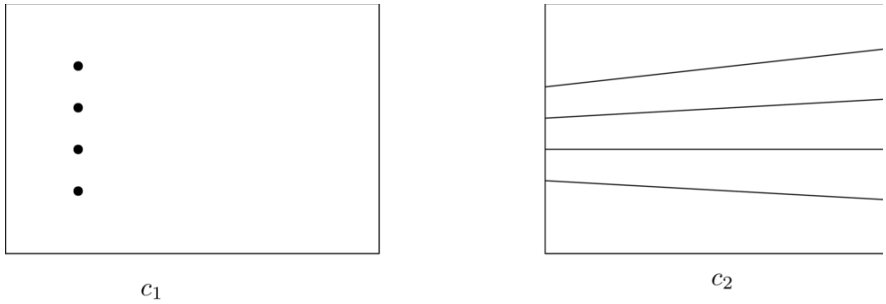


Illustration 16: Each point lies along an epipolar line on the other picture

One way to attack the matching problem would be to compute for any given point on the first picture the epipolar line on the second picture, and then search for the best match along the line. For this to work we would need to input as parameters the precise location and attitude of both cameras relative to one another. Depending on the amount of points to be matched this could have some advantages, but the exploration of the second picture along slanted lines presents some practical difficulties.

Another approach is to place both cameras facing forward perpendicularly to the line that connects the optical centers. Deciding to arrange the cameras like this simplifies two things at the same time. It does make the triangulation necessary to determine the coordinates of *p* becomes trivially simple, and it also makes the search for correlations computationally easier.

Firstly, let's consider again the point *p* on a top down projection of this orthogonal camera setting. In the picture *p* is drawn to the right of the cameras, but the results hold also when the point is in between. Let's call *f* the focal distance and *Z* the distance to the cameras which is the target of our calculations. We have u_1 and u_2 as data, because they are just the *x* coordinates on the projection planes, and we know that from the cameras' geometry.

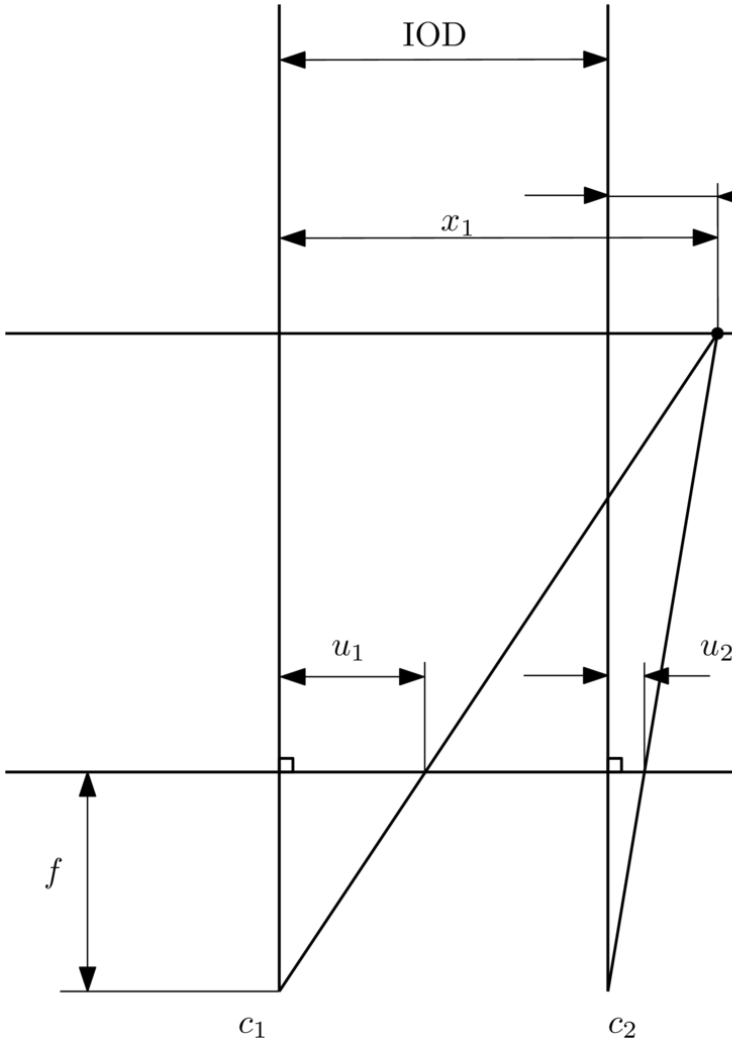


Illustration 17: Orthogonal setup

Also, we have x_1 and x_2 as the horizontal position of p relative to each camera axis. We observe that there are two sets of similar triangles that yield the proportionality equations:

$$\frac{u_1}{f} = \frac{x_1}{f + Z} \qquad \text{(Equation 7.1)}$$

$$\frac{u_2}{f} = \frac{x_2}{f+Z} \qquad \text{(Equation 7.2)}$$

If we subtract the second one from the first one we have

$$\frac{u_1 - u_2}{f} = \frac{x_1 - x_2}{f+Z} \qquad \text{(Equation 7.3)}$$

Now, we can define the difference between u_1 and u_2 as disparity (d)

$$d = |u_1 - u_2| \qquad \text{(Equation 7.4)}$$

Also, we can observe that the difference between X_1 and X_2 remains constant and is equal to the separation between the cameras alongside the line between optical centers. Let's call that the interocular distance (*IOD*).

With the new definitions,

$$\frac{d}{f} = \frac{IOD}{f+Z}$$

Solving for *Z:*

$$Z + f = \frac{IOD \cdot f}{d}$$

Making the assumption that Z is orders of magnitude larger than the focal distance, we find an inverse relation between Z and the disparity.

IOD and f are the product of our choice of cameras and setup, and *d* is easily obtained subtracting the horizontal coordinates of the projection of *p* in both images.

$$Z \approx \frac{IOD \cdot f}{d} \qquad \text{(Equation 7.5)}$$

We will be using this proportional relation from now on.

As evidenced from the formula, smaller disparities correspond to more distant objects. A very distant object, like the moon, yields a disparity of zero in any practical situation. As the objects get closer to the system, larger disparities can be observed.

The second big benefit that stems from this particular camera arrangement is the ease of exploration. Once we have decided to look for a particularly salient feature on the first image, we are confident that we can find it along a horizontal line on the second image:

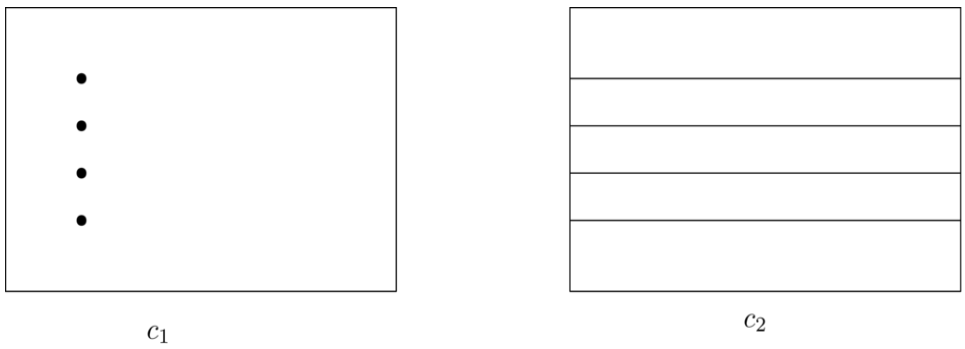


Illustration 18: Epipolar lines become horizontal

This is the case because in this particular disposition the epipolar lines are always horizontal.

In a practical situation, we deal with digitally acquired pictures and make our calculations piping them through a DSP. The fact that we can feed the DSP sections of picture lines instead of having to find our way along a slanted line in search for correlations is a big advantage.

Actually, there is a hidden problem in Illustration 17. This picture assumes that both cameras have exactly the same focal distance, which may not always be the case. The way to adjust the relative focal distances would be to scale one of the rectified pictures to compensate for this.

7.2 Optical correction

An idealized camera obscura is a theoretical optical device that lets in in through a small opening and projects an upside down image of the world outside. Using this kind of picture it is easy to make direct inferences about the geometry and placement of real objects in the scene. In this kind of cameras, lines that are straight in reality are projected also as straight lines.

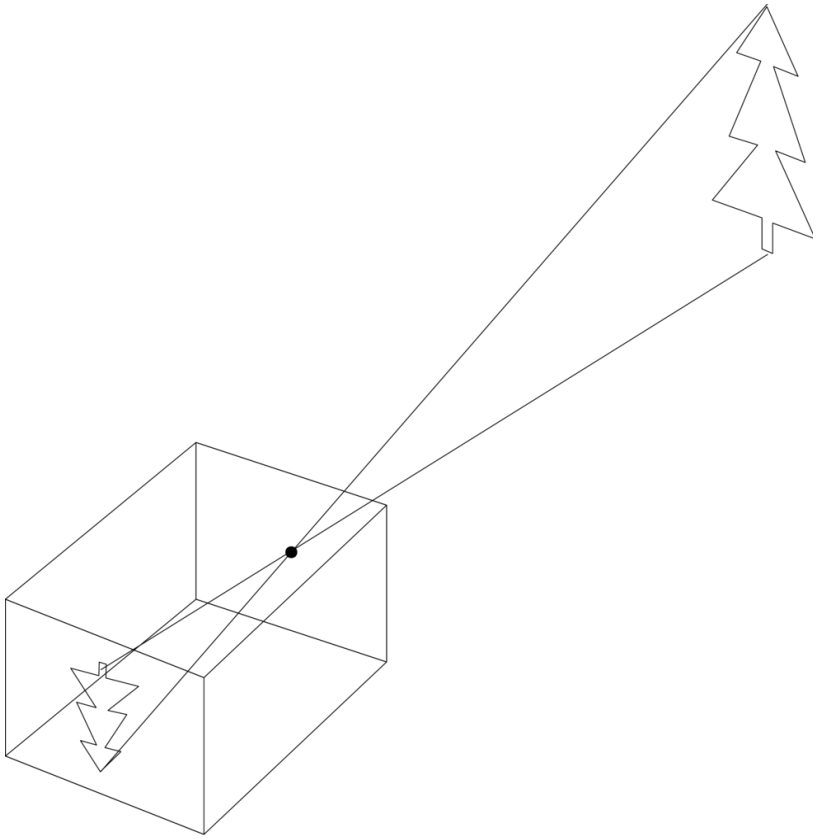


Illustration 19: A camera obscura

The problem with real cameras is that they don't actually work this way for a number of reasons.

1. Lenses are necessary to produce pictures brighter than those that the hole of a pinhole camera would allow. An ideal point-like opening lets in no measurable light.

2. Also, we may want to capture as much field of view as may be necessary for the application without having to imagine an absurdly large CCD.

For rail security purposes, the system needs to have a wide enough angle to cover the ground between the rails and some adjacent areas, specially those at close quarters. The system may often be placed at an angle respect to the horizontal and needs to be able to detect objects as close as close to the camera as possible.

The use of lenses to converge a wide field of view into a tiny CCD causes a most common kind of optical distortion called barrel distortion. It is a type of nonlinear distortion that bends straight lines into a sort of fish eye view picture.

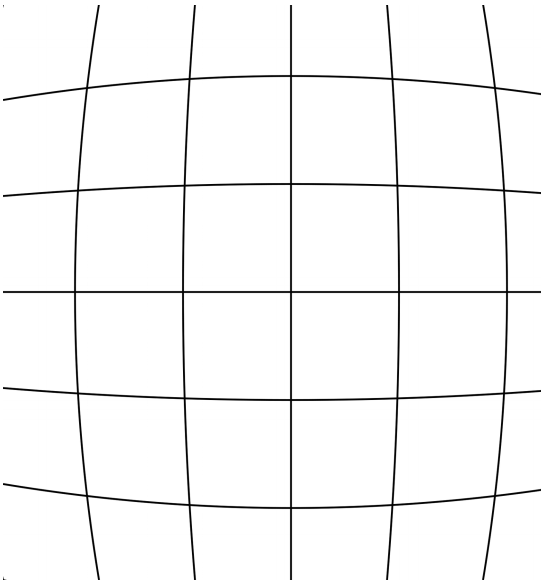


Illustration 20: Barrel distortion

Prior to utilizing the images that result from any cameras it is necessary to correct optical distortions so that the cameras behave, in fact, as linear projection cameras.

We will assume for the rest of the discussion that cameras do behave as sets of focal plane and optical center and pictures are projected onto the focal plane *in front* of the optical center. The choice to project in front and not on the back, mimicking a pinhole camera, is made to allow for an

easier understanding of the geometry and to avoid dealing with negative quantities or reversed pictures.

Looking at Equation 7.5 we can see by dimensional analysis that if the disparity is expressed in pixels, then *f* should be expressed in pixels as well. In a digitally acquired picture it makes little practical sense to address the actual size of the CCD, and the pixel is a more natural measure. We will express the focal distance in pixels of the rectified picture.

From Equation 7.5 we can also see that for a given focal (fixed by the camera) and a given disparity resolution (limited by the processing power investment on interpolation, and ultimately by optical resolution), we can get a larger Z range if IOD gets bigger. It would seem that there is no theoretical limit for this. The counterweight that makes the choice of IOD a compromise is the fact that as the cameras drift apart from each other the projections of the objects become more and more dissimilar, making it harder to correlate between the pictures.

Also, practical considerations of compactness advise against choosing IOD too high, because it would make the camera pair too bulky and cumbersome to install in a practical setting.

Keeping always Equation 7.5 in mind we see that, all things being equal, there is direct proportionality between the focal distance and the estimated distance.

That means that a 5% error estimating the focal distance translates directly in a 5% error measuring actual distances in the field and is a direct hindrance to the performance of the system. Focal distance should be accurately determined using calibration.

One empirical way to find out the focal distance of our camera of choice is to:

1. Place the camera on a wide surface that can be drawn on, and mark its position on it.
2. Draw lines on that surface that appear to be exactly vertical on the rectified camera image.
3. Remove the camera and extend those lines.

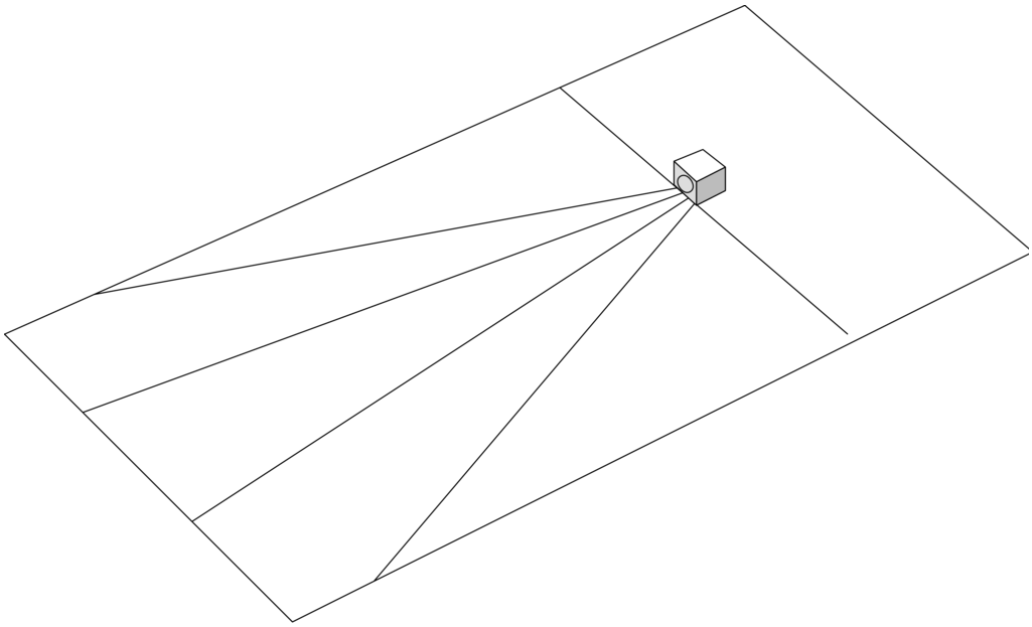


Illustration 21: Finding the optical center

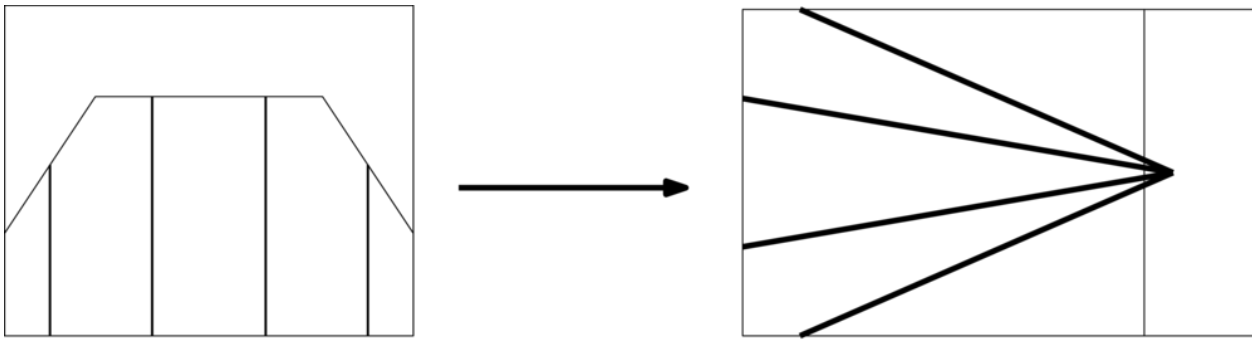


Illustration 22: Lines must appear vertical on the image

Once the focal distance is known we can then easily make a conversion to pixels aiming the camera at a pattern screen placed at a known distance, and measuring how many pixels wide (W_p) is the image of half of that pattern screen on the rectified picture. We know that half of the pattern screen is W mm wide. We plug in the value D adding the distance from the camera to the screen to the estimated focal length in mm

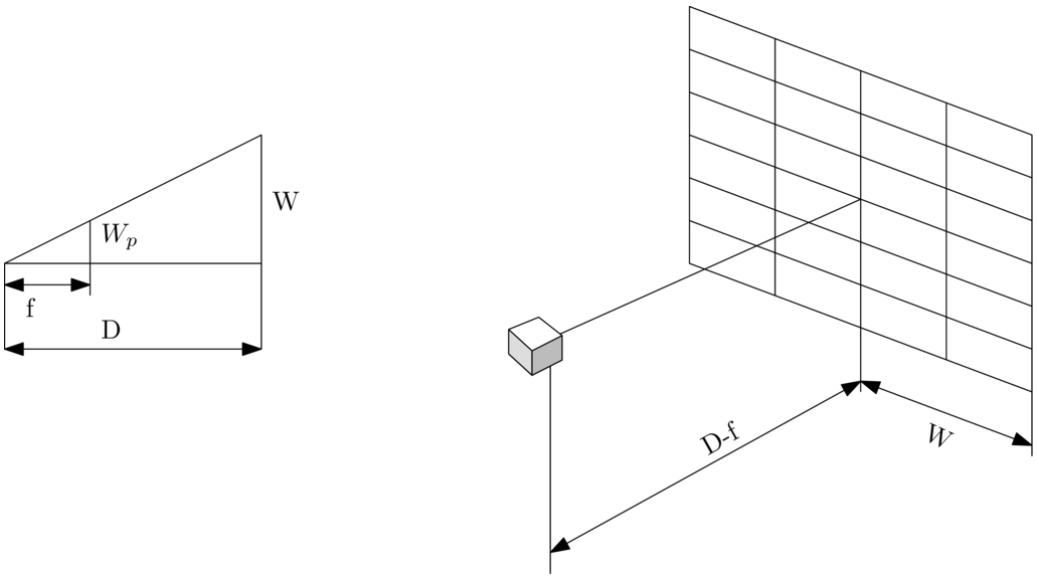


Illustration 23: focal distance conversion

Then we can find the focal length in pixels as

$$f = \frac{W \cdot W_p}{D} \qquad \text{(Equation 7.6)}$$

The dimensions of W and D cancel out and f must have the same dimensions as W_p. That means that f must be expressed in pixels if W_p is expressed in pixels.

Another important optical feature to characterize is the center of the image. That would be the projection of the optical center of the camera onto the focal plane. It often coincides with the center of the CCD, but it need not be the case. It is an important parameter because to perfectly align both pictures we need to make the projection plane of one of the pictures tilt and rotate around this point.

An empirical way to find the optical center of the image would be to use the orthocenter of a vanishing point triangle, as seen on the optical distortion corrected picture of a camera:

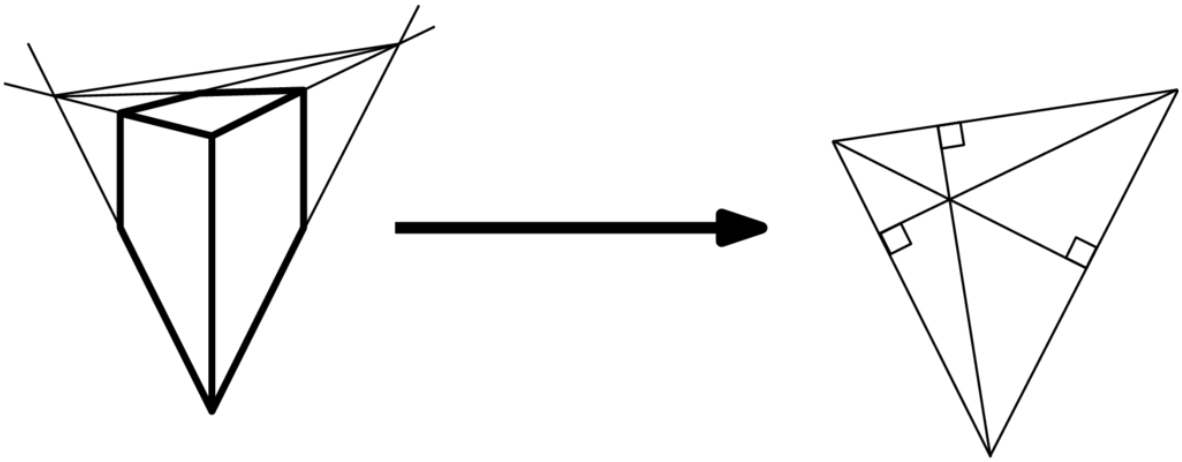


Illustration 24: Finding the center of the image

The lines will converge on the nodal point, which we will note as a characteristic of the camera.

A way to formalize the projection between real word coordinates and picture coordinates expressed by Equation 7.5 is using the matricial form:

$$\begin{bmatrix} u \\ v \\ f \\ f/z \end{bmatrix} = \begin{bmatrix} f/z & 0 & 0 & 0 \\ 0 & f/z & 0 & 0 \\ 0 & 0 & f/z & 0 \\ 0 & 0 & 0 & f/z \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \quad \text{(Equation 7.7)}$$

We are commonly using X Y and Z to refer to real world coordinates, and u, v to refer to picture coordinates. Picture coordinates correspond to pixels, but as we will see using interpolation it is commonplace to use fractions of a pixel. To all extents and purposes, u and v coordinates do represent the camera obscura projection of the real world scene.

The conversion back from picture coordinates to real world coordinates is performed taking advantage of the inverse matrix:

$$\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} z/f & 0 & 0 & 0 \\ 0 & z/f & 0 & 0 \\ 0 & 0 & z/f & 0 \\ 0 & 0 & 0 & z/f \end{bmatrix} \begin{bmatrix} u \\ v \\ f \\ f/z \end{bmatrix} \quad \text{(Equation 7.8)}$$

Chapter 8 System Implementation

8.1 Algorithm

To filter any kind of signal we need to have an idea about what our desired signal looks like. Being the desired signal "people in the danger zone" our set of assumptions is somehow reduced. We cannot yet make sense of the picture to determine what constitutes a person, nor decide whether something is at the appropriate distance or not, because depth of field calculations have not yet been performed.

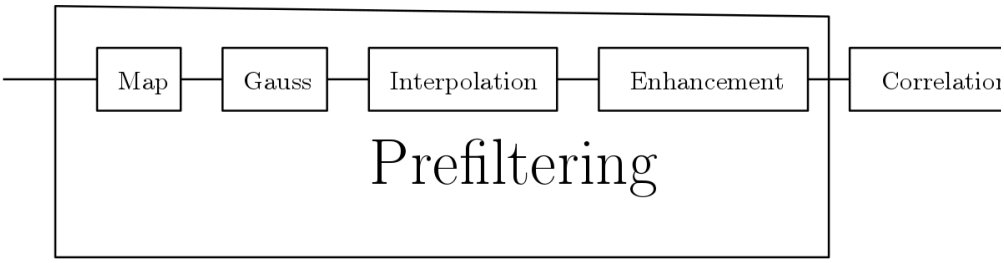


Illustration 25: Prefiltering stages

Prefiltering

We assume at this point that we have already available in memory two images, taken exactly at the same time.

The object of the prefiltering process is to prepare the pictures for processing, it is a signal conditioning of sorts. This involves aligning perfectly the pictures to be able to extract the disparity information, removing noise, and interpolation. Additionally, to prepare the pictures for the selection of relevant features, some kind of enhancement is necessary.

That is, in sequence:

1. Optical correction
2. Low pass filtering
3. Interpolation
4. Picture enhancement

The most important optical correction action is to get rid of lens distortion. The pictures captured by the cameras look nothing like the picture an idealized camera obscura could take. Our choice of cameras has been enforced by the necessity to cover a certain field of view, and that leaves us with pictures that initially have a remarkable fish eye distortion. All of our depth calculations depend on simulating a set of linear projection cameras.

Lens distortion is unique to each camera and is depending on manufacturing process tolerances. It has to be individually corrected for every camera prior to system deployment on the field.

The way to correct the picture is to apply a transformation that maps the distorted pixels of the captured image to the idealized position where they should have appeared in an idealized distortion-less camera.

After the distortion is individually corrected for each camera we still have to deal with the fact that the cameras may not be pointing in the

same direction. It is possible to build a mechanical camera assembly that can keep the cameras in the same position and attitude relative to their casing through time, even considering vibration and some impacts, with very little variation. This has been verified by testing. The tricky part is to be able to assembly the two cameras with a matching orientation with a very reduced tolerance.

This has proven impossible to do mechanically and it has to be assumed that any camera set will happen to be misaligned in by a significant amount.

It is necessary at this time to apply a second transformation that matches the pictures captured by each camera. Stereo vision depends on the successful construction of the cyclopean image.

The map to be applied is the cascade of the distortion and the alignment maps.

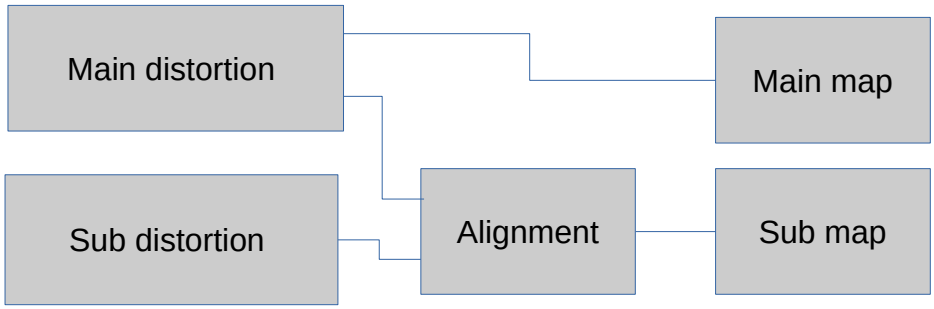
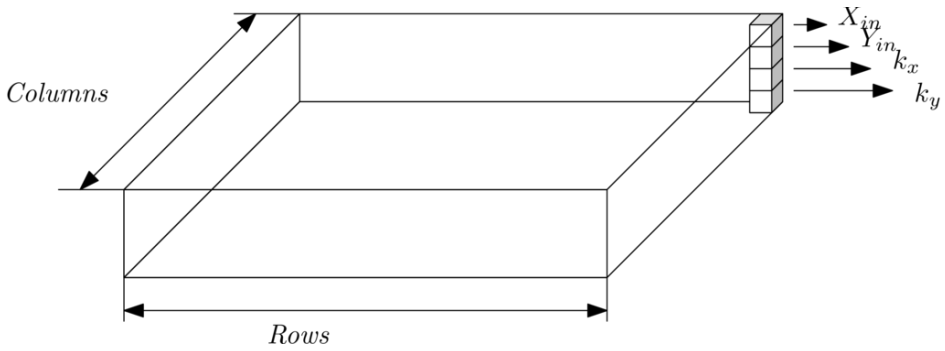


Illustration 26: Optical correction

All of this transformations are condensed in the application of a set of two maps, each between a distorted picture and its rectified and well aligned counterpart.

In the *Map creation* section there is the explanation about how to create such a map.



The output image is made from the original picture using subpixel resolution. In the space comprised between four original picture points that are used as coefficients *(a00, a01, a10, a11)* a number of pixels can be interpolated.

The original picture points to take as a reference are specified in the map structure first and second layers.

$$\begin{aligned} \mathrm{a}_{00} &= originalpicture(m,n) \\ a_{01} &= originalpicture(m+1,n) \\ a_{10} &= originalpicture(m,n+1) \\ a_{11} &= originalpicture(m+1,n+1) \end{aligned}$$

(Equation 8.1)

That is, they are four contiguous points starting at the value that is stored in the map.

a_{00} a_{01}

a_{10} a_{11}

Illustration 27: Four contiguous pixels

The coefficients *kx* and *ky* take a value between 0 and 1. With this premises many interpolation methods can be applied.

We have chosen bilinear interpolation to save processing time. We already profit from the enhanced disparity resolution of the interpolation. Further improvements could be certainly beneficial but on a second level.

$$\mathrm{a}_{00}(1-k_x)(1-k_y)+a_{10}(1-ky)k_x+a_{01}k_y(1-k_x)+a_{11}k_yk_x$$

(Equation 8.2)

After the map is obtained, it must be applied to the captured pictures.

Interpolation

Disparity information is always conveyed in pixels. It belongs in the *uvd* space, where we integrate picture coordinates with the disparity as a measure of the depth associated to each pixel. Disparity information

relates to the real world in proportion to focal distance. As a pixel is a discrete sampling of a picture, and as there are a finite number of them, there is the obvious question of how much distance corresponds to a single pixel disparity difference, and whether that distance is enough for our purpose.

In other words, if we take pixel information as it is we end up with some kind of quantification step that effectively puts a hard limit on our spatial resolution based on CCD density. Actually, it gets worse, because the same pixel absolute difference yields increasingly bigger spatial errors as the distance to the camera grows larger.

Luckily, there is a way to extract subpixel information through interpolation.

In actuality, beyond CCD pixel count, disparity resolution is theoretically limited by the quality of the lens. Before reaching that limit, though, disparity resolution is limited by the amount of cycles we wish to invest in the interpolation method. Sophisticated interpolation methods would yield better results at a higher cost.

A number of interpolation methods can be proposed, each with its virtues and its computational costs. As we are using an orthogonal system and the disparity should align perfectly along the horizontal dimension of the picture, it is justifiable to interpolate only along the horizontal dimension using a one dimensional interpolation.

We have chosen a cubic Hermite spline to fill in the gaps between samples. This kind of interpolation tends to produce smoother pictures than the original.

As we want to limit the impact of noise we don’t want to emphasize high frequency components.

The result of interpolation, in any case, is a picture larger than the original that can be used to determine disparities smaller than one pixel.

Feature detection

Prior to any depth calculation there is the task of choosing the points in 3D space that are good candidates for scene reconstruction. We don't have access to the spatial points themselves, only to two sets of pictures that should reveal the information.

The task is to choose features in the images that are unique enough as to be reasonably sure that the occurrence of a similar enough feature in the other image originates from the same point in space.

There are, as usual, two kinds of errors. The first would be to wrongly match two features that do not, in fact, correspond to the same point in space. That would be a type I error. That would contaminate the input information with spurious points, leading to wasted calculation cycles, and, in the worst case, to a false positive from the whole system.

On the other hand, type II errors are rejecting too many correlations as not valid that actually should be considered valid. A system that rejects too many similar enough features works with too little information and can fail to identify significant volumes, causing a false negative.

The problem of feature matching can actually be divided into two sequential problems. The first one is to find areas on the first picture that are unique enough. The second problem is to match those areas reliably to the corresponding ones in the second picture.

It should be considered carefully which points are considered interesting enough to be picked as candidates. Uniformly lit flat surfaces are obviously poor choices, but what constitutes a good choice?

Highly textured zones could be good candidates, but we can not be entirely sure that an erroneous match can be found somewhere before or after where the correct disparity should appear.

Bright areas are not in themselves good candidates. We cannot trust the absolute value of the brightness because we are dealing with two different cameras, and their particular exposure values at any given time together with the tolerances of their components discourages the use of the direct value of the pixels as the root for comparisons.

The best approach is to start from the next stage and think it backwards. The purpose of feature detection is to select some areas on the first picture to correlate. We will correlate those areas to the second picture and we hope to do it with great accuracy. We would want the value of the cross-correlation to go up and down sharply, having a maximum only around the point where the displacement of one picture respective to the other matches the value of the disparity.

Taking as cross-correlation of two areas of the images a weighted sum of squared differences:

$$E_{WSDD}(u) = \sum_i w(x_i)[I_1(x_i + u) - I_0(x_i)]^2 \qquad \text{(Equation 8.3)}$$

Where I_1 and I_0 are the images to be correlated, *w* is a weighing window and *u* is a displacement vector. The index *i* takes the sum over

every point in the area to be compared. We want E_{WSDD} to decrease quickly for any value of u. That would mean that we have a clear local maximum that delivers a clear disparity value. In a well adjusted orthogonal system we wish E_{WSDD} to behave this way especially in the horizontal direction, and the maximum should be found when the value of u is (disparity, 0).

But, to be clear, what we are doing in stereo vision is correlating two images of the same objects, only from slightly different perspectives. That means that if we wish for the cross-correlation of two features to have a clear local maximum around the disparity value, we would expect the auto-correlation of the first image around the zone where the salient feature is located to have a clear local maximum as well.

$$E_{AC}(\Delta u) = \sum_i w(x_i)[I_0(x_i + \Delta u) - I_0(x_i)]^2 \qquad \text{(Equation 8.4)}$$

The auto-correlation function should have a maximum when Δu equals zero.

Taylor expansion allows to approximate the auto-correlationcan as

$$E_{AC}(\Delta u) \approx \sum_i w(x_i)[\bigtriangledown I_0(x_i) \cdot \Delta u]^2 \qquad \text{(Equation 8.5)}$$

Where

$$\bigtriangledown I_0(x_i) = (\frac{\partial I_0}{\partial x}, \frac{\partial I_0}{\partial y})(x_i)$$

is the gradient of the first image at the point x_i.

The auto-correlation then can be computed as

$$E_{AC}(\Delta u) = \Delta u^T \mathbf{A} \Delta u$$

Where **A** is the auto-correlation matrix, written as

$$\mathbf{A} = w \cdot \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

The fact that the auto-correlation can be written in terms of the gradient justifies that a good way to find auto-correlation function local maximums is to compute the gradient or some variant of it and locate the maximums. As it has been argued, the areas around those maximums are good candidates for the cross-correlation search between pictures.

The gradient is proportional to the local variation of the brightness, and is likely to multiply the presence of noise. Special care has been taken to adjust the camera parameters so that the exposure is adapting to most lighting conditions. But it is impossible to have a completely noise free picture. In order to avoid the creation of artifacts, it is advisable to smooth the image first, using a Gaussian filter.

Once we have the enhanced picture we can use it to locate the best candidate areas to perform the correlation search.

The images have been normalized to $int16$ type. The purpose of this is to use integer math throughout the process, as it leads to faster execution times on the DSP.

Scene reconstruction

The aim of scene reconstruction is to have a computational model of the scene holding the spatial information necessary for volumetric detection. At this point, specific processing can be made to try to make sense of the data and to take appropriate decisions.

Being the system fundamentally a presence detector, what is relevant about the scene is whether or not a large enough object is present inside a defined region.

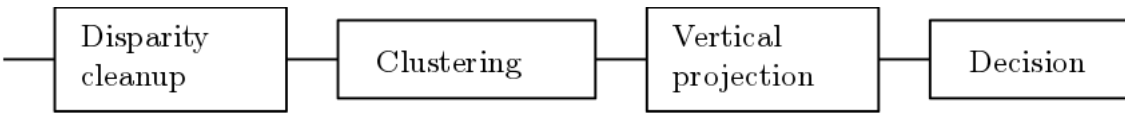


Illustration 28: Scene reconstruction stages

At this stage the data can be represented by a disparity picture made of the points that were found to match between the pictures during the feature extraction stage.

Experimentally it has been found that the disparity picture made from this points is contaminated with unreliable information which takes a form similar to “salt and pepper” noise: many points stand out among their neighbors for having a disparity either too low (dark) or too high (bright) compared to that of their surroundings.

Characteristically, these points appear to be floating in mid air.

In part we can correct this noise appearance by tweaking the correlation thresholds, but there is a tradeoff: as the thresholds become stricter, useful information is destroyed alongside the spurious points.

The source for this kind of mistakes is diverse. Even when care has been taken to adjust camera exposure parameters to avoid saturation, occasionally some pixel areas may display the maximum possible value no matter what. This is specially possible at dawn or dusk, or if the camera catches reflections of the sunlight. The disparity errors created by this situation depend strongly on the correlation method and on the picture prefiltering techniques employed.

Another source of error is specular reflection, typically a natural occurrence when there are pools of water on the ground. If some image could be resolved inside the pool from the camera's viewpoint then there is no easy way to counter that, because the virtual image created by a mirror is as good as the real thing. It would lead to a disparity much higher than the surroundings.

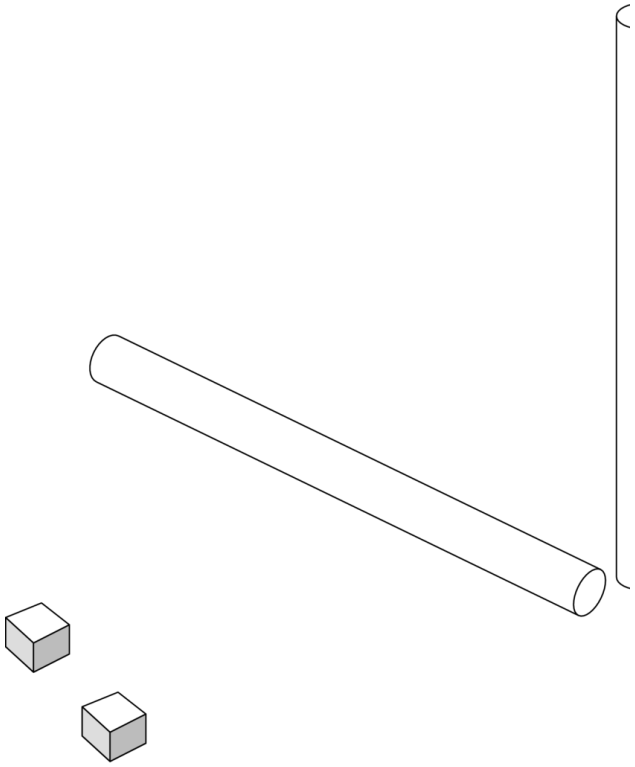


Illustration 29: Crossing illusion

Yet another way to err in disparity appreciation is the illusion of distance created when an object crosses behind another at a different distance.

In Illustration 29 two beams crossing at different distances are shown. From the cameras' point of view, an apparent junction is formed that does not exist in reality:

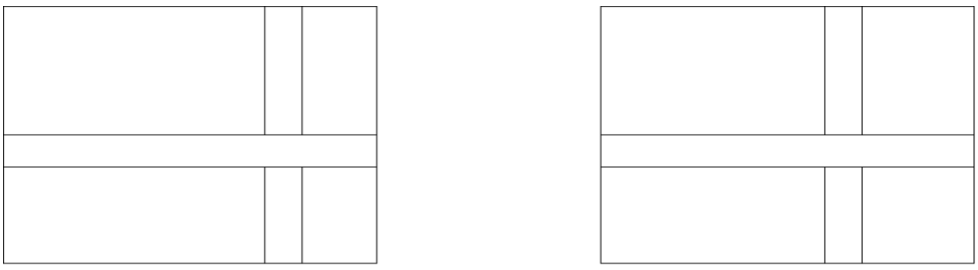


Illustration 30: Crossing illusion view

There is no straightforward way to get rid of this kind of illusion. Furthermore, these points are very rich auto-correlation wise, and will likely appear in the disparity map. This kind of orthogonal crossing is frequent in man-made environments like rail work sites. There is an abundance of poles and beams that can create bad references to base the scene reconstruction off. A general method to clean the disparity is necessary.

The chosen method to clean the picture is specially effective against salt and pepper noise, which is the main annoyance, and has some limited success with other sources of error.

The method is based on the idea that any disparity information piece that is originated from solid existing objects should be parsimonious with at least part of its surroundings.

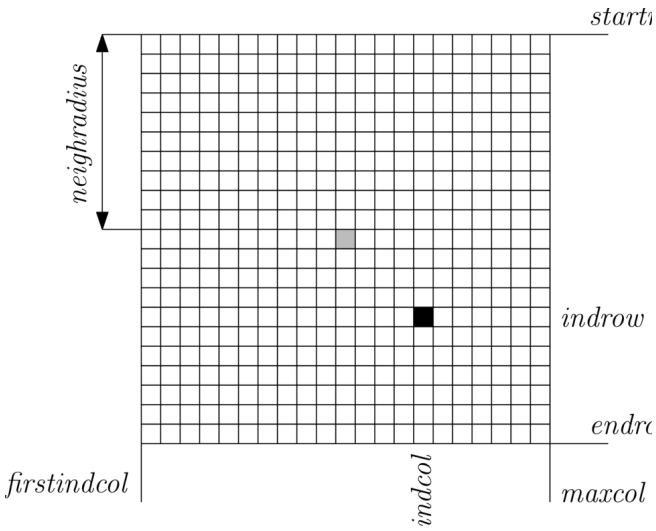


Illustration 31: Disparity neighbors search

We use the disparity point information to validate one another, detecting outliers in their midst. For any given point, a square area is searched for potentially similar points.

The choice of a rectangular area is mandated by computational convenience. If a small enough radius compared to the expected bulk of the detectable object is used then it makes little difference.

The “radius” of the square is an algorithm parameter, as well as the number of acceptable neighbors found. To accept a neighbor’s disparity as close enough, a kind of histogram method is applied.

Clustering

Some computer vision systems can make sense of the cloud of points extracting surface information and at a later stage deciding about which surfaces are part of the same object. Our goal is pretty modest in comparison. The alarm system just needs to extract some kind of volumetric information about the objects that occupy the danger or warning zones.

Nevertheless it is still the need to group the points together in some kind of way, in order to be able to assign more weight to anything we believe to be part of a big enough object.

Let's put in place some basic assumptions that can be made about real world objects that can help tell apart spurious from real data.

1. Objects are made of matter.
2. Matter sticks to itself. Solid bodies are chunks of matter that fill up some contiguous space.
3. Bodies move at finite speed and don't pop in and out of existence.

Making this assumptions allows us to prime our detection strategy in favor of points that correlate well with contiguous points, both in time and space.

On the contrary, if there is some floating point that appears at random and doesn't seem to be connected to any surrounding point it is advisable to demote its importance.

Also, some assumptions can be made regarding data points themselves regarding their position in space.

- The closer a point is to the camera, the higher the danger.
- Far away objects' distance carries a bigger inexactitude
- If a point is so close to the camera that it should be "flying" we can safely disregard it.

The second point is illustrated by the fact that the farther an object is, the less pixel real state on the actual CCD is bound to get as a result of the common laws of perspective.

The third point, related to camera closeness, is very useful in order to alleviate the computational load of the DSP. As correlations are explored on the horizontal axis, putting a limit on the maximum horizontal distance we allow to travel in search of a correlation effectively puts a limit on the minimum distance from the camera at which a match can be found.

While it is true that our most precious data in terms of utility stems from the closest points it is also true that a simple weighing of the points to emphasize their importance is not enough. A balance has to be found.

There are lots of clustering methods that can be used when resources are unlimited. At any rate, computational budget is almost spent in the

making of the disparity picture, so a really simple method had to be found. Clustering algorithms tend to scale very quickly depending on the number of points. The motivation for simplicity also arises from the kind of task the system must carry out. A sophisticated algorithm could bring in more special cases where the system falls prey to its own artifacts. We must not lose track of the original intent of the project.

A proposed algorithm projects all the points onto the XZ plane. Dividing the relevant part of that plane (the danger zones) into tiles, each tile is assigned a weight according to how many points are weighing down on it. Each tile gets activated only if enough "weight" is placed on it. The size of the tile is a configurable parameter.

As a second stage, the number of neighboring tiles could be easily calculated, giving an idea of the size of a given object.

This algorithm has not been tested extensively, and a more primitive method based on continuity along a line was put in place for on the field test units.

Vertical projection

At this point, all acquired nodes are considered valid. There is also important associated information related to each node regarding the reliability of their belonging to an existing object.

There is still a minor hurdle to go through before considering the decision regions. Right until this point all calculations have been done in camera coordinates.

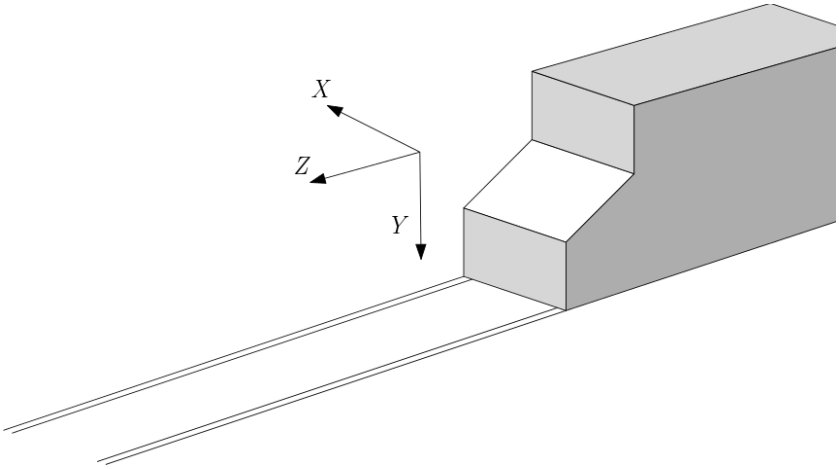


Illustration 32: World coordinates

Once all the points are acquired, it is convenient to translate the coordinates of the points to world coordinates to determine how many fall within the limits defined by the different boxes. In the world coordinate system, X points to the right, Y points down and Z points away from the camera.

Most likely the horizontal plane is at a known location relative to the camera. This relative position can be inferred from the parameters supplied by the final customer during the installation.

Given *p*, a *nx3* stream of points to move and rotate into the correct position, it is easily shown that

$$\begin{bmatrix} q_x \\ q_y \\ q_z \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & move_x \\ 0 & 1 & 0 & move_y \\ 0 & 0 & 1 & move_z \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} \qquad \text{(Equation 8.6)}$$

represents the new coordinates of the point once the origin has shifted *[move$_x$,move$_y$,move$_z$]*.

And applying the usual rotation matrices:

$$[rot_x] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(x_{rot}) & -sin(x_{rot}) & 0 \\ 0 & sin(x_{rot}) & cos(x_{rot}) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{(Equation 8.7)}$$

$$[rot_y] = \begin{bmatrix} cos(y_{rot}) & 0 & sin(y_{rot}) & 0 \\ 0 & 1 & 0 & 0 \\ -sin(y_{rot}) & 0 & cos(y_{rot}) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{(Equation 8.8)}$$

$$[rot_z] = \begin{bmatrix} cos(z_{rot}) & -sin(z_{rot}) & 0 & 0 \\ sin(z_{rot}) & cos(z_{rot}) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{(Equation 8.9)}$$

is a similar process.

The way to obtain the new *qx* coordinates after the origin has shifted by *move$_x$*, *move$_y$* and *move$_z$* and after the axis have rotated by *[x$_{rot}$,y$_{rot}$,z$_{rot}$]* radians is to multiply the movement and rotation matrices in sequence and to apply the resulting transformation matrix to the starting *p* coordinates:

$$\mathrm{q}=\mathrm{rot}_x rot_y rot_z movematrix \cdot p \qquad \text{(Equation 8.10)}$$

Once the point cloud is in world coordinates we can simply ignore the *z* component to determine if any point is within the floor plan of a forbidden area or not, or we can use it to filter out points that are way too high to represent a person.

Decision making

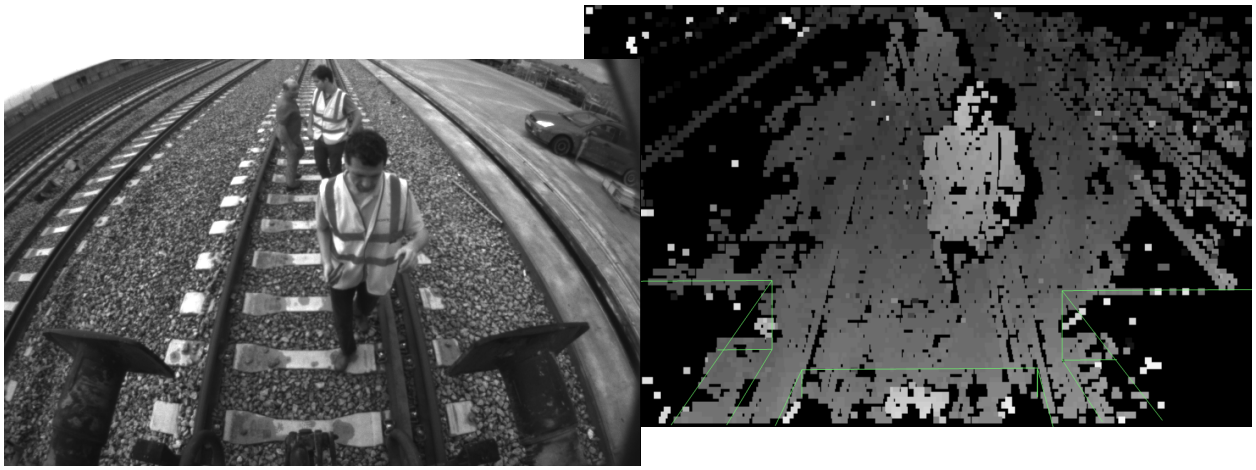
Once the data are acquired, there is still the task to evaluate them in order to take a decision. We have a cloud of points in a 3D region and must devise a way to trip the alarm only when those points originated from a real object fall inside the danger zone, or to trigger a warning if enough points are inside a warning zone.

The points have an associated volume information that is the result of the clustering stage. Weighing each point by its associated volume the decision can be taken more reliably, because experimental tests have shown that as the associated volume increases, what also increases is the likelihood that the point is part of a material object.

Danger and caution zones are defined as per system specifications as orthogonal boxes with sides parallel to the conventional world coordinates. As seen on Illustration 32, Z axis points and down, X axis points sideways and Y axis points forward along the railways.

There is also another kind of box defined for deployment convenience. Often times there are fixed structural parts of the machine that enter the field of view. If this was not taken into account the system would

constantly trigger an alarm because anything belonging to the machine is of course very close to it and that is usually considered the danger zone. To this extent the possibility to define as a parameter a *negative* decision box must be provided. This boxes are used to encase train stoppers or any structural element that has a fixed position relative to the camera and to exclude them from the decision.



One could then, if the system is well adjusted, stand near the stoppers or even between them and trigger the alarm, while the stoppers themselves are consistently ignored.

Once the volumetric node information is in, the amount of node associated volume that falls within each region is calculated and compared to a threshold set by the end user as a parameter.

The state of each box as occupied or vacant is then reported as an output of the complete system, to be taken care of by the electronics of the machine where the system is installed.

8.2 Matlab implementation

The algorithms have been integrated into a wrapper application running under Linux. After profiling the performance to find out which are the most intensive segments, some parts have been rewritten in Matlab for optimization, and some have been ported to DSP code.

This project deals with the devising of the original Matlab algorithms per se, and doesn't delve into the DSP optimization phase.

Prefiltering

There are, as we mentioned, four stages implemented into the prefiltering:

- Optical correction
- Low pass filtering
- Interpolation
- Feature enhancement

As mentioned before, optical correction depends on a map between captured and target pictures. Once we know which origin nodes target which target nodes, the correspondence is established.

The three dimensional matrix is defined in MATLAB as

```
%               mapa(1,x_out,y_out)=x_in
        %               mapa(2,x_out,y_out)=y_in
        %               mapa(3,y_out,x_out)=kx
```

```
            %               mapa(4,y_out,x_out)=ky
```

The function *transforma* is in charge of computing the transformation.

As this is a function computed on a per pixel basis it is quite expensive in terms of computational power. Its final implementation was rewritten for optimal DSP performance as part of a separate project.

```
for j=1:m
    for i=1:n
       if mapa(1,i,j)>-1

          indexx=mapa(1,i,j);
          indexy=mapa(2,i,j);
          a00=imatge(indexx,indexy);
          a01=imatge(indexx,indexy+1);
          a10=imatge(indexx+1,indexy);
          a11=imatge(indexx+1,indexy+1);

          kx=double(mapa(3,i,j))/16384.0;
          ky=double(mapa(4,i,j))/16384.0;

          imatgeout(i,j)=a00*(1-kx)*(1-ky)+a10*(1-ky)*kx+a01*ky*(1-kx)
+a11*ky*kx;

       end

       end
end
```

The coefficients *kx* and *ky* can take values between zero and one and allow for an output picture that may be bigger than the originally captured one.

This coefficients allow us to achieve sub pixel resolution using a bilinear interpolation.

Even though the function *transforma* is very intensive, a lot of the work is saved by using the map as a system parameter. The map is computed once in offline mode, and is tied to a specific set of cameras. The system is updated with the correct map once during installation and there is no need to ever recalculate it.

After this stage we have a set of fully undistorted and aligned frames.

Feature enhancement

The MATLAB version of the filtering definition is quite straightforward. MATLAB has a rich set of tools to define and evaluate filters.

The feature enhancement work is taken care of by a function called *edgedxcomptrasp*. It takes the image as input data and *scale* and *step* as parameters:

```
function [ edge_out ] = edgedxcomptrasp( imatge , scale, step) %#eml
```

Let’s take a sample picture to illustrate the process.

Illustration 33: Original picture

As for the filtering itself, firstly we use a standard gaussian filter on the whole image, to even out the noise:

```
%bgausst=filter2(gauss,imatge);
bgausst=filtre_htal(gaussint,int16(imatge),8);
```

The commented out line is the standard MATLAB version to do this.

The filtering is done instead using the function *filtre_htal* to retain exact control of what is done. An odd size filter is simply swept through the image. Integer math is used to speed up calculations and ensure DSP integer unit compatibility.

```
function [ imatge_out ] = filtre_htal( filtre, imatge, shift )%#eml
%function [ imatge_out ] = filtre_htal( filtre, imatge )
```

```
%   passa un filtre  a la imatge. Introdueix artifacts als
%   extrems de la imatge.
% imatge ha de ser int16, filtre int16. La suma de filtre es tipicament
% 2^15. El resultat es divideix per 2^shift
% Les dimensions del filtre han de ser senars.


[p,o]=size(imatge);
imatge_out=zeros(p,o,'int16');


t=o*p;
[q,q2]=size(filtre);
r=(q-int32(1))/int32(2);
r2=(q2-int32(1))/int32(2);


for i=r2:(o-r2-int32(1))
  %  indexfila=i*int32(p);
    for k=r:(p-r-int32(1))
        acumula=int32(0);
        for j2=0:q2-int32(1)
            indexfila=(i+j2-r2)*int32(p);
            indeximatge=indexfila+k-r+int32(1);
            indexfiltre=j2*q;
            for j=0:q-int32(1)

acumula=acumula+int32(filtre(indexfiltre+j+int32(1)))*int32(imatge(indexima
tge));%int32(imatge(i+1,k+j-r+int32(1)));
             indeximatge=indeximatge+int32(1);
            end
        end
        imatge_out(k+int32(1),i+int32(1))=int16(acumula/int32(2^shift));
```

```
        end
        for k=0:(r-int32(1))
imatge_out(k+int32(1),i+int32(1))=imatge_out(r+int32(1),i+int32(1));
        end
        for k=(p-r):p-int32(1)
            imatge_out(k+int32(1),i+int32(1))=imatge_out((p-r),i+int32(1));
        end
    end
    end
```

The filter is put in place partly because at later stages we take the picture gradient and don't want to emphasize isolated points, and partly to reduce Gaussian noise that can appear in the picture, specially in low light conditions.

Illustration 34: Gaussian filtered

The filtered image is passed to the interpolation block.

The interpolation is taken care of by the function *interpolatrasp*. This function uses the previously mentioned parameters *step* and *scale* to implement a cubic Hermite spline as mentioned in the algorithm section.

```
function [ imatgeout ] = interpolatrasp( imatge, ratio ) %#eml
```

The function returns a picture that is *ratio* times bigger than the original only in the horizontal direction.

Firstly, we define the filter in the *filtre_spline* function.

Starting from the Cubic Hermite Spline matrix:

```
M=[ -0.5 1.5 -1.5 0.5; 1 -2.5 2 -0.5; -0.5 0 0.5 0; 0 1 0 0 ];
```

We construct the filter applying the scale factor:

```
t=[0:1/scale:(1-1/scale)]';
t_exp=[t.^3 t.^2 t t.^0];

filtre=t_exp*M;



filtre_out=zeros(1,scale*4-1);

for i=1:(scale-1)

    filtre_out(i:scale:scale*4)=filtre(scale+1-i,:);
```

```
end

  filtre_out(scale:scale:(scale)*3)=filtre(1,1:3);
```

Next we apply the filter to the image:

```
for i=2:ratio
  k=ratio-i+1;
  imatgetemp(i:ratio:nxsc,1:ny)=imatgeamp(1:nx-1,1:ny)*filtre(k)
+imatgeamp(2:nx,1:ny)*filtre(k+3) + imatgeamp(3:nx+1,1:ny)* filtre(k+6) +
imatgeamp(4:nx+2,1:ny)*filtre(k+9) ;
end
```

And make sure the types are still integer:

```
imatgeout=int16(imatgetemp/int32(2^15));
```

Illustration 35: Interpolated picture

The output picture is interpolated and is now ready for feature enhancement.

For the final edge detection sections there is the need to generate a modified Gaussian filter:

```
temp1=fspecial('gaussian',6,1);
```

```
temp2=temp1(3,:);

dx=conv(temp2,[-1 1]);
```

This stands for the windowed differential Gaussian that was talked about previously. To speed up execution, once the filter is validated, the generation part is commented out and the actual values are hard coded directly into the script:

```
%temp1=fspecial('gaussian',6,1);

%temp2=temp1(3,:);

%dx=conv(temp2,[-1 1]);

dx=[-0.00619310442953756,-0.0395680916268514,-
0.0786306316322448,0,0.0786306316322448,0.0395680916268514,0.00619310442953756]';

dxint=int16(dx*2^15);
```

To do the enhancement we proceed as follows:

Take the first and second derivatives of the image using the modified Gaussian.

```
bgaussdx=filtre_htal(dxint, int16(bgauss),12);


bgaussdxdx=filtre_htal(dxint,bgaussdx,12);
```

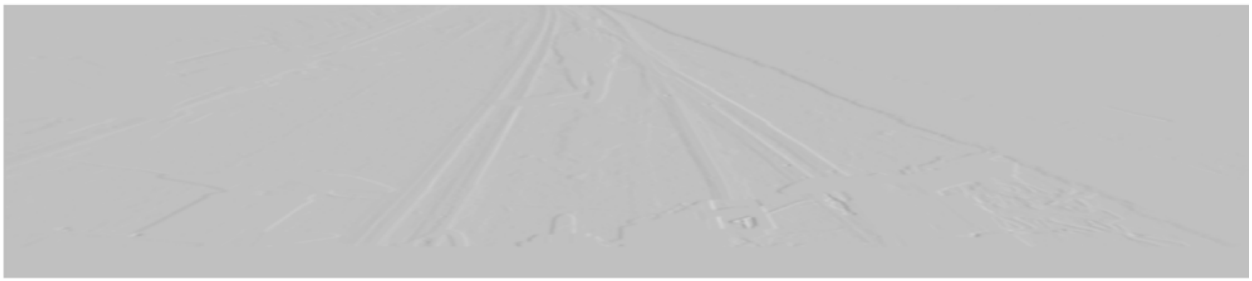


Illustration 36: First derivative

Illustration 37: Second derivative

Calculate a peaking of the absolute value of the first derivative:

```
bgauss3dx=filtre_htal(int16([-1 2 -1])',abs(bgaussdx),1);
```

Generate a binary image that gets value 1 whenever the second derivative is positive, and peak it with a simple peaking filter.

```
cond=bgaussdxdx>0;

edge=filtre_htal(int16([-1 2 -1])',int16(cond),1);
```

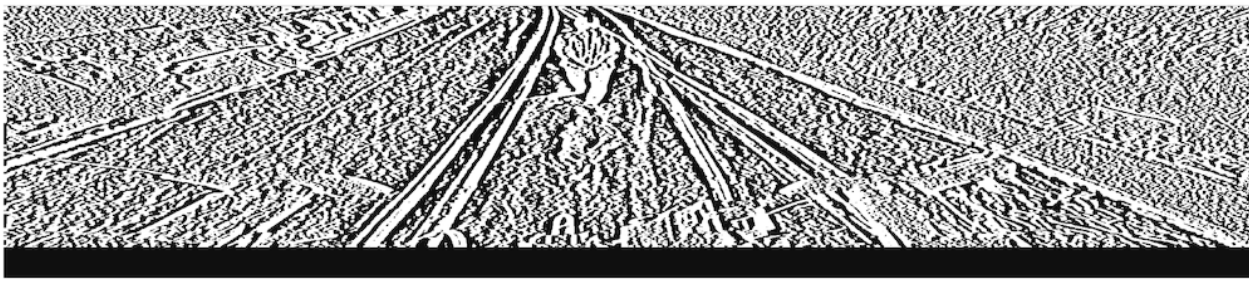


Illustration 38: Cond binary picture

The final edge condition includes the points where this edge condition is met and also the first derivative is changing:

```
edgefinal=edge>0 & bgauss3dx>=0;
```

This filter departs from the canonical filtering mainly because we disregard changes along the vertical direction. We are not interested in those because our orthogonal system can only resolve disparities in the horizontal direction.

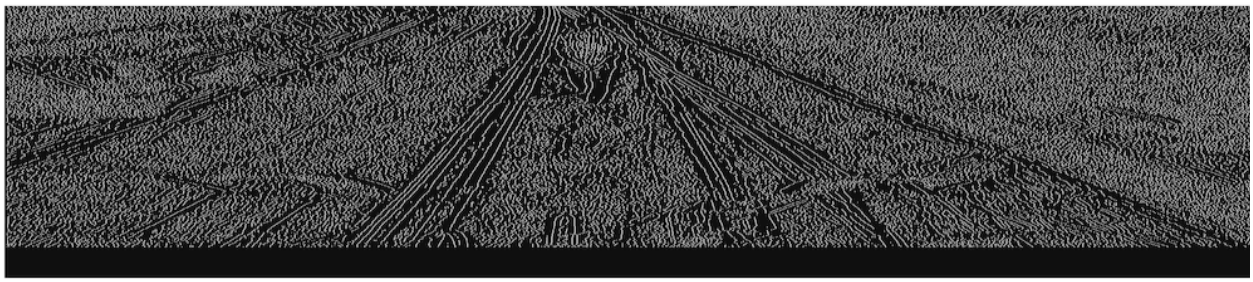


Illustration 39: Final enhancement

The final edge detection assessment is the binary image of the detected edges weighted by the actual value of the first derivative. The derivative is available because it has just been calculated and we don't want to throw away this wealth of information. We assign it as a weight to the edges so that the pixels can carry it. It will become immediately useful when looking for unique candidate points to match between pictures:

```
edge_out=int16(edgefinal).*abs(bgaussdx);
```

Feature detection

When it comes to extract disparity from a set of pictures, the crucial step is to match a detail of the first picture to a detail of the second picture that actually can be said to originate from the same point in space. This calls naturally for some kind of cross-correlation of picture areas.

The exact way in which this correlation is implemented and thresholds are set will define the likelihood of a correlation *hit* actually mapping to a

point in space, and also will affect greatly the computational cost of the whole process and in the end the feasability of the system.

A tradeoff is already in place between correlation accuracy and frame throughput, and the way out of the tradeoff has been testing on site. All key parameters can be tweaked on the field using an external parameter file.

We will begin by scanning the main (enhanced) picture *em* until a relevant feature is found. The quality of being relevant is defined by the condition

```
if em(j,ii)>threshm
```

being *threshm* the first key parameter of the feature extraction. As *threshm* is lowered, more information is fed into the algorithm. As this happens, we reach quickly the point of diminishing returns, because we are requiring more and more processing power to process lower quality information to begin with.

Next, we take a sample cut out of the main picture. The window is a square of odd size centered around the point we just found:

```
a=mscal(j-sizecor*scale:scale:j+sizecor*scale,y-sizecor:y+sizecor);
```

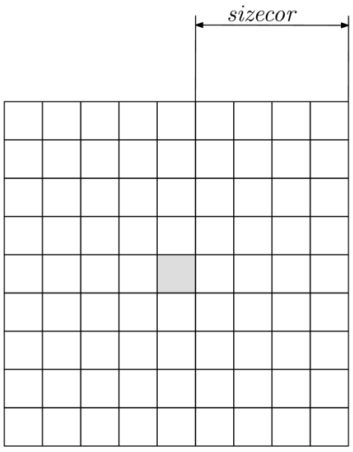


Illustration 40: Picture sample window

This small chunk of picture has to be matched as closely as possible to a chunk on the second picture. The difference in *x* coordinate will be the disparity. If, for a given disparity, the correlation is high enough, we will assume that the disparity is actually matching the depth of the object in space.

After several trials it was decided that the best compromise for the value of *sizecor* was 3, so the effective size of the square is 9x9, that is, 81 pixels.

It is important to get to know this particular square, so we characterize it by calculating its energy and AC and DC components:

```
a32=int32(a);
a32prod=a32.*a32; %_ each pixel squared
```

```
   ener_a=double(suma81_int32(a32prod));  %_ we use cumulative sum for
speed

   continua=suma81_int32((a32));

   factorcontinua=int32(9*9); %_ matching pixel number

   alterna=((a32)*factorcontinua-continua)/factorcontinua;
```

The ratio of AC energy to the total energy is calculated, and will be like a fingerprint for the sample:

```
   prodalterna=(alterna).7*(alterna); %_ AC squared

   enera_alterna=double(suma81_int32(prodalterna));

   ratioalterna=enera_alterna/ener_a;

   f=1-(1-thresh_corr)*(1-ratioalterna)*dc_reject;
```

The *f* factor is also calculated, and will be later used to penalize the cross-correlation of those windows that are, in relative terms, composed mostly of a DC component. That is to say that we want to prioritize the cross-correlations resulting from windows that show a lot of variation *within* the window over cross-correlations that may score high just because the window contains a lot of energy but this energy comes mainly from the background brightness of a DC component.

In other words, DC is featureless and identical to itself anywhere, and should be avoided.

Once we have chosen and examined a square window of the main picture it is time to scan the sub picture for matching candidates.

```
   for k=(j+disparitatmin):(j+disparitatmax) %_ scan right

       if   k>sizecor*scale+1 && k<(size(em,1)-sizecor*scale) &&
es(k,ii)>threshs
```

The above stopping condition can be expressed as:

- Start with a full window
- Don’t hit the right edge
- Find a bright enough point

The first two conditions are in place to avoid artifacts on the picture edges, the third one is analogous to the condition we used to choose the main picture square: a bright enough point found at the prefiltered picture. Notice the introduction of a second threshold parameter, *threshs*.

We scan right, of course, because all possible candidates lie to the right. Negative disparities have no base in an orthogonal system.

We limit the range of disparity to save valuable processor time. The upper disparity limit, *disparitatmax,* is linked to a distance that is too close to the camera for any practical purpose. The lower disparity limit, *disparitatmin,* is ultimately conditioned by the accuracy of the optical alignment and the optical resolution of the cameras.

Instead of choosing the first candidate that scores above a threshold, we scan horizontally for the ideal window around a candidate *disparitatmax-disparitatmin* times, and construct a *numcandidatsx2* matrix reporting the findings.

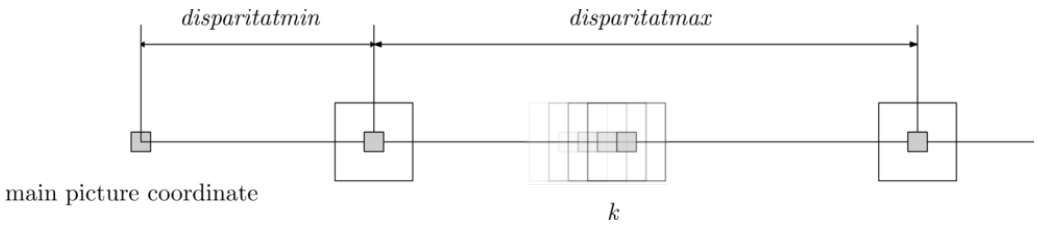


Illustration 41: Scanning the sub picture

The matrix has each time as many rows as viable candidates found (up to *disparitatmax-disparitatmin*). The first column holds the *x* coordinate of the candidate found, while the second column holds the cross-correlation:

```
candidats(numcandidats,2)=f*correlacio_2(a,b,ener_a,energies_b(k,ii));   (Equation 8.11)
```

The correlation function takes as inputs the two relevant squares (*a* from the main picture and *b* from the sub picture) and the calculated energies of said squares.

The correlation is calculated as a normal cross-correlation:

```
creuatint=suma81_int32(int32(a).*int32(b));

creuat=double(creuatint);

co=creuat*creuat/double(enera2*enerb2);
```

But then the ratio of energies is calculated

```
ratio=enera2/enerb2;
```

And those cross-correlations of squares with too dissimilar energies are penalized:

```
if ratio<0.65
    co=co*(ratio+0.35);
elseif  (1/ratio)<0.65

    co=co*(1/ratio+0.35);
end
```

As can be seen in Equation 8.11 the correlations are further penalized by the previously computed *f* factor to the extent that the contents of the original window "*a*" may consist of a DC component. To sum up, wee take a cross-correlation and penalize for dissimilar energies and excess of DC component.

At the end of this step we have obtained from the second picture a list of possible candidates to match the main picture sample.

We select, to begin with, the candidate that has yielded the highest cross-correlation value (cross-correlation values are listed along the second column):

```
[maxim,index]=max(candidats(1:numcandidats,2));
```

And then we clone the list of candidates and build a second version of the list attenuating points that are too close to the found maximum:

```
candidats2=candidats;
atten=(1-(1-maxim)*1.5);
for t=1:numcandidats
    difind=abs(candidats2(t,1)-k);
```

```
      localarea=30; % zona d'atenuació (escala inclosa)

      if difind< localarea

            candidats2(t,2)=candidats2(t,2)*(atten+difind*(1-
atten)/localarea);

      end

  end
```

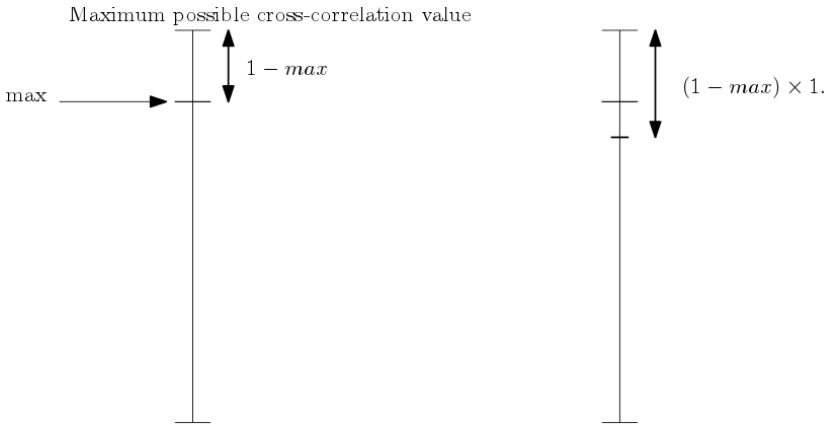


Illustration 42: Maximum point penalization value

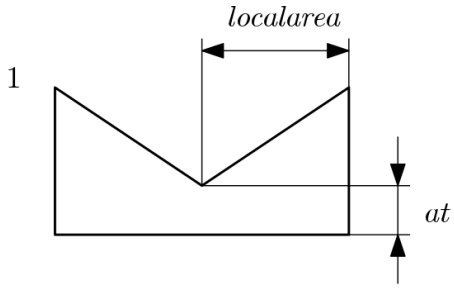


Illustration 43: Attenuation around the maximum

We can see that the value of the cross-correlation of the best candidate so far is penalized by 50% of the distance that separates the cross-correlation value from a perfect matching. Furthermore, the points around a very close local area are also attenuated, and more so if they get really close to the first candidate.

The object of this list refinement is to avoid picking any point that has a second candidate of similar value too close by.

The final step is to select the best candidate based on a final threshold, *thresh_corr*, that will be used to judge over the cross-correlation values.

```
if maxim>thresh_corr    %0.9

      candidats2(index,2)=0;

      [maxim2,index2]=max(candidats2(1:numcandidats,2));

      k2=candidats2(index2,1);

      x=uint16(j/scale);

      if maxim2<(1-(1-maxim)*rej_ratio)  %default rej_ratio=1.5    hit!

            disp(y-sizecor/2:y+sizecor/2,x-sizecor/2:x+sizecor/2)=(k-
j)*ones(sizecor+1);


            dispnodes(:,indcellrow,indcellcol)=[double(j/scale), double(k-
j)/scale, maxim, double(k2-j)/scale , maxim2, 1 ];
```

```
            indcellcol=indcellcol+1;

            j=j+2*scale;

      else

            dispnodes(:,indcellrow,indcellcol)=[double(j/scale), double(k-
j)/scale, maxim, double(k2-j)/scale , maxim2, 0 ];

            indcellcol=indcellcol+1;

      end

end
```

The correlation threshold, as all the previous thresholds, has to be adjusted by experimental testing so as not to flood the algorithm with poorly correlated hits or, on the other end, to throw out too much information out of the system.

The condition to get a second hit can be expressed as getting a next best maximum that is worse enough than the best. We don´t throw away the near hit information and we store it in the *dispnodes* matrix marked as a zero. During the investigation this information has proven its utility in order to diagnose problems under different conditions.

This point marks the end of the intensive calculations. Everything up to this moment has taken as input a whole picture frame, and has performed calculations that require a very high cost in terms of computing power, like filtering, or performing correlations or energy comparison.

The whole algorithm until this point has been the object of a successive optimizations that ended in the writing of parallel DSP code. This project deals only with the original algorithm.

In order to ease the transition into the embedded version the Matlab code is grouped into two big functions.

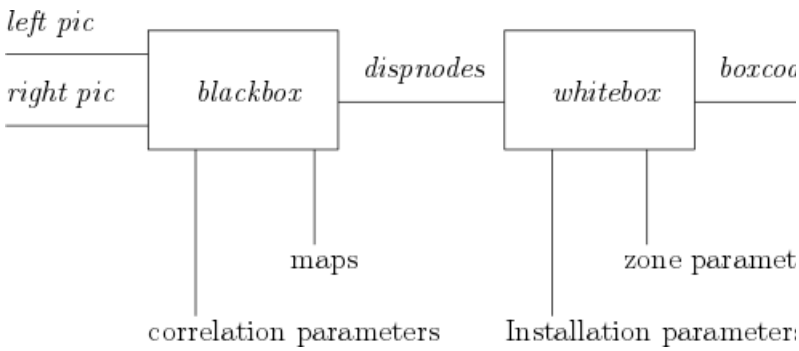


Illustration 44: Code blocks

For convenience we labeled this first block *blackbox* and defined exactly everything that should go in and out prior to the separate optimization.

As can be seen in the block diagram, the input to *blackbox* are the two raw pictures and the output is the node list *dispnodes*. The main parameters of the system are the two correction maps and the various thresholds involved. All of those can be updated into the system after deployment, using a specially prepared USB stick during startup time.

The rest of the algorithm chain was labeled *whitebox.* This block takes as an input the output of *blackbox*. The parameters that *whitebox* takes are related to the physical location and attitude of the camera on the machine and to the kind of danger and warning zones that the final contractor wishes to define for a particular work site situation and how to map these occupation information to a set of physical alarms or triggered actions.

These parameters can also be updated after deployment, and a graphical user interface has been provided in order to facilitate the modification of the parameters as necessitated by circumstances.

The output of this second and final block is just a small binary vector determining which boxes are occupied.

The enveloping Linux system takes care of mapping this occupation information to the appropriate warning and alarm triggers, and tripping the relays that interact with the rail work machine.

From this moment on we have no longer two images and we no longer deal with pixels. We have a list of nodes. The nodes represents a cloud of points in three dimensional space. The number of nodes is orders of magnitude smaller than the number of pixels. Any manipulations that take place in the node realm are cheaper in terms of computational cost than those that take place in the pixel realm, so we can enjoy the relative luxury of employing more computational time per node than we are allowed per pixel, because there are so few of them and they contain the densely packed information we have been able to distil from the scene.

The remaining task now is to make sense of this information and to prepare it for the final decision at the end of the chain.

Scene reconstruction

From now on all the information we need is contained inside the variable *dispnodesmat*. Let´s take a look at its structure.

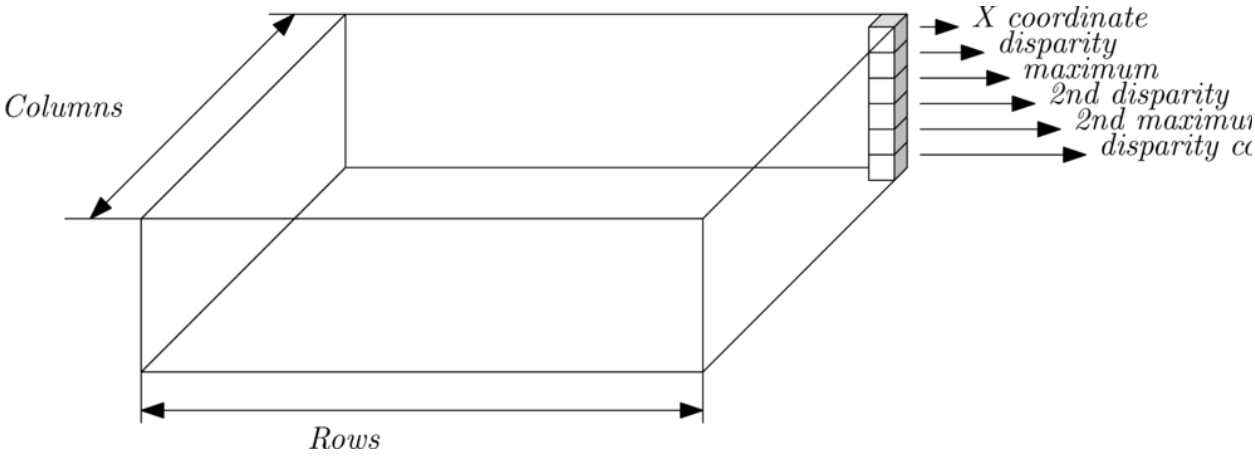


Illustration 45: Dispnodesmat structure

It takes the form of a three dimensional matrix. We can visualize it as a solid prism of *6xrowsxcols* or as a 6 deep stack of *rowxcol* sized pictures. The rows and columns are related to the interpolated picture. Each of the six stacked "pixels" has a distinct meaning:

The first position stores the x coordinate of the main picture. Even if the values are integers, they may represent fractionary pixel values, because we are working from interpolated pictures.

The second position holds the disparity value of the best matched point. It has the same scale and dimensions than the x position, that is, it is expressed in pixels.

The third position holds the cross-correlation value of each disparity point.

The fourth and fifth position are respectively, the disparity value and the cross-correlation value achieved by the second best correlated point on the sub picture. The purpose to hold these data is for algorithm analysis and future refinement.

Lastly, the sixth position contains a value coded to indicate what is the current assessment about the data in the column.

A zero means that there has been no conclusive match. The point should be disregarded.

A one means that this point is considered a disparity node and its information should be considered for scene reconstruction. Higher numbers have been used during development to mark special kinds of wrong matches, like occlusions.

In the best case we could start collecting the disparity information from *dispnodes* and that would be the end of it, but the experimental results have shown that some further refinement is required.

In particular, we want to get rid of the salt and pepper noise as much as possible.

The function in charge of the cleaning is called *goodneighboursmat3*. It takes *dispnodesmat* as data and then a number of different parameters.

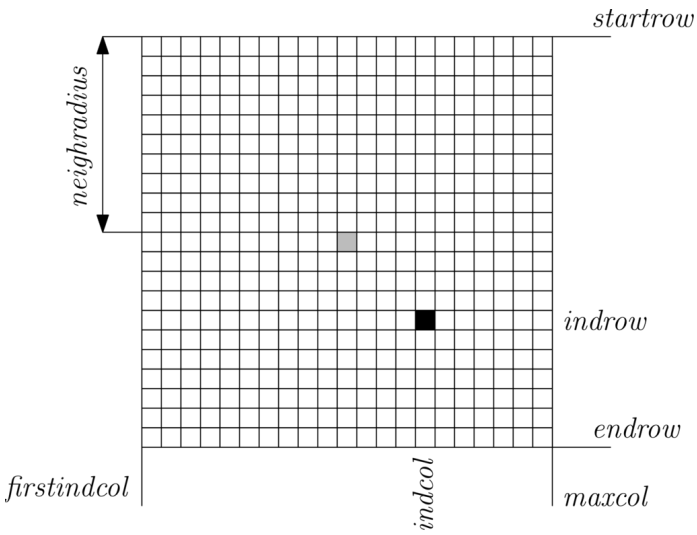
It is called from inside a wrapper function called *gameofdipmat2*. Basically this wrapper function rounds up the disparity nodes and calls *goodneighboursmat3* once for every node.

```
  for indonesx=1:lengthonesx

    [condi,firstindcol] = goodneighboursmat3(dispnodesmat, ycoord, indrow,
     onesx(indonesx), firstindcol, NEIGHRADIUS, MATEMIN, TRIRADIUS,
MINDISP,    MAXDISP);

     if ~isequal(condi,1);%then do away with current point

        dispnodesmat(:,indrow,onesx(indonesx))=[1;0;0;0;0;0];

        bodycount=bodycount+1;
```

```
    end
  end % for ii
```

If the restrictions imposed by *goodneighboursmat3* are not met, the node is scratched off the list.

Looking inside *goodneighboursmat3*, the first step is to explore around the current node in a square of 2**NEIGHRADIUS*+1.



We collect all nodes in this area and add them to a one dimensional vector:

```
  for indrowpossible=startrow:endrow
    indcol=firstindcol; %start at first col
    while (indcol<=maxcol) % filter for nonzero
     currentnode=dispnodesmat(:, indrowpossible, indcol); % filter for
distance
     if (abs(currentnode(1)-xval)<NEIGHRADIUS)&&currentnode(6)==1
```

```
        if (firsttime==1)
          firstindcol=indcol;
          firsttime=0;
        end
        vecdispz(vecdispzsize+1)=dispnodesmat(2,indrowpossible,indcol); %
add to disparity list
        vecdispzsize=vecdispzsize+1;
      end
      indcol=indcol+1;
    end
  end
```

Next we analyze the vector. We will be using a histogram function called *histog*:

```
function [histogram]=histog(vecdisp, scale, mindisp, maxdisp) %#eml
%function [histogram]=histog(vecdisp, scale, mindisp, maxdisp) %#eml
if nargin<3
    maxdisp=max(vecdisp);
    mindisp=min(vecdisp);
end
histogram=zeros(1,(maxdisp-mindisp)*scale+1);
lengthvecdisp=size(vecdisp,2);
for ii=1:lengthvecdisp
    binindex=int16(round((vecdisp(ii)-mindisp)*scale+1));
    histogram(binindex)=histogram(binindex)+1;
end
```

This function, as the name implies, simply tallies up the rounded up nodes and assigns them to a number of different bins according to their disparity.

For every one dimensional collection of disparities we apply the following code:

```
if size(vecdisp,2)>3
    dispvalue=dispnodesmat(2,indrow,indx);
    [histogram]=histog(vecdisp, scale, MINDISP, MAXDISP); %#eml
    mybin=(dispvalue-MINDISP)*scale+1;
    coefs=zeros(size(histogram));
    indtri=1;
    mincorr=0.1;
    dispcorrection=(1-mincorr)*mybin/numbin+mincorr;
    TRIRADIUS=floor(TRIRADIUS*dispcorrection);
    for bincursor=mybin-TRIRADIUS:mybin+TRIRADIUS
        if bincursor>0 && bincursor <=numbin
            coefs(bincursor)=1-abs(TRIRADIUS+1-indtri)/(TRIRADIUS+1);
        end
        indtri=indtri+1;
    end
    condi=sum(histogram.*coefs,2)>MATEMIN;
    condi=all(condi); %type mismatch countermeasure
else
    condi=false;
end
```

What we want is to count just how many nodes with a similar enough disparity are in the immediate surroundings. We apply a triangular shape to the histogram in order to:

1. Include some bins around the one occupied by our node and those with very similar disparity.
2. Give less and less weight as disparity differs.

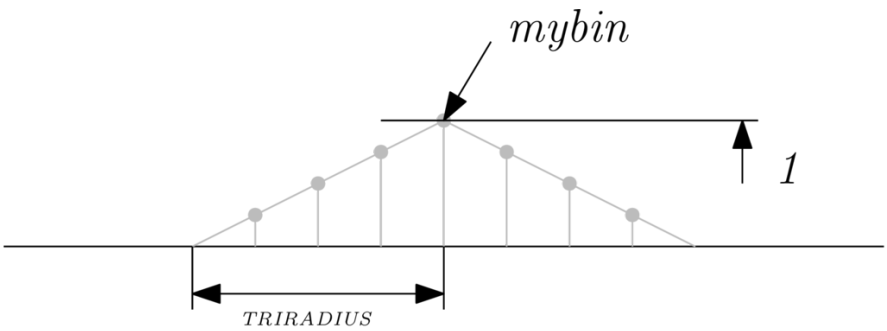


Illustration 46: Weighing of adjacent bins

The triangle gets applied to as many contiguous bins as determined by the parameter *TRIRADIUS*.

As the bins get further away, the number of nodes contained in each of them gets weighed down by a smaller coefficient. If the total sum of nodes exceeds the key parameter *MATEMIN* then we determine that the disparity node we are evaluating is not an outlier compared to its surroundings, and it gets to stay.

If, for whatever reason, there are not enough points in the vicinity to back up the disparity node then its information is removed.

For this method to work correctly, the value of the parameters must be in harmony with the thresholds used during the disparity extraction phase. A loose set of thresholds during feature extraction requires a stricter parameters during scene reconstruction (smaller *TRIRADIUS* and higher *MATEMIN*). Conversely, if the correlation thresholds are very strict, then too little information is coming through and smoothing requirements might be relaxed.

Let’s take a look at what the application of this method does to the disparity information.

Taking this scene as source:

Illustration 47: Adjustment position

We take the disparity map.

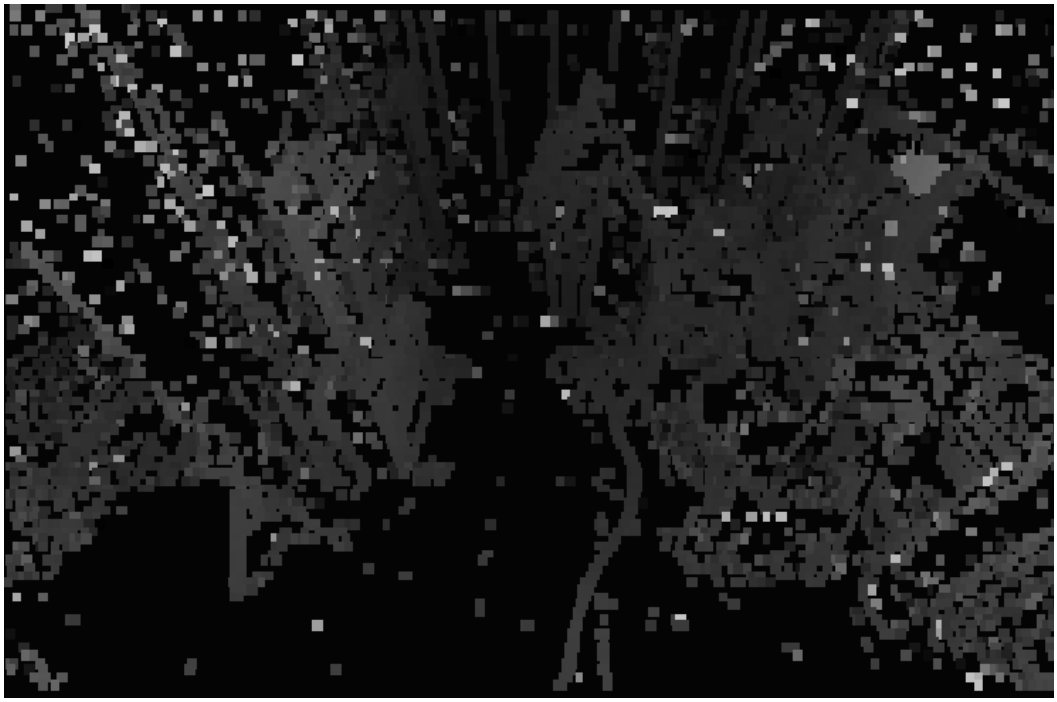


Illustration 48: Original disparity

We can see that he picture is cluttered with salt and pepper noise.

After applying the *gameofdipmat2* algorithm, many spurious points have been pruned out:

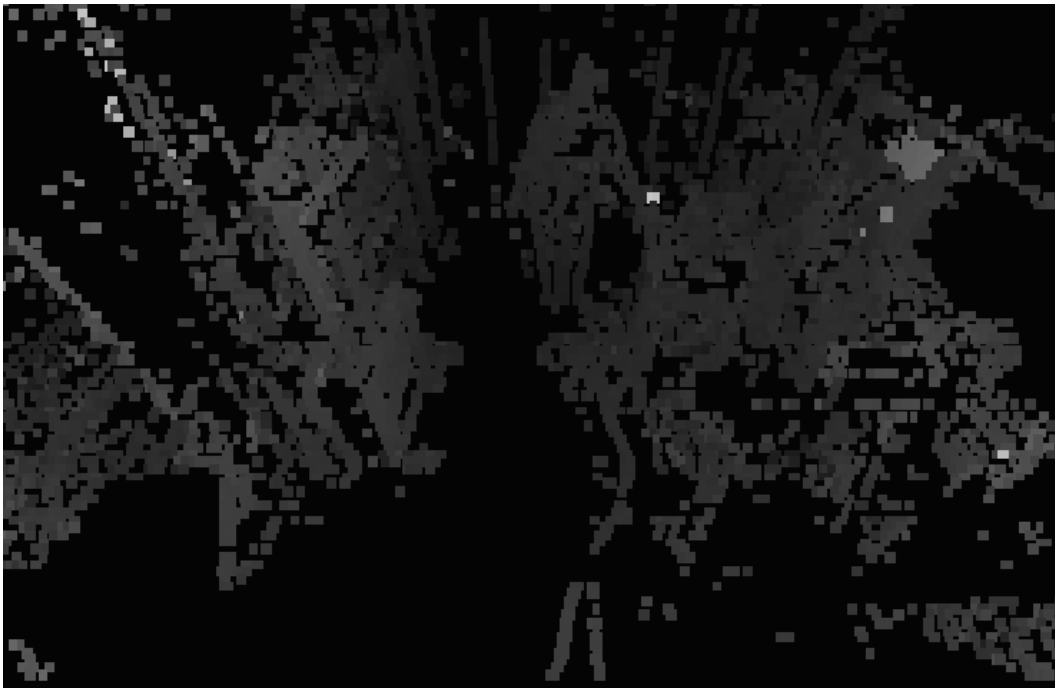


Illustration 49: Cleaned up disparity

There are still some disparity errors, and some good nodes have been eliminated, but the result can be worked with to reconstruct the scene.

A second example, using this source scene:

Illustration 50: Lab picture

This is the original disparity map:

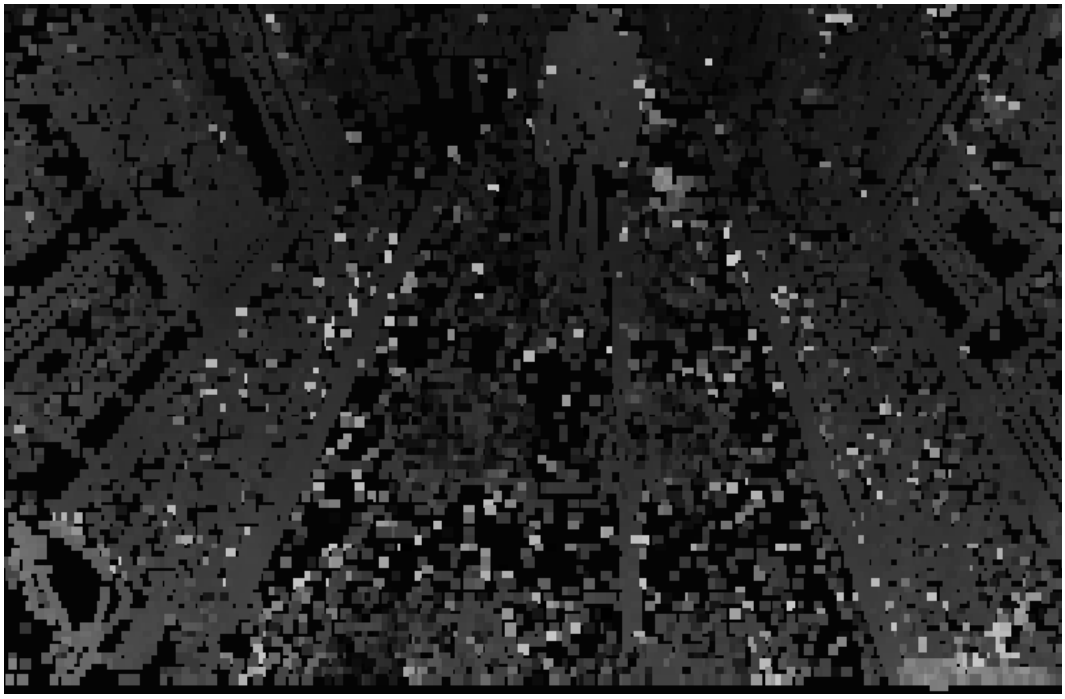


Illustration 51: Original disparity

This is the disparity after cleanup:

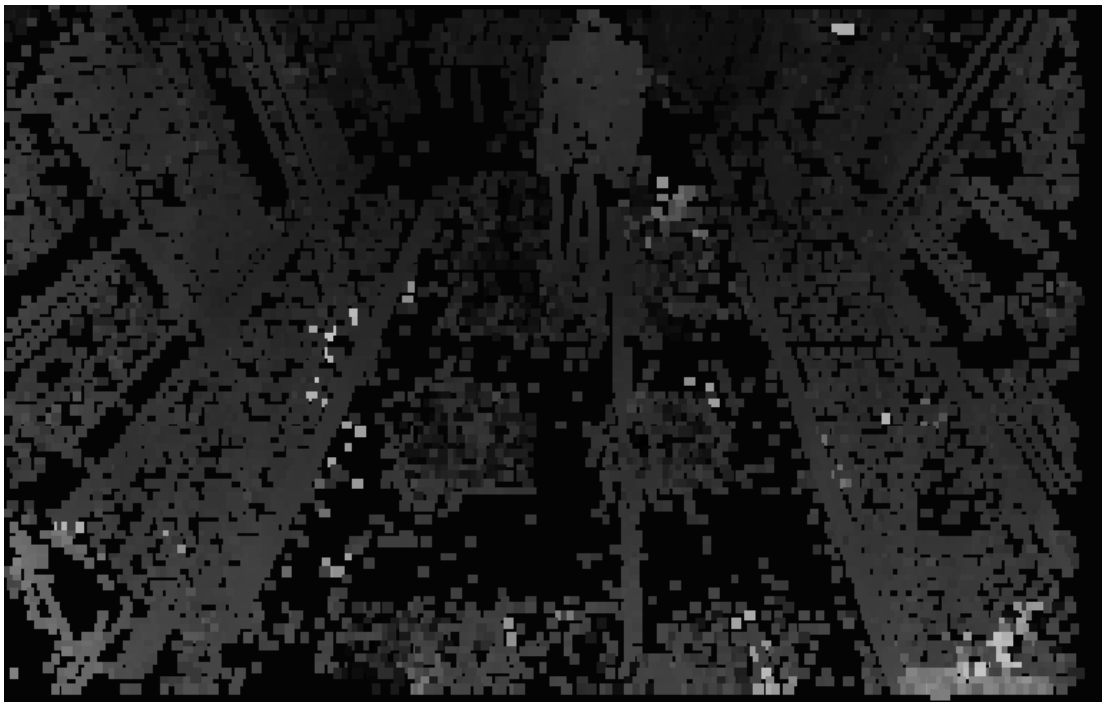


Illustration 52: After cleanup

We can see that in this case the original picture had much less spurious information, and the final picture is also cleaner. It is notable to see that specular reflections on the ground are not possible to eliminate. In a practical setting a similar phenomenon can happen caused by reflections on metallic surfaces or in pools of water. That is a limitation of the system that can only be overcome partially using a correct set of warning and danger zones.

In our field tests this has not been an issue, because the ballast bed is intended to avoid such pools, and metallic surfaces are usually painted over.

Also, a lack of sensitivity to horizontal structures reveals the kind of correlation employed. Even faint vertical markings on the ground are clearly identified, but horizontal lines get ignored, as they can't convey disparity information for our system. This can also be seen on the checkered pattern on the background of the previous example. Only vertical lines are detected.

Clustering

We are confident enough at this point that the collected nodes are reliable. Whatever calculations remain take place at the end of *whitebox*. Even if they seem pretty intensive, their impact is very reduced because there are very few nodes left.

We have specified an algorithm parameter called *objectsize*. This is the minimum size that an object should have in order to be considered.

The first thing to do is to gather all the nodes with a disparity attached to them. Then, determine if every node and the next one along the same row are within *objectsize* distance:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%% distance calculation %%%%%%%%%%%%%%%%%%%%%%

uvd1=[dispnodesmat(1,indrow,onesx(indonesx))-mapparam.centerx
,ycoord(indrow)-mapparam.centery, dispnodesmat(2,indrow,onesx(indonesx))];%
u, v, disparitat

uvd2=[dispnodesmat(1,indrow,onesx(indonesx+1))-mapparam.centerx
,ycoord(indrow)-mapparam.centery,
dispnodesmat(2,indrow,onesx(indonesx+1))];% u, v, disparitat

z1=mapparam.focal*IOD./uvd1(3);

x1=uvd1(1)*z1/mapparam.focal;

y1=uvd1(2)*z1/mapparam.focal;

z2=mapparam.focal*IOD./uvd2(3);
```

```
x2=uvd2(1)*z2/mapparam.focal;

y2=uvd2(2)*z2/mapparam.focal;

dist=(x2-x1)^2+(y2-y1)^2+(z2-z1)^2;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

If they are within that distance, then we will define a horizontal bar between them, and their disparities will be averaged.

If they are not within said distance, their disparities will be kept but as individual short bars.

Illustration 53: Fillbar clustering

This much simplified information will be the basis for our decision.

Vertical projection

This is accomplished using a single function called *moverotxyz*. It uses the camera position and attitude information input by the user at setup time and makes move and rotate transformations in reverse to compensate for the camera typically being at an elevated position and pointing down.

```
for indpoint=1:maxx
    b= (zrotmat*yrotmat*xrotmat)*[xyzstream(indpoint, :), 1]';
    a= (movemat)*b;
    xyzstreamout(indpoint, :)=a(1:3)';
end
```

Decision making

Once all the calculations are done, decision boxes are in place and thresholds are correctly set, decision making is down to an accounting task.

As a tool to tally the points that are relevant to the different decision areas, a function called *boxkillvolmat* is used.

It takes node information and box dimensions as input. The *inout* parameter signals the function whether to count points within or without a given box.

The output is a set of nodes that has been pruned out of those nodes that are not inside (or outside) the given box.

```
function [xyzbarmatout, fillbarvol] = boxkillvolmat(xyzbarmat, boxdim,
inout) %#eml
```

```
  %function [xyzstreamout, fillbarvol,fillbarstreamout] =
boxkillvol(xyzstream, boxdim, inout, fillbarstream)

    % xyzbarmat: an nx4 list of xyz points and bar value

      % boxdim: a vector of box dimensions [xmin, xmax, ymin, ymax, zmin,
zmax]

      % inout: 1--> keep inside points,  0--> keep outside points

      %

      % xyzbarmatout: only the coordinates of those points within/without
the cage

      % fillbarvol: fillbar values of points within or without the cage

    %%%%%%%%%%%%%%%%%%%% PARAMS %%%%%%%%%%%%%%%%%%%%%%%%


XMIN=boxdim(1);XMAX=boxdim(2);YMIN=boxdim(3);YMAX=boxdim(4);ZMIN=boxdim(5);
ZMAX=boxdim(6);

    %%%%%%%%%%%%%%%%%%%% init %%%%%%%%%%%%%%%%%%%%%%%%%%

      a=xyzbarmat;

      insiders = (a(:, 1)<XMAX).*(a(:, 1) > XMIN).*(a(:, 2)<YMAX).* (a(:,
2) > YMIN).* (a(:, 3)<ZMAX).* (a(:, 3) > ZMIN);

      if inout==0

          insiders=ones(size(insiders))-insiders;

      end

      xyzbarmatout=a(find(insiders), :);

      fillbarvol=sum(xyzbarmatout(:,4));


  end %function (boxkill)
```

The function also outputs the total bar volume associated with the nodes that are within the region.

Repeated calls to this function yield the information about the occupation of the different boxes:

```
[xyzmatfoot,fillbarvol0]=boxkillvolmat(xyzmatfoot1,boxdim, 1);

[xyzmatfoot2, fillbarvol1]=boxkillvolmat(xyzmatfoot1,boxdim1, 1);

[xyzmatfoot3, fillbarvol2]=boxkillvolmat(xyzmatfoot1,boxdim2, 1);
```

The final output of the MATLAB section of the algorithm is decided according to a simple threshold test. Any box containing a volume larger than the decided threshold is considered occupied.

A simple algorithm is put in place to produce the output code for the system. The function boxstate is repeteadly called for each box using the volume inside the box as input and the box thresholds as parameters.

```
function [boxstate]=boxstatecalc(currentboxnum, boxstate, thresholdlh,
thresholdhl, vol) %#eml

%function [boxstate]=boxstatecalc(boxstate, thresholdlh, thresholdhl,
vol)

    if boxstate(currentboxnum)=='0' && vol > thresholdlh

        boxstate(currentboxnum)='1';

    end

    if boxstate(currentboxnum)=='1' && vol < thresholdhl

        boxstate(currentboxnum)='0';

    end
```

In the current implementation the system outputs the code of the occupied box with higher priority. That is, the warning signals are not output if the danger zone is lit up, unless they are the same.

```
for boxcounter=boxnum:-1:1

  if boxstate(boxcounter)=='1'

      boxparam(boxcounter).visible=1;

      if biggestcounter==0

          biggestcounter=boxcounter;
```

```
        end % if biggestcounter
    else % if boxstate
        boxparam(boxcounter).visible=0;
    end % if boxstate
end %for boxcounter
boxcode=300+biggestcounter;
```

The default value is 300, and the value increases as boxes light up.

The alarm and warning system later uses this information to decide whether to trip an alarm or not, and if that's the case, which kind. This is explained in the System deployment section

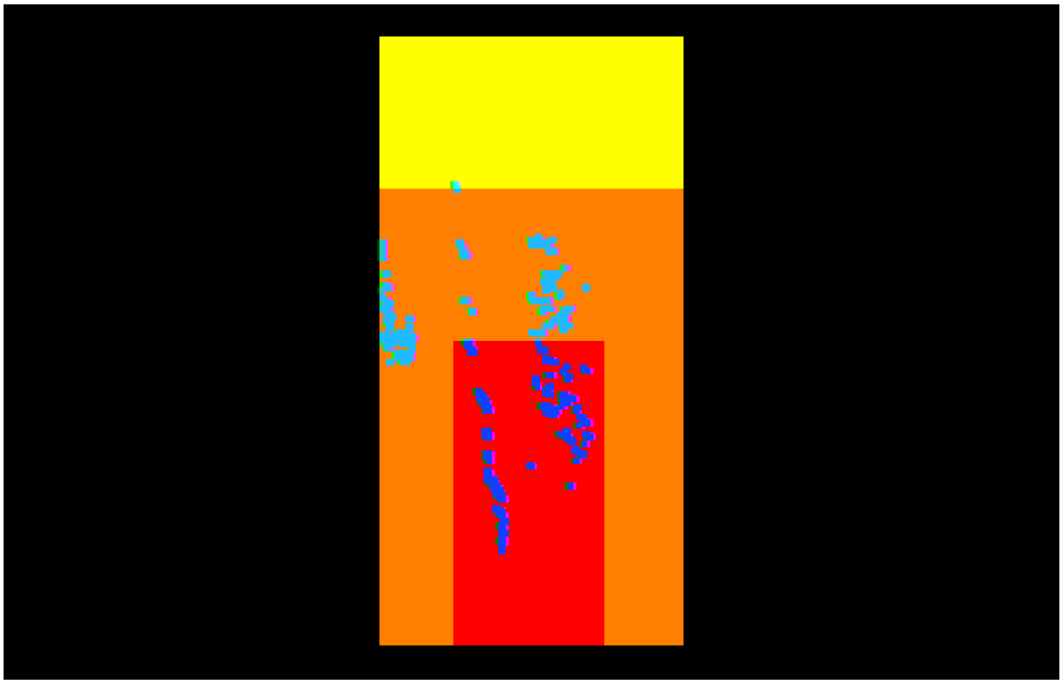


Illustration 54: Top down view of a scene with highlighted areas

In Illustration 54 we can see the video output of the system. This output is only used during installation in order to fine tune the boundaries between areas.

We have used a set of boxes as:

```
boxdim= [-1 1 -3.5 0 0 4];

boxdim1=[-2 2 -3.5 0 0 6];

boxdim2=[-2 2 -3.5 0 0 8];
```

That is, the danger box (red) is two meters wide and goes forward 4 meters. The warning zone (orange) is 4 meters wide and reaches until 6 meters, and there is a wider zone (yellow) that reaches 8 meters and does not give any warning. Each box is lit as objects or people are placed inside, and the volumetric triggers are also set as parameters:

```
volth0hl=10;

volth0lh=13;

volth1hl=10;

volth1lh=13;

volth2hl=10;

volth2lh=13;
```

There are actually two sets of thresholds, *low to high* and *high to low*. This is done in order to implement hysteresis into the system. As in any kind of sharp threshold, there are values that can intermittently trigger the alarm, and that shouldn't be allowed, because it turns the detection system into an annoyance.

Overlaid to the rectangles we can see the detected nodes moving around as people move around, at a rate of around one frame per second.

8.3 Compilation

Matlab is an interpreted language that runs on PC systems. The target application is an embedded system, so the code must first be converted to C and compiled.

Matlab offers sets of toolboxes that provide useful functions to be used in different areas. Often times those functions are also interpreted code, and can be viewed using the command *type*.

Of all the available functions, only a subset is compatible with the *emlmex* compiler. Additionally, some advanced Matlab functionalities like cell types and dynamic indexing, are not compiler supported. Some of the code made extensive use of cell types and had to be rewritten because of this. All variables are fixed size matrices now.

A third type of problem is the existence of matrices that change size during runtime. This is forbidden by the construction of *emlmex*. Although Matlab can usually work with matrices that change size as more values are added to them, the compiler needs fixed size matrices because it allocates memory in advance.

Once the code is successfully translated there is the problem of optimizing the speed. The first tun yielded a processing time of more than one second per frame, and it was clearly unsatisfying.

Dependency tree

This is a hierarchical summary of all the functions that are called during the execution of a detection run.

ROOT : > c:/kstaup/blackbox.m

1 1 : | blackbox 1: c:\kstaup\disparitat2.m

2 2 : | disparitat2 1: c:\kstaup\correlacio_2.m

3 3 : | correlacio_2 1: c:\kstaup\suma81_int32.m

4 4 : . | suma81_int32 0:

5 3 : | edgedxcomptrasp 1: c:\kstaup\filtre_htal.m

6 4 : . | filtre_htal 0:

7 4 : | interpolatrasp 1: c:\kstaup\filtre_spline.m

8 5 : . | filtre_spline 0:

9 3 : . | interpola3 0:

10 2 : . | transforma 0:

ROOT : > c:/kstaup/whitebox.m

1 1 : | whitebox 1: C:\Users\user\Documents\MATLAB\goodrepo\svn2\Matlab\staupbox\utils\uvd2pic.m

2 2 : . | uvd2pic 0:

3 2 : . | xyz2uvd 0:

4 2 : . | abcd2rotdesp 0:

5 2 : . | boxkillvolmat 0:

6 2 : . | centermat 0:

7 2 : . | collectfillbarsmat 0:

8 2 : . | fillbarsmat2pic 0:

9 2 : | findplane 1: c:\kstaup\grabarandompoint.m

10 3 : . | grabarandompoint 0:

11 2 : | gameofdispmat2 1: c:\kstaup\goodneighboursmat3.m

12 3 : | goodneighboursmat3 1: c:\kstaup\histog.m

13 4 : . | histog 0:

14 2 : | moverotxyzmat 1: c:\kstaup\moverotxyz.m

15 3 : . | moverotxyz 0:

16 2 : | topdown8 1: c:\kstaup\rectangulate8.m

17 3 : . | rectangulate8 0:

18 2 : | wireframeoverlay 1: C:\Users\user\Documents\MATLAB\goodrepo\svn2\Matlab\staupbox\utils\xyz2uvd2.m

19 3 : . | xyz2uvd2 0:

20 3 : . | linexy 0:

Profiling

Using the provided profiler in Matlab it is possible to estimate quickly which functions are taking more time.

From the beginning, the integration route was to use *emlmex* to generate the C code and then cross compile this C code on a linux PC for our target architecture using the gcc compiler.

Depending on the criticality of the offending function and the amount of improvement needed, there are several alternatives available:

1. Rewriting the matlab code in order to generate faster code. Usually what runs faster in Matlab translates into C code that also runs faster.
2. Usually the C code generated by the compiler is all but impossible to read, so understanding and optimizing that C code is out of the

question. What can be done instead is replicating the most critical functions in C and trying to optimize those.

3. The heaviest functions are rewritten for the DSP at the lowest level.

Only the first step was addressed by this project. Further improvements were carried out separately. Still, timing the relative execution speed of the functions was the starting point for the optimization.

The code for the systme is split into two blocks:

Blackbox

It is the front end to the system. Its task is to rectify the acquired picture according to the distortion and alignment correction matrix and then calculate the disparity.

Whitebox

Whitebox uses the disparity information computed by the previous stage. It takes that as an input and uses also the physical constraints of the caution and danger zones as parameters.

The first block, *blackbox*, has been the object of a separate line of work concerning optimization. In the end, the code that got implemented into the system is only a distant cousin of the prototype built with Matlab. The second block, whitebox, was shown not to be critical according to the number of cycles consumed, and it is implemented directly by compiling the Matlab code using the emlmex and GCC compilers.

Many reasons could be given for the difference of execution load between the two blocks, but the main reason is simply the volume of data

handled by the two. *Blackbox* deals with pixel data from two pictures. The disparity "picture", on the other hand, is orders of magnitude lighter because it contains only the points that have been found to contain relevant and reliable information about the scene. Another reason is that filtering and exploration algorithms are not performed inside *whitebox*, just the processing of the data we already have.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
test_demostaupmatcomp	1	62.407 s	0.120 s	
blackbox	1	60.278 s	0.001 s	
disparitat2	1	58.491 s	1.570 s	
suma81_int32	13322	33.698 s	14.421 s	
edgedxcomptrasp	2	19.774 s	0.056 s	
filtre_htal	10	19.697 s	19.697 s	
correlacio_2	6172	15.763 s	0.165 s	
intmax	1079082	9.685 s	9.685 s	
intmin	1079082	9.592 s	9.592 s	
interpola3	2	3.256 s	3.256 s	
whitebox	1	1.896 s	0.017 s	
transforma	2	1.786 s	1.786 s	
gameofdispmat2	1	1.648 s	0.035 s	
goodneighboursmat3	550	1.607 s	1.587 s	
uvd2pic	2	0.144 s	0.144 s	
imread	2	0.092 s	0.028 s	
imagesci\private\readbmp	2	0.041 s	0.000 s	
topdown8	1	0.038 s	0.011 s	
ima...\private\readbmpdata>bmpReadData24	2	0.035 s	0.028 s	
imagesci\private\readbmpdata	2	0.035 s	0.000 s	
eml.target	5389	0.028 s	0.028 s	
rectangulate8	1	0.027 s	0.027 s	
wireframeoverlay	1	0.024 s	0.004 s	
imformats	2	0.023 s	0.001 s	
imread>parse_inputs	2	0.023 s	0.000 s	
imformats>find_in_registry	2	0.022 s	0.022 s	
interpolatrasp	2	0.021 s	0.020 s	
linexy	12	0.020 s	0.020 s	
rgb2gray	2	0.019 s	0.002 s	

In the generated profile we can see that the two top level functions take up dramatically dissimilar times to run. *Blackbox* uses 60 seconds while *whitebox* is done in under two.

In chronological order:

1. The map is applied to both pictures by *transforma* in around 2 seconds.
2. The prefiltering done by *edgedxcomptrasp* function takes almost 20 seconds, mostly consumed by *filtre_htal* while computing the different filters involved.
3. The interpolation *interpola3,* which takes place just before disparity extraction, takes up 3 seconds.
4. The cross-correlations performed during the search for matching features take nearly 33 seconds, consumed not by *correlacio_2* but rather by *suma81_int32*. It is notable that the time consumed by the function itself is only 14 seconds, while more than double is consumed. That may be related to the fact that the function is called more than ten thousand times, and the constant loading and unloading of the function stack may account for the overhead.

In contrast, *whitebox* runs in about 2 seconds, most of them taken up by the function gameofdispmat2, which is in charge of suppressing spurious points from the disparity.

Even when a profiling done in MATLAB is no measure of the final performance, it has been used to locate where the most intensive calculations were performed and structure the transition to embedded.

8.4 System deployment

System deployment

The commercial name of the system is Argus, after the giant in Greek mythology that had multiple eyes in order to act as a guardian.

The range of possible applications for the system is quite diverse. Most working paramenters have to be determined on the field and it is necessary to provide the installer with a user friendly tool to adapt the system to any given machine.

To this effect, a Python desktop program has been developed. The sofware presents a series of screens where the user inputs the particulars of the installation, e.g. height, angle and so on. The output of the program is a parameters file called *userparms.conf*.

In order to configure the system with this parameter file it is enough to cold start the system while a specially prepared USB stick that contains, among other things, this parameter file, is inserted in the USB port.

Also as a box option there is the possibility to declare the box as a *stopper container*. The purpose of this feature is to enclose any protruding part of the machine that enters an otherwise regular orthogonal prism danger zone. The points that are caught inside the stopper box are substracted from the cloud prior to any further considerations.

There are two programs that have to be run in sequence on a Windows PC:

1. argusinstall.bat
2. argusapp.bat

A welcoming screen asks for the serial number, machine description, and machine width. This last data will fill in automatically some values in the box definition screens.

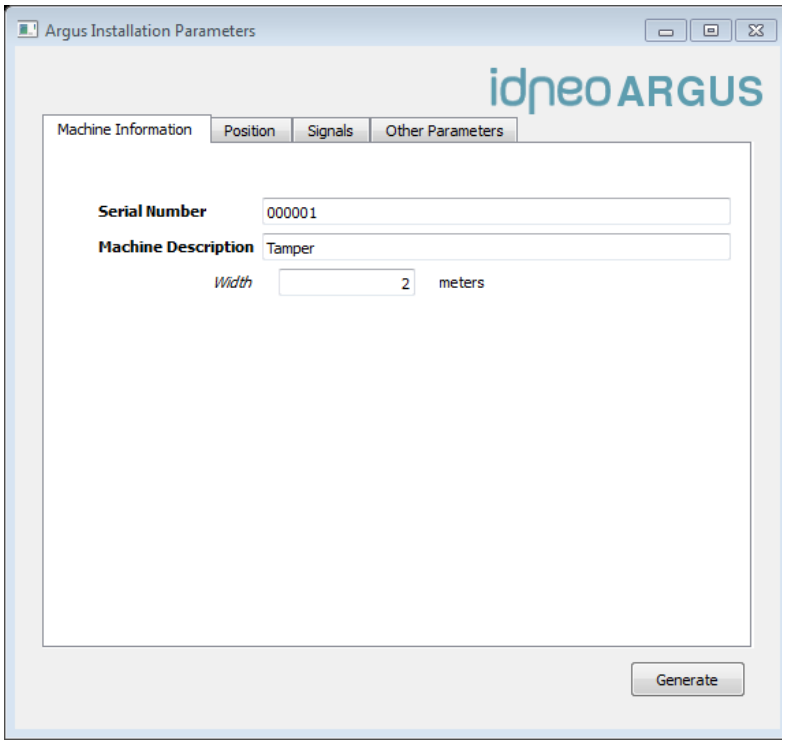


Illustration 55: Machine information

A second screen must be used to fill in the details about the camera placing on the machine. Typical stopper dimensions and separation are filled in by default.

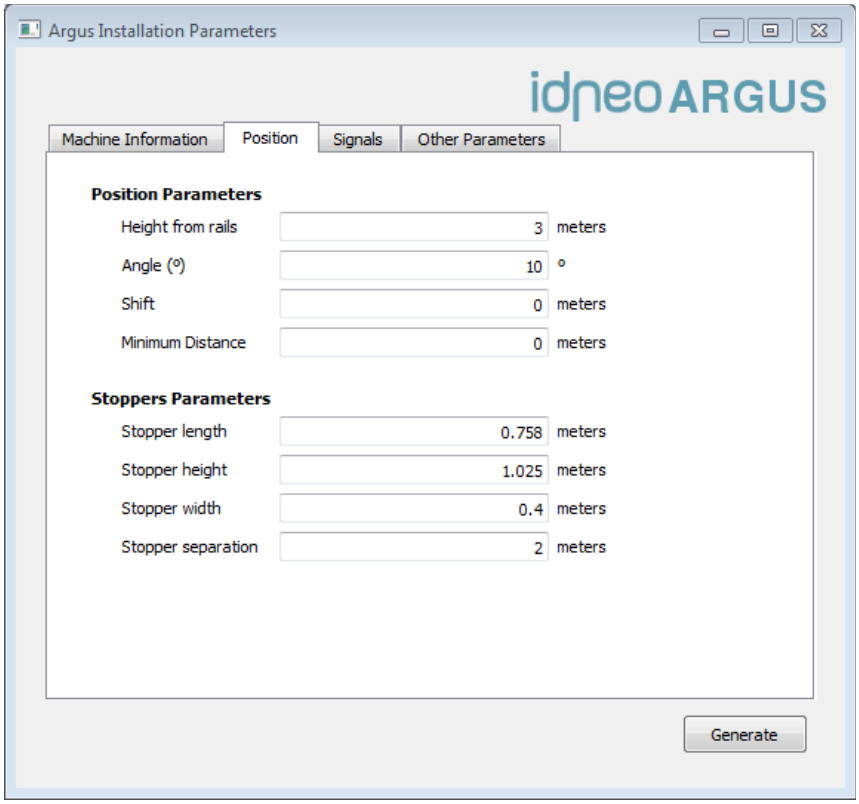


Illustration 56: Positioning screen

The next screen is the one where the mapping of the output of the detection system to the different actuators of the train is defined.

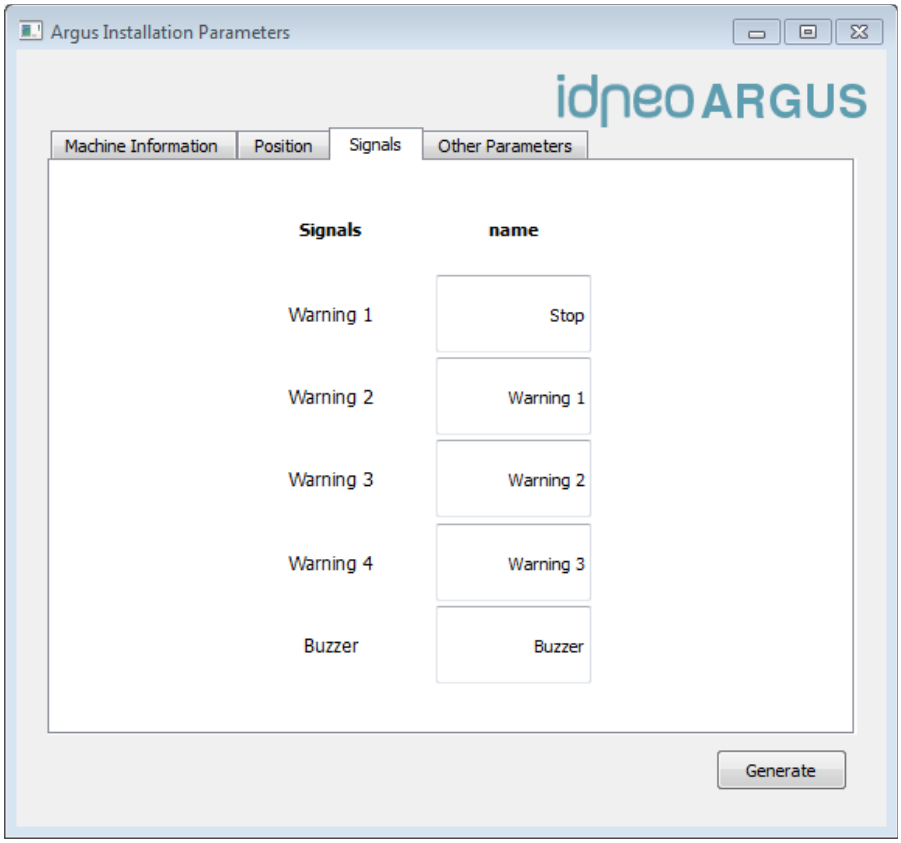


Illustration 57: Signal configuration

It is left to the final contractor to configure system outputs in the way that better fits existing cabin equipment like horns, sirens or alarm lights. There is the possibility, if nothing else, to use the built in Buzzer as a warning. A common use would be to use the top level warning to disengage the machine´s clutch. Stopping the engine itself is strongly advised against by contractors, because it is too disruptive as engines take quite a long time to start again if stopped.

In a typical situation, there will be times when the output of the system must be purposefully disregarded by the driver. On board systems will allow for this, while also recording the fact that the alarm system has been bypassed. This kind of considerations belong to a layer of implementation that is one step above and deals with safety issues as a whole and, ultimately, with liabilities.

The result of running this first program is a configuration file in text form. This file is the input to the second program, *argusapp.bat*.

We choose the file name that ends with the serial number we previously entered from the drop down menu:

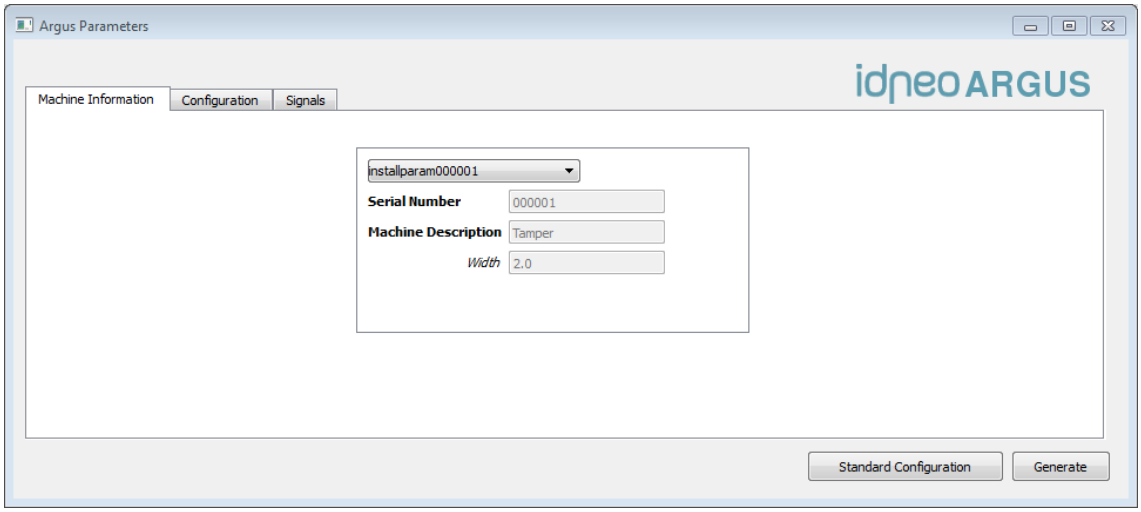


Illustration 58: argusapp.bat welcome screen

The grayed out information has been automatically filled in with the previously inputted information.

While the first program dealt with machine characteristics, *argusapp.bat* deals with the environment where the machine should typically work.

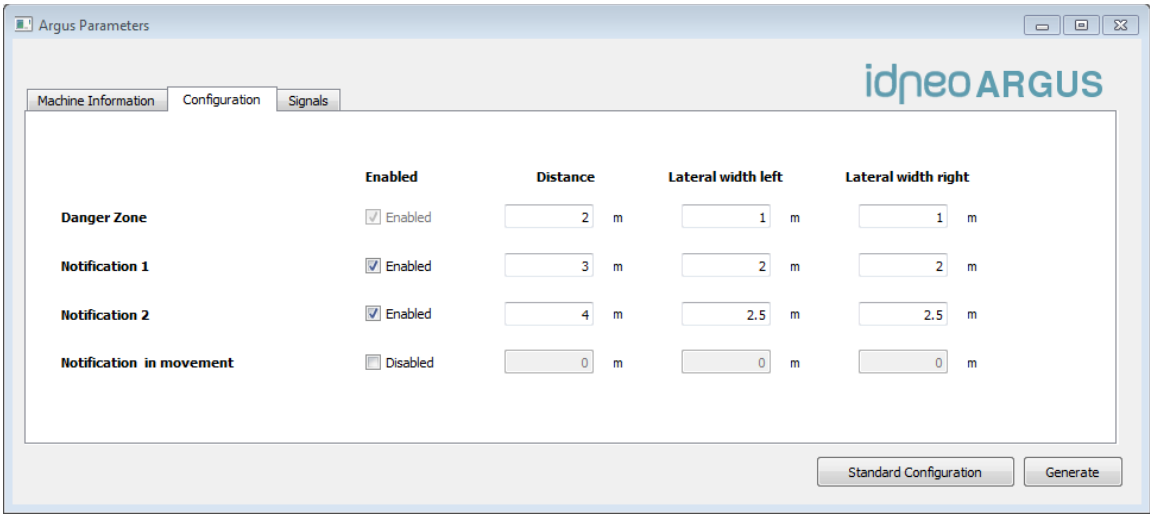


Illustration 59: Zone definition screen

Any configuration of zones is possible, although the original motivation of the project stems from a sudden start situation, and it makes more sense to assign the Danger zone to the area closest to the machine.

The bit about notification in movement is experimental and has not been sufficiently tested.

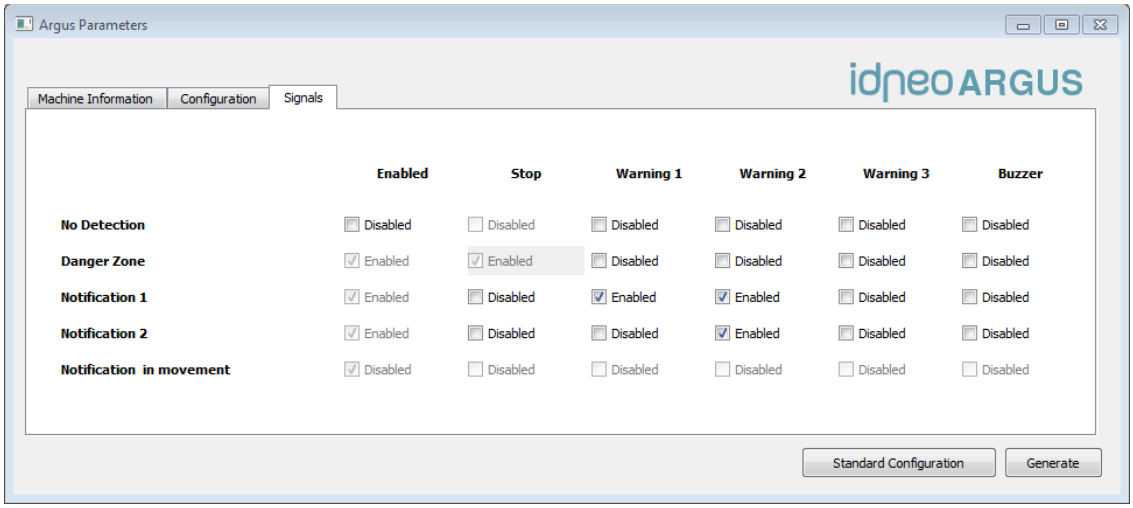


Illustration 60: Zone to signal mapping

Finally, a screen allows to assign each zone to a system output signal.

The configuration file *userparms.conf* is created in the local directory:

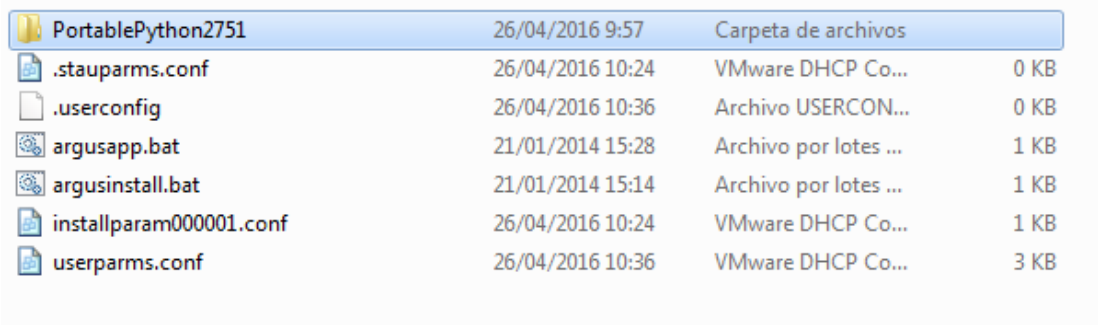


Illustration 61: Output file

It can be then copied to an USB stick in order to configure the machine on site.

User parameters

```
[parameters]
movev0 = 0.0
movev1 = -2.64
movev2 = 0.3
rotv0 = -0.7854
rotv1 = 0.0
rotv2 = 0.0
boxparam1dimension1 = -2
boxparam1dimension2 = 2
boxparam1dimension3 = -3.5
boxparam1dimension4 = -0.8
boxparam1dimension5 = 0
boxparam1dimension6 = 8
boxparam1stopper = 0
boxparam1visible = 0
boxparam1color1 = 0.597
boxparam1color2 = 0.3258
boxparam1color3 = 0.2522
boxparam1thlh = 8
boxparam1thhl = 6
boxparam2dimension1 = -1.2
boxparam2dimension2 = -0.8
boxparam2dimension3 = -1.225
boxparam2dimension4 = -0.825
boxparam2dimension5 = 0.0
boxparam2dimension6 = 0.758
```

```
boxparam2stopper = 1
boxparam2visible = 0
boxparam2color1 = 0
boxparam2color2 = 0
boxparam2color3 = 0
boxparam2thlh = 8
boxparam2thhl = 6
boxparam3dimension1 = 0.8
boxparam3dimension2 = 1.2
boxparam3dimension3 = -1.225
boxparam3dimension4 = -0.825
boxparam3dimension5 = 0.0
boxparam3dimension6 = 0.758
boxparam3stopper = 1
boxparam3visible = 0
boxparam3color1 = 0
boxparam3color2 = 0
boxparam3color3 = 0
boxparam3thlh = 8
boxparam3thhl = 6
boxparam4dimension1 = -1.5
boxparam4dimension2 = 1.5
boxparam4dimension3 = -2.0
boxparam4dimension4 = 0.0
boxparam4dimension5 = 0.0
boxparam4dimension6 = 6.7
boxparam4stopper = 0
boxparam4visible = 0
boxparam4color1 = 0.4651
boxparam4color2 = 0.295
boxparam4color3 = 0.2084
boxparam4thlh = 8
boxparam4thhl = 6
```

```
boxparam5dimension1 = -1.5
boxparam5dimension2 = 1.5
boxparam5dimension3 = -2.0
boxparam5dimension4 = 0.0
boxparam5dimension5 = 0.0
boxparam5dimension6 = 3.7
boxparam5stopper = 0
boxparam5visible = 0
boxparam5color1 = 0.4805
boxparam5color2 = 0.3852
boxparam5color3 = 0.8237
boxparam5thlh = 8
boxparam5thhl = 6
boxnum = 5
speedmaximumclose = 0.0
speedmaximummove = 0.0
boxnotmoveminx = -1.5
boxnotmovemaxx = 1.5
boxnotmoveminy = -5.0
boxnotmovemaxy = 0.0
boxnotmoveminz = 0.0
boxnotmovemaxz = -0.3
boxnotmovesignals = 0.0

[output codes]
301 = 00000000
302 = 00000000
300 = 00000000
0 = 00000000
303 = 00000000
304 = 00000010
305 = 00000111
```

```
[information]
serial = 2002
machine = th

[base output]
0 = 0

[output and]
300 = 255
301 = 255
302 = 255
303 = 255
304 = 255
305 = 255

[output or]
300 = 0
301 = 0
302 = 0
303 = 0
304 = 2
305 = 15

[photo box]
300 = 0
301 = 0
302 = 0
303 = 0
304 = 0
305 = 1

[action delay]
300 = 2
```

```
301 = 2
302 = 2
303 = 2
304 = 2
305 = 2

[disable delay]
300 = 0
301 = 0
302 = 0
303 = 0
304 = 0
305 = 0
```

The first 6 parameters correspond to camera positioning and attitude. This is taken into account only at the final decision stage, when the detected elements are matched to the zones highlighted by the user.

The algorithms are prepared to deal with an arbitrary number of boxes, but all client requirements have called for simplicity in the standard deployment route, so six boxes cover most situations.

The first box is considered the "universe" box. Anything outside of this box is quickly suppressed.

The second and third boxes are the stopper boxes. Their dimensions are hard coded to adapt to the dimensions mandated by international standards, and they are not subject to change. Points found within these regions are removed from calculations.

The fourth and fifth boxes are the warning and alarm zones. Typically the alarm zone is completely inside the warning zone, though this need not be the case.

Boxes are numbered in order of evaluation. Potentially more boxes could be added just adding their *boxparam* lines to the user parameters file.

The first six *boxparam* parameters correspond to the physical dimensions of the detection areas. They are, as listed, minimum and maximum *x* corresponding to the machine width, minimum and maximum *y* related to the height of objects, and minimum and maximum *z* corresponding to distance to the machine. All values are input in meters.

The *Y* dimension parameters are hard coded by the GUI application and not open directly to the user except if he chooses to tinker with the configuration file directly. As default value a catch-all height setting is in place for the universe box and a setting that looks at a 40 cm slice is in place for the alarm and warning zones.

The stopper parameter indicates that the box encloses a structural element of a machine. All the nodes found inside the box have to be suppressed prior to other considerations.

There are two *lh* and *hl* parameters introduced to allow for hysteresis and avoid the alarm to go on and off constantly in case some object is sitting just on the limit of a zone. This is quite important because it could give the impression of system malfunction, with the consequence that the driver could turn off the system completely.

The *visible* and *color* parameters belong only to the installation phase and are used to draw the boxes on the HDMI output of the system while

adjusting on the field. The video output is not used during normal operation as it slows the detection quite noticeably.

The *output* section is the way to enter the mapping between triggered zones and output signals into the system. It is quite literally the transliteration of the output signal arrangement. For instance, zone 305 is set to 00000111. That means that whenever something enters zone 5 signals 3, 4 and 5 will be activated.

The and/or section is a refinement to be able to output a more complicated logical function using the occupied boxes. By default, the boxes are mapped to signals individually, without making any combinations.

The photo section is included as an additional feature, mainly for debugging. If any of the five boxes is set to one then the system will record a photo on external recording device whenever that particular box alarm is triggered. In this way, during debugging we can tell if the system actually reported a false positive or not by evaluating the evidence.

The delay is set in order to avoid early triggering due to spurious noise activating the system. At least two consecutive frames are required to trigger an alarm as a default.

Reporting

Once the alert has been triggered, the system should have a way to communicate the alert level to the external host, e.g. the tamping machine. There are a lot of machines that could benefit from this alert

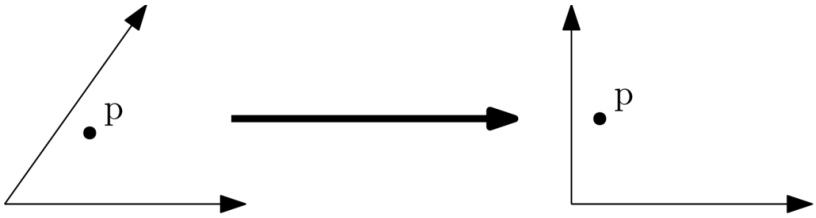
system, and different companies have different regulations and requirements. In order to translate the alert level to any kind of action, a logic matrix has been implemented in the settings file. By changing the settings file in the user interface, we can assign any number of outputs to any given alert, or disable momentarily the outputs, allowing the driver to bypass the system during particular maneuvers or transportation.

Chapter 9 Map creation

9.1 Algorithm

In order to correct the distortion, we start by taking a picture of a reference rigid checkered board. We now that the intersections of the squares are evenly spaced. The objective is to find a transformation such that applying it to our distorted picture we can replicate the evenness of the checkered board.

The algorithm idea is to refer each point in the original picture to a system of coordinates along the sides of the square where it belongs. Since we know that the squares are actually perfect, changing the system of coordinates to an orthogonal one and making a change of base should allow us to find a point in the rectified square that occupies a position relative to the sides of the square that is in proportion to the one the original point occupied in the original distorted square.



It is, in fact, a piecewise linear transformation of the picture.

The smaller the squares are, the more granular the distortion correction will be. Before building the adjustment tool we experimented varying the number of squares using a large screen TV and a pattern generator until reaching a compromise between computational cost and distortion errors left in the picture.

Define matching nodes

First off, we must find intersections on the checkered image. This is quite straightforward and can be done using aggressive filtering, because the image is pretty well contrasted and we don't care for details or shadows. In this way a number of nodes can be obtained from the distorted picture.

The theoretical positions where the nodes should have been are very easy to generate as well, but then they have to be matched to the nodes available in the distorted picture. It is often the case that the rows and columns are cut at the center because of barrel distortion, and it is sometimes difficult to assign the points.

As it is detailed in the implementation section, a special MATLAB user interface was devised to overcome this particular hurdle.

Once we have a paired list of squares, half from the distorted picture and half from the theoretically calculated perfect checkered pattern, we slice the squares down the diagonal and build a list of paired triangles that is better suited to our calculations:

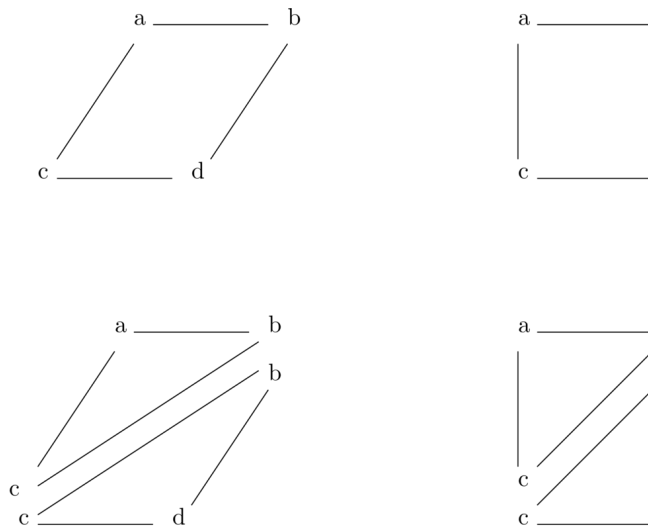


Illustration 62: Slice squares into triangles

We end up with twice as many triangles as there were squares. Each of this areas will be matched against the corresponding one, and the pixels warped to proportional positions.

Base change

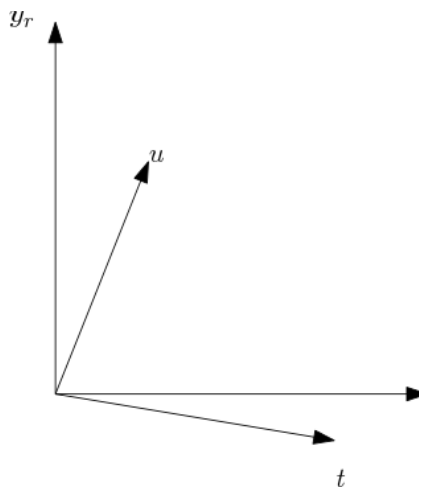


Illustration 63: Base change

Let's consider the corner of a typical triangle. In the original distorted picture we take the sides of a square that is quite askew acting as the original basis in which every point of the original square is to be referenced. We call this non orthogonal base the *(u, t)* base.

We will call (x_r, y_r) the basis for the orthonormal base which is the target of the transformation. It is easy to see that the vectors of the *(u, t)* base expressed in (x_r, y_r) coordinates are precisely the coordinates of the points of the triangle in the picture, if we adjust for the common origin to be the center:

$$\begin{bmatrix} u \\ t \end{bmatrix} = \begin{bmatrix} m_{11} m_{12} \\ m_{21} m_{22} \end{bmatrix} \begin{bmatrix} x_r \\ y_r \end{bmatrix} \quad \text{(Equation 9.1)}$$

This is how both sets of base vectors are related.

As in $x=m_{11}x_r+ m_{12}y_r$ and $x=m_{21}x_r+ m_{22}y_r$, in the original picture taking always the corner node as origin. In order to define any point as it relates to the (x, y) base then we have to left multiply for the inverse of the *M* matrix, being *M*:

$$\mathrm{M} = \begin{bmatrix} m_{11} m_{12} \\ m_{21} m_{22} \end{bmatrix} \quad \text{(Equation 9.2)}$$

Then we would have

$$\begin{bmatrix} x \\ y \end{bmatrix} = \mathrm{M}^{-1} \begin{bmatrix} u \\ t \end{bmatrix} \quad \text{(Equation 9.3)}$$

as a means of translating any vector from the distorted basis to the orthonormal one.

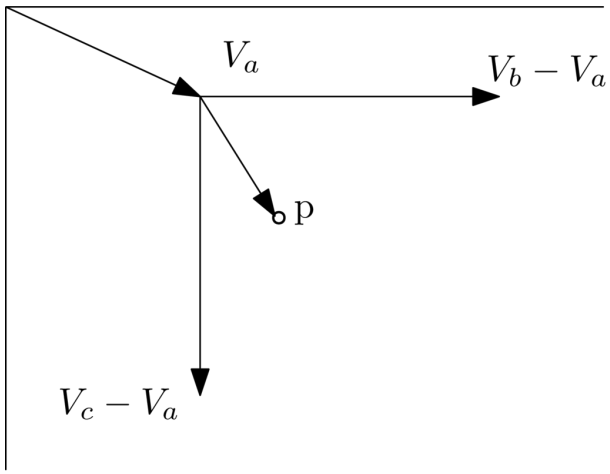
This must be done now for all points of the original picture that fall within this triangle.

Map construction

Once we have each point expressed in orthonormal coordinates, we have to calculate the coordinates of the actual picture point. We are using

a different base for every square of the checkered pattern, so we have to compensate for the offset of every origin of coordinates.

We have expressed the point p_r in terms of *Vb-Va* and *Vc-Va*. To get the actual picture coordinates we have to add back *Va:*



I

n order to ease later calculations, the integer and the decimal parts of the coordinates are coded as integers and stored separately.

Alignment

Assuming all individual distortions are taken care of, the job is not done. Our intention is to extract disparity information from cameras that are facing exactly forwards and are straightened along the axis that connects the two optical centers. This can be achieved mechanically only to some extent. Even the slightest mechanical tolerance can throw disparity off by a handful of pixels.

As we step away from the camera, even a pixel can map to a very sizable distance. Furthermore, misalignment cannot be corrected using just an offset, as it is a vectorial magnitude that affects three axis. A proper lineal transformation must be put in place in addition to the non linear one already necessary to account for optical aberrations.

A way to go about this would be to choose one of the cameras as the master, and take a set of two pictures one with each camera. In these pictures, a number of physical points in space would be easily identifiable between both pictures.

An algorithm should be written to rotate and twist the second picture so that every point on the first picture finds its match in a point on the second picture that has the same *v* coordinate. The algorithm will reach its conclusion when some measure of error is minimized.

In an ideal situation, the difference of *v* coordinates between the points of the first and of the second picture would be zero. A good way to optimize the transformation would be to minimize the sum of squared differences:

$$\sum_{i=1}^{n}(v_i - u_i)^2 \qquad \text{(Equation 9.4)}$$

There is no easy way to ensure the convergence of this algorithm, but one approach could be to separate the transformation into three rotations along the three axis.

9.2 Matlab implementation

Define matching nodes

Firstly we pair the nodes from the raw picture to their ideal positions.

Once the checkered board nodes have been paired to those of the ideal picture, we start the creation of the distortion map.

As we mentioned in the algorithm section, we try at first to find which triangle is containing each original picture pixel. We use the function *buscatriangle_nocell,* which in turn calls repeatedly the function *pintri:*

```
function [ in ] = pintri( n0,n1,n2,p ) %#eml




s0=sign(vectprod(n0-p,n1-p));

s1=sign(vectprod(n1-p,n2-p));

s2=sign(vectprod(n2-p,n0-p));


in=false;


if s0>=0 && s1>=0 && s2>=0

    in=true;
```

```
end
if s0<=0 && s1<=0 && s2<=0
    in=true;
end
end
```

As can be shown, the condition for a point to be on the inside of a triangle is met if the vectorial products of are all the same sign.

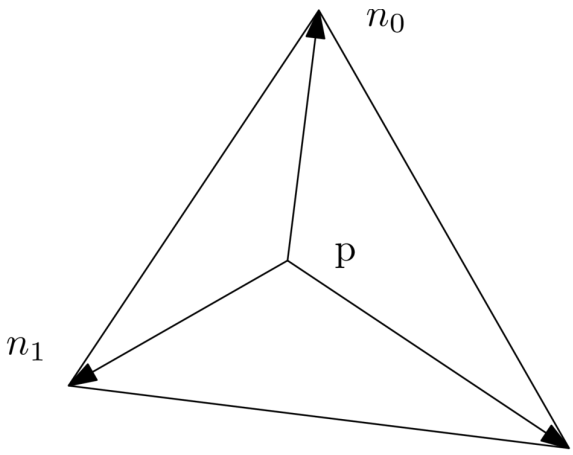


Illustration 64: n0xn1, n1xn2 & n2xn0 have the same sign

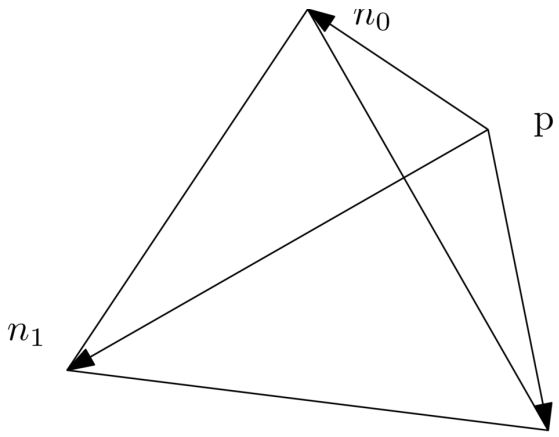


Illustration 65: One of n0xn1, n1xn2 & n2xn0 has a different sign

We assign the corresponding original and target triangle points:

```
a=triorg(1,:,n);

b=triorg(2,:,n);

c=triorg(3,:,n);

va=tritarget(1,:,n);

vb=tritarget(2,:,n);

vc=tritarget(3,:,n);
```

Base change

Now we calculate the inverse matrix. In order to speed up things we store the inverted matrix so that the inverse only has to be computed once per checkered board square, not once for every pixel.

```
if hihainv(n)
```

```
        temp=inversa(:,:,n);
    else
        M=[(c-a)' (b-a)'];
        temp=inv(M);
        inversa(:,:,n)=temp;
        hihainv(n)=1;
    end

    TU=temp*(punt'-a');

    t=TU(1);
    u=TU(2);
```

For each point in the original picture we obtain a set of *(t,u)* coordinates that are the coordinates of that same point expressed as a function of the vectors that define the skewed triangle where it belongs.

We then calculate the point that would proportionally match that point in the rectified picture:

```
valorpunt=double(va)+double(vc-va)*t+double(vb-va)*u;
```

We have added the coordinates of the new origin to the same proportion of the rectified square axis as taken by the original point in the original square.

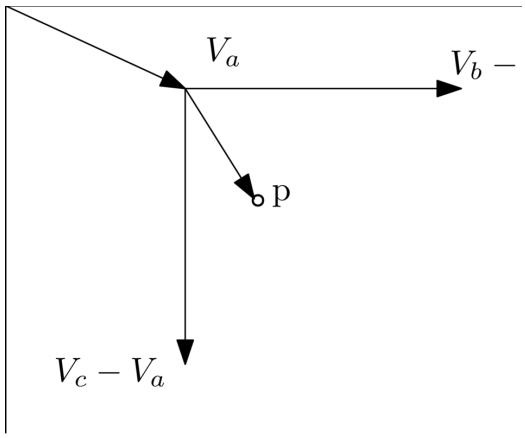


Illustration 66: Point recomposition

Map construction

As we have seen in the *transforma* function, we make a separate use of the integer and decimal part of the map coordinates.

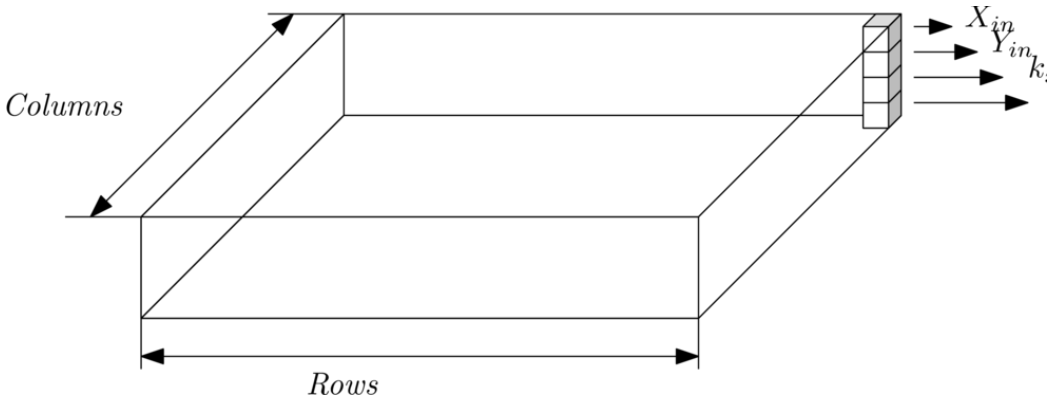


Illustration 67: Map structure

We separate them and encode them into integers.

```
            mapa(1,j,i)=floor(valorpunt(2));

               mapa(2,j,i)=floor(valorpunt(1));

               mapa(3,j,i)=int16((valorpunt(2)-
double(mapa(1,j,i)))*16384.0);

               mapa(4,j,i)=int16((valorpunt(1)-
double(mapa(2,j,i)))*16384.0);
```

The reason behind this is that DSP floating point operations are way slower than integer operations. Once the map is finished it will not be recalculated again. Instead, every time a picture is acquired, the fixed map will be used to rectify the pictures as an entry point to the system.

Distortion correction

We have devised a controlled procedure to construct the map. This makes it easier to detect errors and fine tune the algorithms.

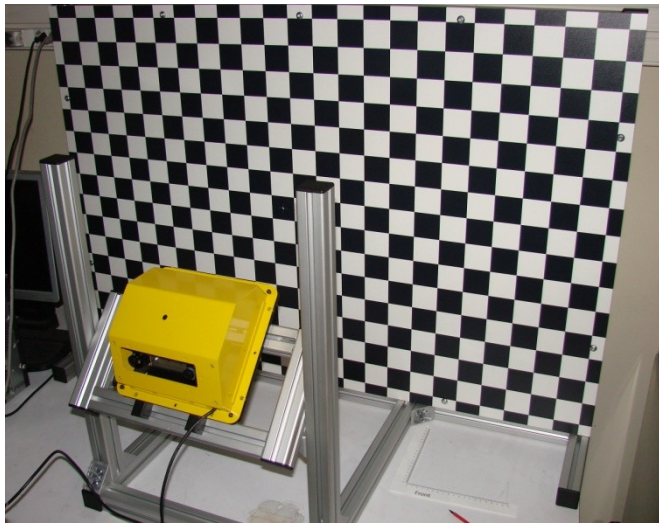


Illustration 68: Distortion correction setup

Two checkered patterns were created. A smaller one to correct for individual camera distortion, and a larger one to make the alignment. The

reason for choosing a white and black square pattern is to simplify pattern recognition. In both distortion correction and alignment the purpose is to identify points in the image that correspond to points in the scene that have a precisely known position. Calibration can take place when our system's measurement can be matched against a known reality.

The assembled system allows the capture of static pictures to an attached USB using a command sent via UART. We have chosen to assemble the full system before adjust. In this way we prevent any mechanical strain introduced during the screwing of the enclosure to introduce any misadjust. Also, the possible effect of the front glass is included in the chain, as all corrections are done including this layer.

In Illustration 68 the head part of the system is shown. The cameras are fully assembled inside and looking out through the front window. On the back there is the distortion correction checkered board. It is a rigid board with a number of black and white squares. During the capture of the images required for distortion correction, the head unit is of course facing the board.

A set of two pictures is taken and then the head unit is moved to the alignment position.

Alignment

A second, bigger pattern is used during alignment correction.

Illustration 69: Large canvas screen

The larger poster is used to take the pictures for the alignment adjust. In this setup distance to the camera is not so critical, but has to be somewhere in the order of magnitude of the subjects we expect to detect in order to maximize the sensitivity around that very region. The pattern is printed on a large plastic canvas.

There is a second position prepared to take the pictures necessary for alignment.

After all pictures are taken, the USB is extracted and the pictures can be used for map creation.

A graphical user interface has been prepared in MATLAB in order to allow an operator to efficiently make maps for a batch of units.

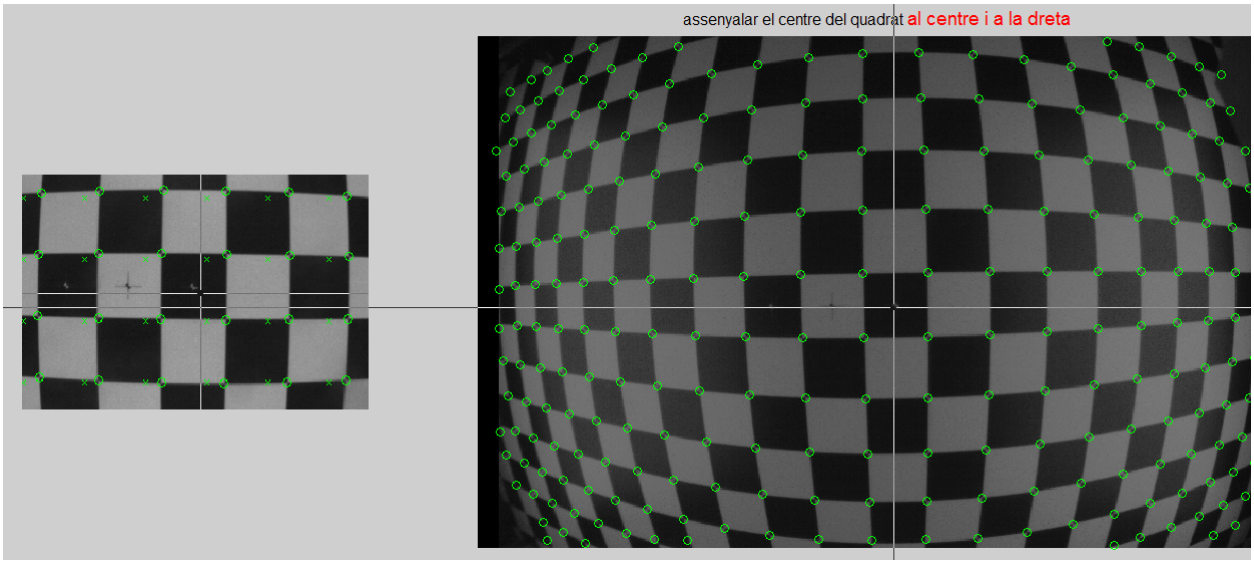


Illustration 70: Distortion correction for main camera

In Illustration 70 we can see the raw image captured by the main camera. The operator is instructed to select the square that is just to the right of the white center of the checkerboard. The square size of the checkerboard is approximately on top of a black square, with the middle white square in between.

In the picture we can see that the program has already detected a number of square crossings. The exact number depends on the camera attitude and field of view. These are the nodes that will be used for optical correction for the main picture. The operator is then asked to count how many *full* rows or columns lie in all four directions relative to the center square. In this example we can see four rows up, four rows down, eight columns up and eight columns down.

The purpose of this input is to disregard the rows or columns that are not completely inside the field of view. We wish to rectify a picture that will be of rectangular shape in the end.

The same process is repeated for the second picture, and distortion maps with the same proportions are individually generated for each camera.

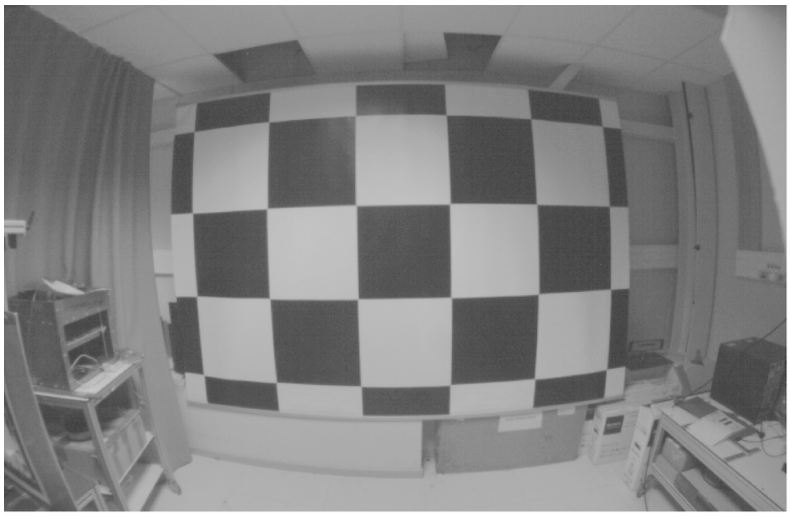


Illustration 71: Main picture before distortion correction

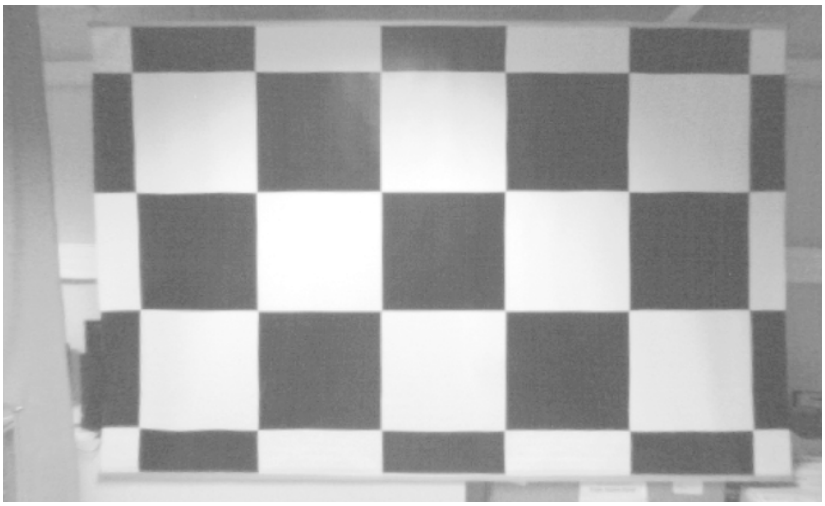


Illustration 72: Main picture after distortion correction

Next up is alignment correction. A distortion free picture is generated for each camera using the preliminary maps and the images are shown on screen for alignment correction.

The user interface shows the image of the big screen once the optical correction is applied. It asks the operator to point at the corner squares of the screen. This is just to delimit the node search area. Alignment adjustment is done at a chosen distance of 4m. At this distance, even with a screen this big, some outside elements come into the picture that could potentially be picked up by the node search algorithm.

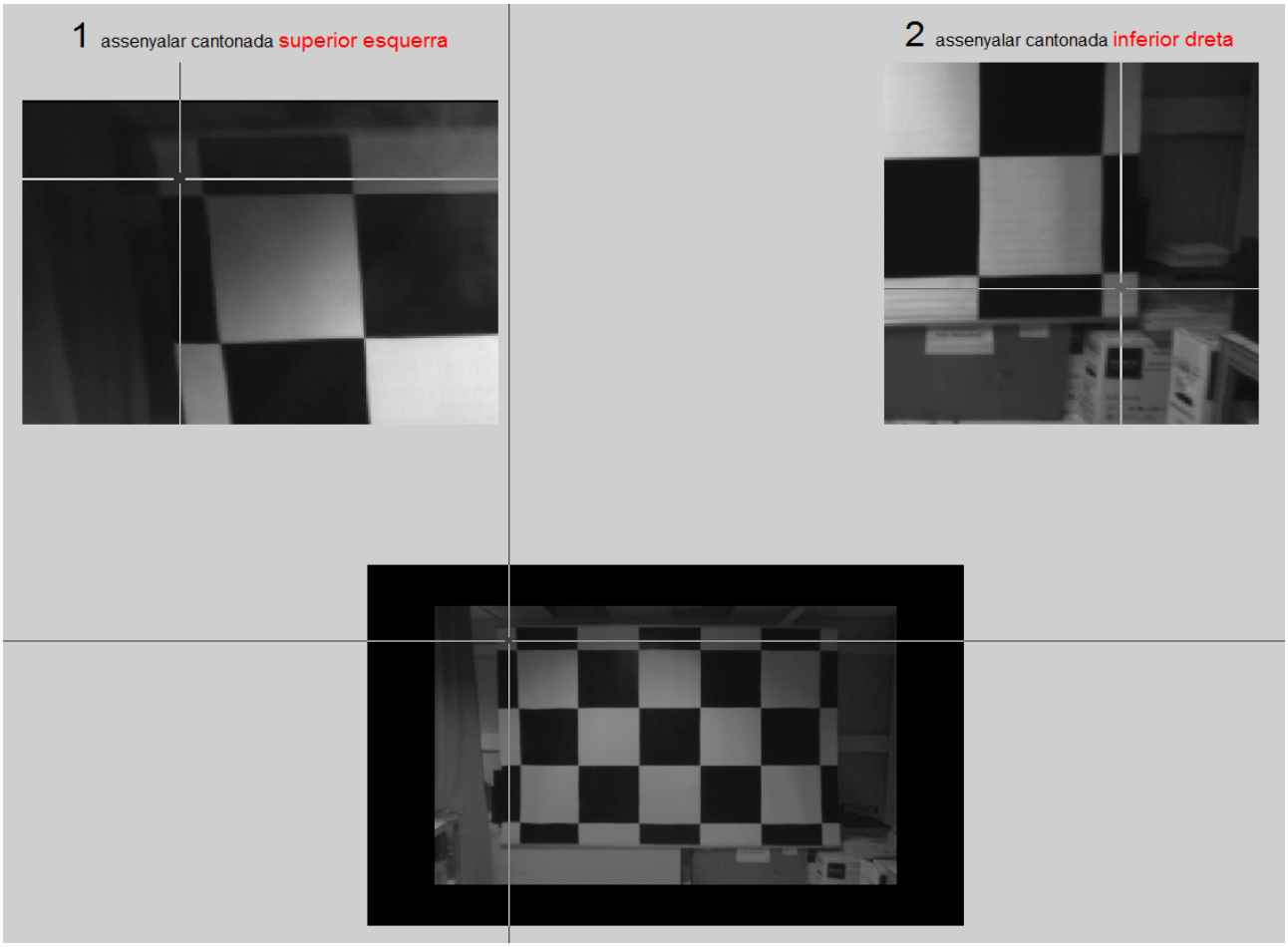


Illustration 73: Alignment user interface

The algorithm picks up the intersection nodes of the checkered board:

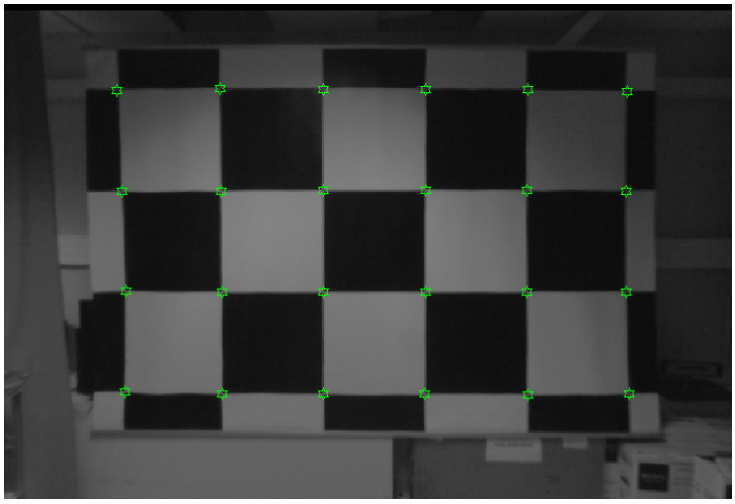


Illustration 74: Alignment nodes

The same procedure is performed on the second camera's distortion corrected picture, and at this point we have two optically corrected cameras and two sets of alignment nodes.

We mentioned earlier that a way to align the pictures woud be to minimize the sum of squared differences between known points of two patterns appearing in the first and second pictures. As seen in Equation 9.4:

$$\sum_{i=1}^{n}(v_i - u_i)^2$$

Where v_i and u_i are the coordinates of a set of points in the first and second picture respectively.

What we will do is consider the first picture as a reference and try to find the combination of rotations and tilts that applied to the second picture result in a minimum of the above quantity.

We make use of *fminsearch*, a function that searches for a minimum in a multivariable setting. This function is not included in the set of functions that can be handled by the MATLAB compiler, but we are not concerned about this, as the map creation stage is made completely offline and the code runs directly on MATLAB and not on the system hardware.

Also, we are not recalculating the map after each iteration, just tilting the sub picture. What is needed is to define an error measure that tends to zero as the alignment between pictures gets better, then define a tolerance limit that sets the criteria for accepting an adjust.

The function fminsearch takes as parameters three items:

x = fminsearch(fun,x0,options)

1. A function handle to a function that has to be minimized.
2. The initial input to the function.
3. An optimization options structure.

In reverse order, the optimization options structure is a special structure that MATLAB uses to adjust the parameters of MATLAB optimization functions, fminsearch among them. The optimset structure is set as

```
options=optimset('MaxFunEvals',100000,'MaxIter',100000,'TolX',1e-
6,'TolFun',1e-6,'Display','final');
```

Inside this structure limits are set on the tolerance that can be accepted on a final result, and on the number of iterations until convergence is reached.

The initial input is an initial guess of a value that minimizes the error function.

Function handles are a MATLAB construct that makes possible passing a function as an input to another function.

One way to create function handles is using the anonymous function syntax. For instance, in an example

```
>> par_r=@(r1,r2) (r1*r2/(r1+r2));
>> par_r(1000,3000)

ans =

   750
```

par_r is defined as an anonymous function that calculates the parallel of two resistors.

The function is immediately available for use and lives in the name space. Same as a variable, it may be deleted from memory using the command *clear*, and very much like a variable, it can be passed to another function as input. The idea is to pass the error function to *fminsearch.*

The key step in the optimization process is to define the error function. To this end, a function called *comparapunts* is defined. It compares two sets of points. To be precise, it compares the nodes captured from the large canvas picture as seen on the undistorted pictures of the two cameras. *Nodesbase* contains a collection of points from the main rectified image. *Nodesoffset* is the location of those same points on the second picture. Furthermore, we know which node of *nodesbase* matches each point of *nodesoffset.*

The function *comparapunts* takes as input:

1. The two ordered collections of nodes.
2. The rotation, translation and tilt quantities to be tried.
3. The target for disparity.

We know which value to assign to the disparity target because we know the precise location of the canvas. The values for rotation, translation and tilt are applied to construct the corresponding matrices and apply this perspective to *nodesoffset*.

Then a measure of squared distance is computed between the nodes.

```
nodesdesp=perspectiva(nodesoffset,desp,rotah,rotav,tilt,f,f,1);

indexes=1:24;

nodes910=nodesbase(indexes,:);

nodes910s=nodesdesp(indexes,:);

difnodes=(nodesbase(:,1)-nodesdesp(:,1));

difx=( disptarget*ones(size(nodes910,1),1)+(nodes910(:,2)-
nodes910s(:,2)));

dif=sum(difnodes.^2)+sum(difx.^2);
```

The output of *comparapunts* is dif.

We have chosen to assign equal weight to:

1. Making zero the *v* component, which is the proper value in an orthogonal setup.

2. Honing in on the disparity target for the *u* component differences.

Once the *optimset* structure is populated and the error function defined, we are ready to ask MATLAB to perform the optimization:

```
vectorout=fminsearch(@(vector) comparapunts(nodesbase, nodesoffset, [0
vector(1) vector(2)]*FACTOR,vector(3),vector(4),vector(5),f, disptarget,
disptargetfar),[  0 0 0 0 0 ],options);
```

Once the optimization has converged, the optimum perspective is applied to the first set of nodes of the slave picture. The slave map is redone using these rotated nodes, and the preliminary slave map is discarded.

A final sanity check can be done making an alpha blended overlay of the rectified images of both cameras:

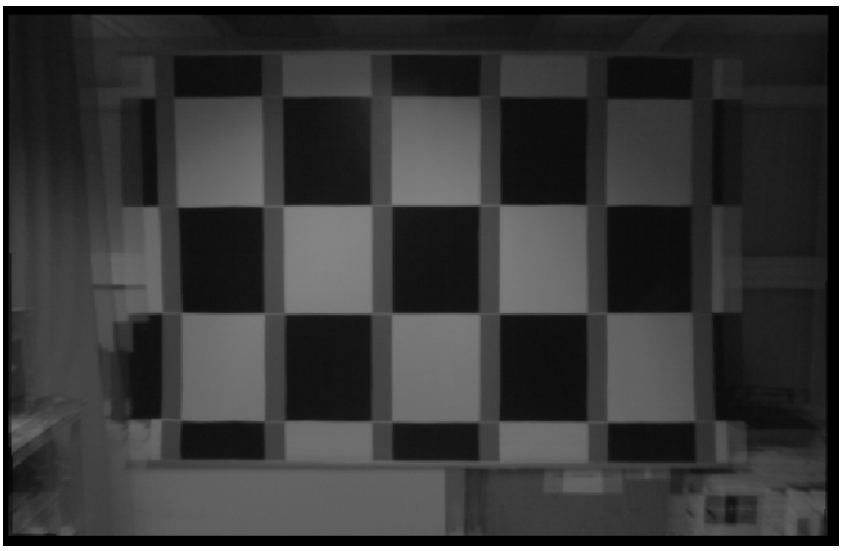


Illustration 75: Combined picture

In this kind of picture alignment errors stand out immediately. Three things can be easily checked:

1. The vertical alignment between every square of the poster should be close to perfect. In reality, the alignment averages out the error over the whole picture, and there can be a rounding up or down of one pixel.

2. The horizontal difference between the pictures, that can be measured zooming in on the vertical edge of any square, should match the disparity that corresponds to the distance between the poster and the cameras, according to the focal distance and to Equation 7.5.

3. That measured disparity should be the same for any square intersection.

Chapter 10 Future developments

There are a number of open exploration lines that can further improve the system.

10.1 Bicubic interpolation

This one is clearly conditioned by the available processing power. The potential benefit from improved interpolation is higher granularity of the disparity extending the range of the detection area.

10.2 Adaptive feature detection

One way to buy back some of the processing power could be to limit the exploration of picture areas with a lower likelihood of hitting a target. The problem is that DSPs don´t work well in a constant start and stopping regime. A coarse exploration of the whole picture and then a fine exploration of those areas that have had hits in the recent past could be proposed. To alleviate the load, those areas should be mapped into a special format in order to streamline the process.

10.3 Improved clustering

The potential benefit of this kind of clustering would be to enable the system to tell apart persons from larger objects.

10.4 Time dependent object tracking

Beyond the repeated triggering of the same zones, identifying the same disparity clusters from one frame to the next has some benefits. If some object is seen in the warning zone with some trajectory, the potential crossing into the danger zone could be forecast. In harsh weather or limited visibility conditions, keeping track of the places where a detection

has just appeared could be used to devise a system of relative sensitization, making it easy for repeated appearances to trigger an alarm and making it harder for a one-off occurrence to be considered.

Chapter 11 Conclusions

1. A compact stereo vision system is feasible as a person detector system in a rail works environment. No special luminary system is needed in addition to the lighting that is already in place to carry out the works. The detectability range can easily go up to 6 meters, which is enough for a system that is intended to prevent running over persons at the time of the machines starting up.

2. Using a sparse matching algorithm based on gradient is enough to obtain disparity pictures that are detailed enough for detection purposes.

3. Detailed scene reconstruction is unnecessary for an alert system, and volumetric calculations are enough.

4. Any kind of alarm system that goes aboard a train must be highly configurable to the needs of the contractor.

5. Even when the feature detection algorithms are valid, specific DSP programming is necessary to make the implementation reach a frame rate that is useful in a practical setting.

6. On the other hand, clustering, disparity cleanup or any other refinement algorithms that takes place at the nodes level are in principle cheap in terms of processing power and can be implemented based on their MATLAB prototypes.


MSA

P_Argus