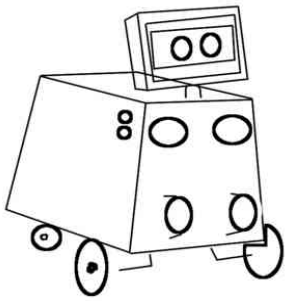
ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS
DEPARTMENT OF INFORMATICS

GuarddoG Project



DRAFT / WORK IN PROGRESS

Author : Ammar Qammaz
Supervisor : Georgios Papaioannou

Athens , December 2011

START

Introduction and motivation

A few opening remarks

Humans increased their physical power during the industrial revolution using machines. They were able to create giant dams , factories , cars , airplanes and skyscrapers to make their everyday life easier . Technology has continued to improve exponentially and in the current age , labeled by some as the age of informatics or the internet , mental capabilities where multiplied. Merging the following two revolutions we can finally partly replace ourselves from dull and repetitive tasks of day to day life that will gradually stop to trouble the human kind leading to a more pleasant life. The GuarddoG project is about making machines that can see and act as a futuristic private guard .

The process of creating an autonomous robot that can perceive its environment and react and interact with it took nature millions of years. From the first bacteria to multi cell organisms , the wolf then the dog and the human , enormous evolutionary differences created beings of immense complexity and perfection. For someone to build something that took such a great amount of time in even a quarter of a lifetime is over-ambitious. An extra observation that is thought provoking is that while humans in complex decision making such as chess playing or tactic games with a limited set of rules have been surpassed by computers. In contrast in simple things for humans such as perceiving space , time , and “natural logic” every human has an innate superiority a result of the millions years of natural selection with these characteristics as a basis.

That being said GuarddoG does not attempt to create a dog (with everything a dog implies) , because this is practically impossible. Its goal is replacing a specific function of a dog as a guardian. I am very optimistic that with time robots will eventually be improved enough to be able to perform a multitude of tasks approaching something that will surely be different than a real dog , better at some things , and worse at some others.

Even though the future will offer even more tools , even now thanks to the marvelous technology and work of all the scientists , mathematicians , physicists , chemists , engineers and computer scientists (we are literally standing on the shoulders of giants) I was able to construct something very close to my original target , spending a fraction of the money and time that would be required before 15 or even 10 years.

A better way for someone to visualize the small subset of functionality that is attempted by computer vision algorithms and in this case GuarddoG , is to compare it to the holy grail of cognition and intelligence , the human brain. Though computers for many years have managed to surpass human experts on tasks like playing chess , remembering sequences of numbers , performing arithmetic calculations on large data sets and recently even guessing questions to answers (IBM Watson on the Jeopardy TV Show) , things that everyone can do without even thinking about , like walking , identifying 3D objects and faces , and coordinating his head , eye and body movement are currently unachievable by machines at least to the extent of human performance .

This is a good indicator of the level of optimization that has taken place through the millions of years of evolution , because there is no doubt that if playing chess was a trait that leaded to natural selection the human brain would be totally different and have a much greater affinity towards these kind of activities. On the other hand , if breathing , beating the heart or walking and identifying objects wasn't crucial for the survival of the human species , to master these kind of activities could may well be as difficult and time consuming to be achieved as mastering chess .
*

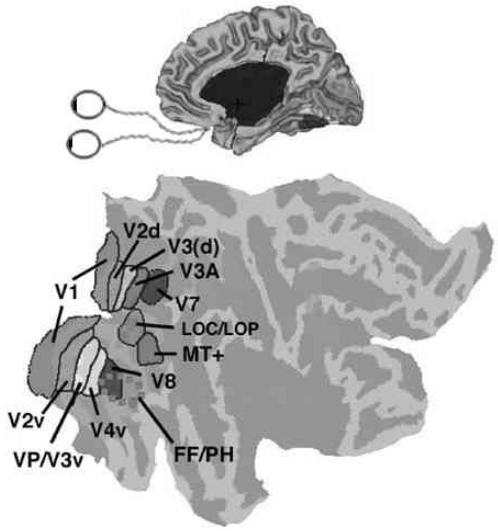


Illustration 1: A recent mapping on the first areas of the brain responsible for human vision ***(TODO : will be replaced , it does not fit)***

*

Project Goal

The goal of the "Guard Dog" Project

The goal of the Guard Dog Project is to build a robotics platform that can act as a guard , traverse a known path and fend off intruders. In case of a security breach it would signal the alarm and begin to follow the perpetrator and after a set distance would resume its previous path.

Robotics and computer vision are not a new domain of computer science and electrical engineering. It was especially shocking for me to see video footage of experiments in the AI Lab of Stanford (for example Les earnest and Lou paul and the Rancho Arm) circa 1971 that perform object detection , complex decision making and that actually use more or less the same algorithms as current robotics projects do. The major difference is not so much about the methods used , but the exponential improvement on computer hardware , popularly coined as Moore`s Law.

We are living in times where many high-end mobile phones actually have more complex processors than the satellites of the first mission to the moon and that experiments such as those that required equipment that cost millions of dollars in 1971 and could only be done in universities or government research centers can be reproduced with consumer electronics readily available everywhere. Unfortunately the consistent computation of the world around a robot is still a very difficult and expensive task with a generic CPU and no specialized hardware , but yet it seems almost feasible when you achieve even something that can work 10 times slower than a human.

Of course an additional goal of the project is to perform guard duties using only cheap building blocks but not passive sensors as modern security systems do. Instead building a semi-intelligent agent that can do this job the way humans would do it. It is an exploration of the possibilities and limits of current technologies along with software that can leverage the capabilities of computer hardware in an efficient way, to achieve an almost working result.

It is also interesting to note that the same computer vision libraries can , with adjustments , be fitted for tasks like driving cars in city streets to helping blind people find their way or any task that involves using optical information of ones surroundings to achieve a related goal.
The main difference would be the risk/cost and risk/performance ratio since a computer driving a car at full speed can do much worse damage when compared to a small robot bumping on a wall.

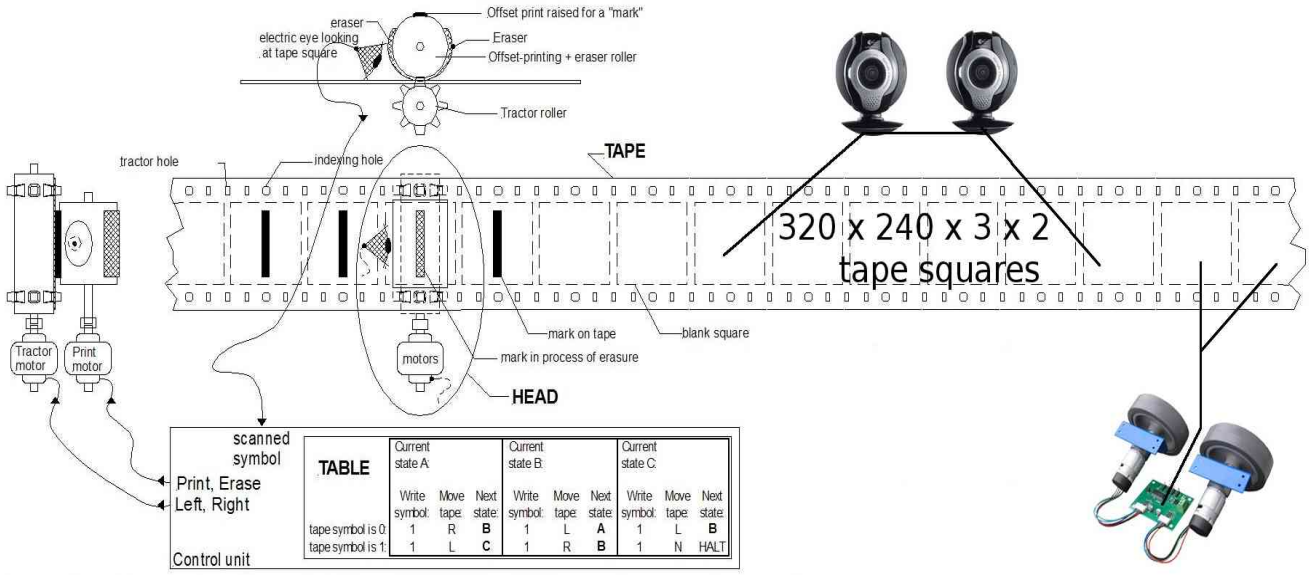
*

Mathematical Framework

1.1.0 Overview

The ease with which humans sense the world makes the problem of computer vision seem “easy” to solve. In fact the way we see is so natural and persistent that even scientists in the field made biased over-optimistic predictions about it. The fact is that despite the exponential growth in computational speed , and although there is a very big market that could certainly use vision algorithms to automate tasks , there is still no defacto algorithm that can compare to what human vision performs. Moreover from simple reflexes as maintaining focus and coordinating ones gaze , reading text , to tracking your position in an unknown city , vision seems to be “AI-Complete” , since understanding and combining what is seen is an altogether different task than the small building blocks which are presented here.

A robot that can see and interact with the world , is basically a Turing machine on wheels. Therefore the whole model presented here is an adaptation of different mathematical concepts and a fusion of them together. The strip of tape in this Turing machine is constantly filled with symbols of light intensity as the light gets reflected and activates the camera sensor elements. When the control algorithm decides that the robot has to move it writes it to the according tape elements and the motors move , producing a new view of the world.



A fanciful mechanical Turing machine's TAPE and HEAD. The TABLE instructions might be on another "read only" tape, or perhaps on punch-cards. Usually a "finite state machine" is the model for the TABLE.

The first thing to take into consideration beginning to approach this problem , is how the physical world is being represented by the cameras. They are , after all , the means with which the GuarddoG/RoboVision algorithm collection , a “meta”physical entity can take a peek into reality. The data acquired must then be filtered to remove deformations and distortions that may corrupt the whole process. These steps are described in the Camera Model , Camera Calibration and Image Rectification parts of this document. Once two corresponding images of the projection of the world on the camera sensors are aquired , they are examined for optical cues that reveal the details of the world in three dimensions and also the robot's position. This is also discussed extensively , and the Disparity Mapping algorithm used by GuarddoG is a new implementation. When all these steps are finished , the next one is tracking the position of the robot (LK Optical Flow , RANSAC Homography) and the combination of the successive 3d Views together (SLAM , Obstacle Detection).
The final piece of the algorithm is a knowledge base that will set its goals and keep the state of the world , and steer the robot towards achieving them. For GuarddoG , its goal is the traversal of a standard path , and raising the alarm if a breach is found.

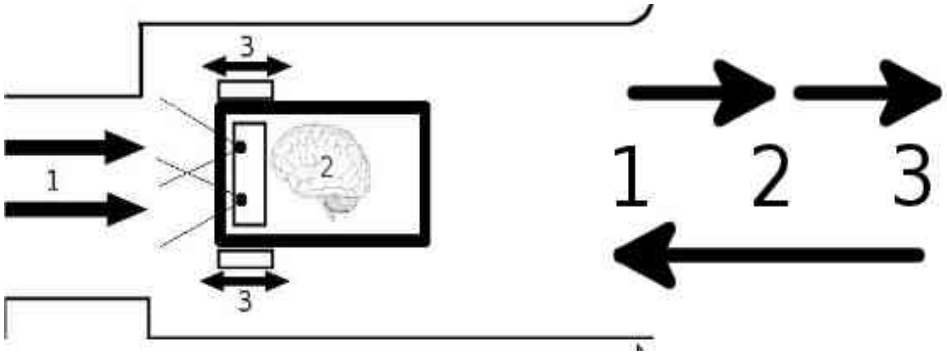
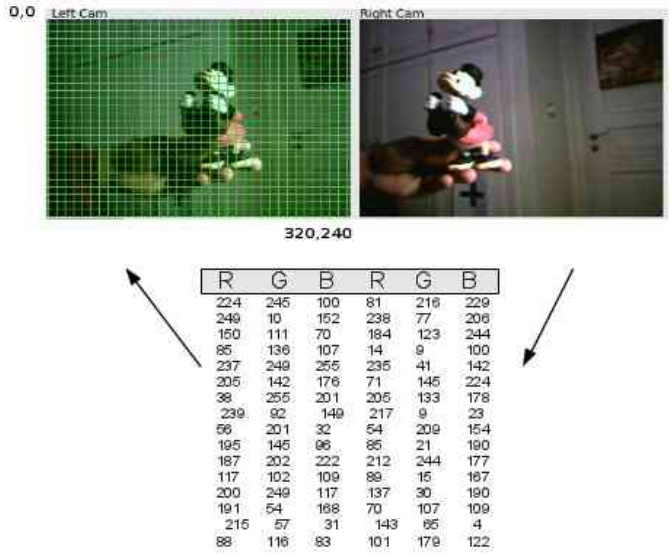


Illustration 2: The chicken and egg problem of an autonomous robot , that with its action changes its perception of the world , and with the changing perception of the world it changes its action..!

When beginning to make a system that sees , one can make many choices about the way with which to gather input. As nature teaches us , and by bringing to mind various insects and animals that have been optimized through a process of millions of years to see one might use anything from ultrasonic sounds , to millions small eyes of insects up to human stereoscopy. With the world represented through the camera being so chaotic , and as this project does not deal with a fixed environment in which to be operated , while also having economic restrictions applied the best choice was a human like stereoscopic camera input. It is true that commercial RGB+depth cameras such as Microsoft Kinect can bypass a very big portion of the computational complexity of this project , but they still have their own drawbacks. The stereoscopic setup wasn't chosen by accident by nature , and the nature of a robot that uses stereoscopic vision makes it closer to the human experience as a mode of viewing the world.

Illustration 3: What computers see



Trying to approach the computational limit of a dense stereoscopic method for two frames sized 320x240 pixels in order for a full search from an image patch sized 40x40 pixels on the left eye to all the possible matching patches along the epipolar line on the right eye , we have to make 320 x 320 x 240 / 40 = 24576000 /40 = 614400 operations in the worst case each time we get a depth map. In order to achieve a “human like” response time from the vision system this has to be done at a rate of 25 frames per second , or with a delay of 40 milliseconds per scan..
The number of operations per second increases exponentially as the image size becomes larger

SIZE	IMAGE RESOLUTION	OPERATIONS	OPERATIONS PER ms
QVGA	320 x 320 x 240 / 40	24,576,000	614,400 operations / ms
VGA	640 x 640 x 480 / 40	196,608,000	4,915,200 operations / ms
XGA	1024 x 1024 x 768 / 40	805,306,368	20,132,659 operations / ms
...	Other Configurations	...	...
WUXGA	1920 x 1920 x 1024 / 40	3,774,873,600	94,371,840 operations / ms

This exponential increase , of course , impacts all the algorithms used on the project , and for every operation there are numerous sub operations implied so the total maximum number of operations ends up being many times larger than the numbers on this table. All the algorithms on the other hand do a better job than this worst case scenario , and specifically the disparity mapping algorithm of GuarddoG , which is one of its novel aspects and is briefly presented in this text . To reduce the number of operations by design , and as an early measure to compensate for the cheap hardware that is used by the onboard computer the resolution of images used by default is QVGA (320x240 pixels) .

Manufacturing a physical stereo rig for the experiments which is perfectly aligned has a crucial effect on the calculations. Not only it increases computational efficiency and reduces errors but it also removes mathematical ambiguity about instances of the world that can be interpreted in many ways.. The relative position of the GuarddoG cameras is supposed to be constant and the two cameras always have a coplanar alignment with a fixed distance between the optical centers. The cameras are also never allowed to change their focus (nor could change it as they do not have an automatic focus control).

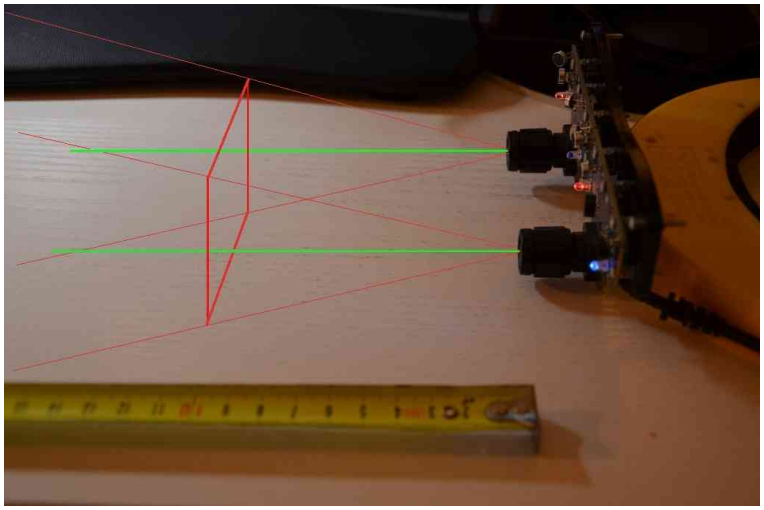


Illustration 4: The fixed parallel camera rig , that GuarddoG uses

STUB : You surely need to provide an overview of either a typical vision system or your vision system somewhere before start talking about "stages" and "pipelining"

To avoid re calculation and use of the CPU for reasons avoidable by a better designed algorithm , the whole vision library uses a pipelined architecture , so that the same image will not have to pass a processing stage twice once it enters and according to the needs of the Robot Hypervisior the different stages try to be combined , or operations stay pending for the next frame.

The first , mathematical , part of the project analyzes all the mathematical background of the algorithms and discusses performance from a complexity viewpoint .

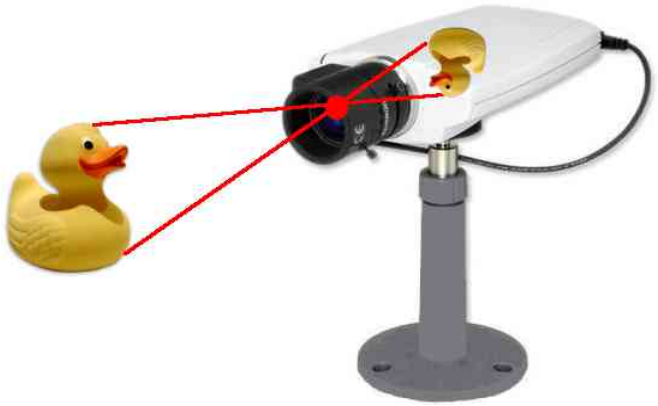
The second part focuses on software and technical details , along with performance statistics for different hardware setups and experimental results.

TODO Add a diagram of the processing pipeline Image → Rectification → Different filters → corner detection → face detection .. etc for coherence

*

Mathematical Framework

1.1.1 Camera Pinhole Model



A pinhole camera is a light capturing device without lens and a very small aperture. Regardless of the imaging sensor , the shutter system , or the integrated circuit on camera , it is fundamental to understand the physical model and how light gets projected on the sensor , in order to start to reverse engineer the physical process that creates the data we acquire.

The smaller the hole of the camera , the sharper the image gets , but as the hole size decreases , so does the total number of photons that pass through it, resulting in a dimmed image for short exposures.

The pinhole camera model applies to most consumer grade web cameras , but it can not be used without some additional processing overhead due to manufacturing inefficiencies that distort the projection on the camera sensor. These are discussed in the calibration and re sectioning parts of this document , that repair the distorted image making it fit to the ideal pinhole camera model described here.

*

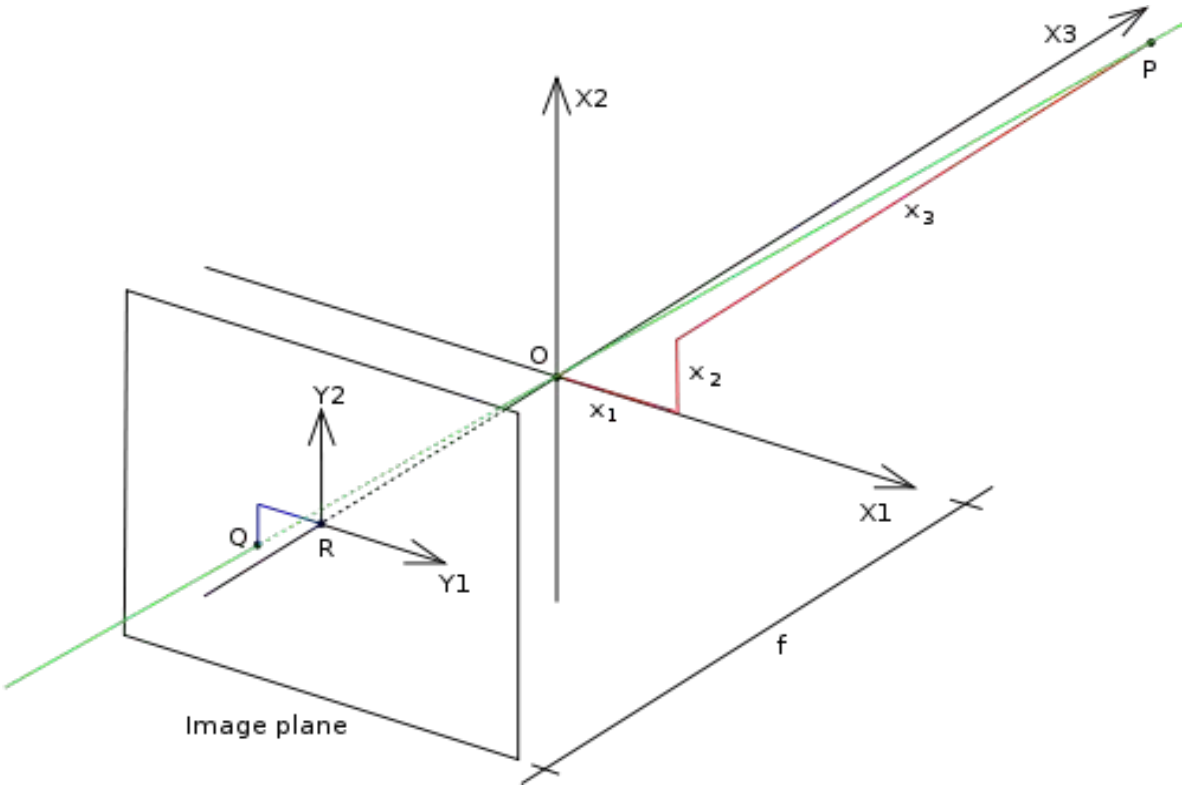


Illustration 5: The pinhole camera model , illustration from Wikipedia , public domain

TODO IMPROVE THIS :P

The point O is where the camera aperture is located , and the start of the axes. The three axes of the coordinate system are referred to as X1, X2, X3. Axis X3 is pointing in the viewing direction of the camera and is referred to as the optical axis, principal axis, or principal ray. The 3D plane which intersects with axes X1 and X2 is the front side of the camera, or principal plane.

An image plane where the 3D world is projected through the aperture of the camera. The image plane is parallel to axes X1 and X2 and is located at distance f from the origin O in the negative direction of the X3 axis. A practical implementation of a pinhole camera implies that the image plane is located such that it intersects the X3 axis at coordinate -f where f > 0. f is also referred to as the focal length of the pinhole camera.

A point R at the intersection of the optical axis and the image plane. This point is referred to as the principal point or image center.

A point P somewhere in the world at coordinate (x1,x2,x3) relative to the axes X1,X2,X3.

The projection line of point P into the camera. This is the green line which passes through point P and the point O.

The projection of point P onto the image plane, denoted Q. This point is given by the intersection of the projection line (green) and the image plane. In any practical situation we can assume that x3 > 0 which means that the intersection point is well defined.

There is also a 2D coordinate system in the image plane, with origin at R and with axes Y1 and Y2 which are parallel to X1 and X2, respectively. The coordinates of point Q relative to this coordinate system is (y1,y2).

*

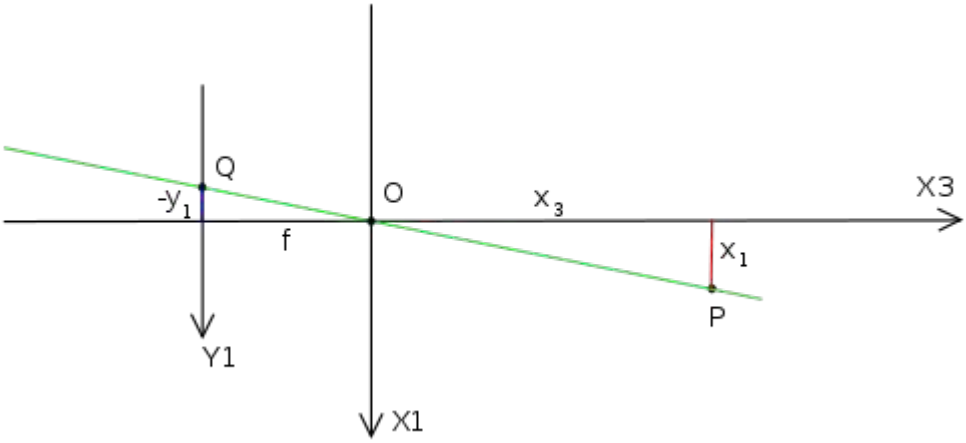


Illustration 6: The pinhole camera model , viewed from the side (from the X2 axis) , illustration from Wikipedia , public domain

The geometry of the pinhole camera viewed from the side , and on two dimensions. The calculations performed are based on similar triangles that are created with the point O as their intersection.

The mathematical equations that condense are the following :

$$\frac{-y_1}{f}=\frac{x_1}{x_3}\vee y_1=-f\,\frac{x_1}{x3}$$

$$\frac{-y_2}{f}=\frac{x_2}{x_3}\vee y_2=-f\,\frac{x_2}{x3}$$

$$(y_1 y_2)=\frac{-f}{x_3}(x_1 x_2)$$

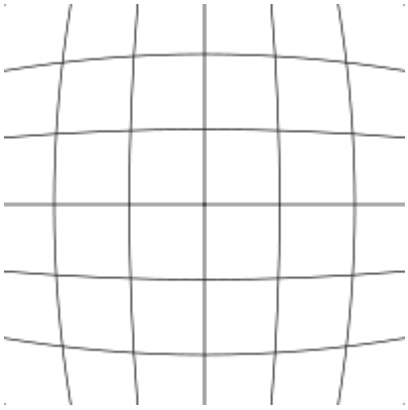
*

Mathematical Framework

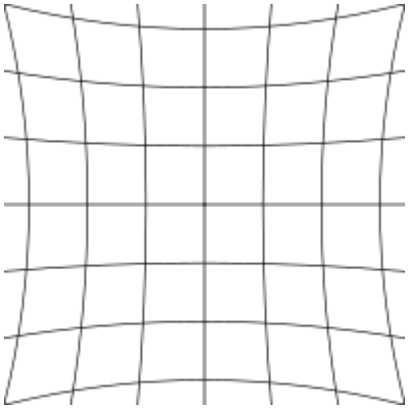
1.1.2 Camera Calibration

Having explained the underlying geometry behind the ideal pinhole camera model we need to adapt it to real cameras and their physical limits. In mathematics, it is possible to define a lens set that will introduce no distortions in the image captured. In practice, however, and due to manufacturing process inefficiencies two types of distortion occur . Radial distortion , caused by the shape of lens not being parabolic , and tangential distortion due to the assembly process of the camera in the factory.

Radial distortion causes a characteristic bending of straight lines as they get closer to the edges of the image and on systems that are heavily based on those images it can have a very detrimental effect on calculations that gets worse as the errors gradually accumulate in time. While disparity mapping algorithms can partly withstand this kind of distortion due to using a relatively large neighborhood of pixels that overall remains the same , point tracking and optical flow algorithms that estimate and track the camera position are very vulnerable to this kind of distortion.The reason this happens is because the relative positions of pixels change as they move to the edges and give wrong constraints for the system of equations to be solved later on.

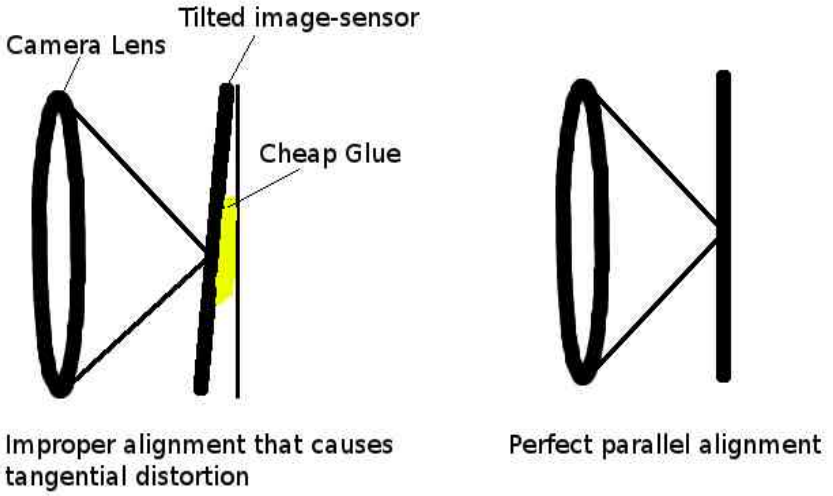


Barrel distortion



Pincushion distortion

Tangential Distortion on the other hand is a matter of misplacing the imaging sensor relatively to the lens (not a fully parallel placement) and therefore receiving a slightly skewed image.



Figuring out the way with witch a camera distorts the projection of the world on to its image sensor is called camera calibration. There are numerous methods and considerations to be taken into account to achieve calibration , even methods that gradually “auto calibrate” the raw input images without special patterns and objects or prior training of the algorithm. [citations needed] . GuarddoG uses a much more common approach , acquiring the intrinsic camera parameters (explained in the next chapter) using a fixed chessboard pattern that on the calibration stage is expected to be moved across the visible image so that enough snapshots of data can be acquired that may lead to a precise calculation . The method used by OpenCV is [citation needed] and the pattern is for GuarddoG is a 10x7 chessboard. After OpenCV finds the calibration parameters , they are stored in a file and used by GuarddoG's pipelining as the first processing step after an image is acquired.

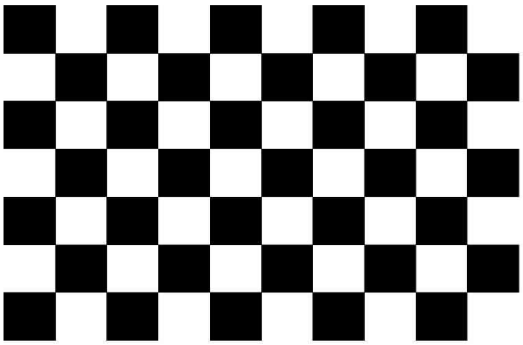


Illustration 7: OpenCV Chessboard 10x7 calibration pattern

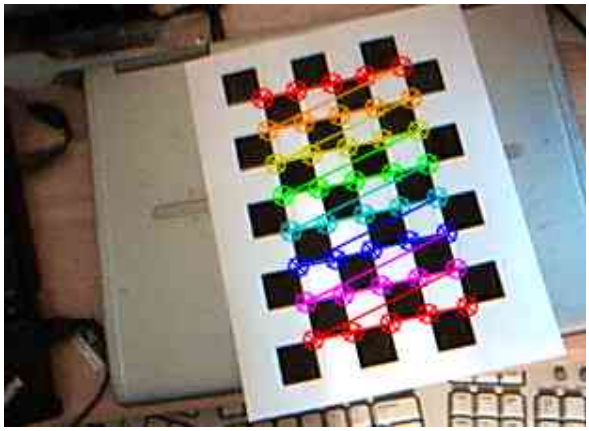


Illustration 8: Typical Detection Image Generated by OpenCV

The OpenCV implementation receives the corners between the chessboard blocks as inputs , which are extracted using a corner detector. First, it computes the initial intrinsic parameters and sets the distortion coefficients to zero. Afterwards using the Levenberg-Marquardt optimization algorithm [citation needed] the reprojection error is minimized until a stable parameter set is found.

A thing that is worth to be mentioned is that the cameras used by GuarddoG are graded by the manufacturer to have a less than 5% distortion rate and the algorithms work sufficiently well even when input is uncalibrated.

The method was conceived by Zhang [citation needed] and Sturm [citation needed]

Illustration 9: Raw images received from the cameras and their calibrated equivalent (the distortion parameters are exaggerated to better show the way calibration alters the input images)

*

Mathematical Framework

1.1.3 Image Rectification

Camera resectioning is the process of using the true parameters of the camera , after calibrating it to produce a correct photograph or video stream of images. Usually, the camera parameters are represented in a 3 × 4 matrix called the camera matrix , and from the pinhole model we get the following equations for a perfect undistorted lens.

PRETTY BIG TODO HERE symbols are incoherently and inconsistently presented. No connection is made to most parameters and no explanation is given below. Additionally, γ (referenced in the text) is not present in any equation. What are x? What are x'? what are X? This part needs a thorough rewriting and extension. *clear, concise and accurate.*

$$s\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} f_x & \gamma & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_1 \\ r_{31} & r_{32} & r_{33} & t_1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = R \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} + t$$

$$x' = x/z$$
$$y' = y/z$$
$$u = f_x x' + c_x$$
$$v = f_y y' + c_y$$

Where u , v are the undistorted pixels , fx and fy is focal length related to the pixel distance ratio [fx = mx * f , fy = my * f , (mx and my are determined by the physical dimensions of the pixel elements on the camera sensor)] , γ is a skew coefficient between axis x and y and for most contemporary imaging sensors is 0 and cx , cy are the coordinates of the principal point where the camera center (ideally at the center of the image) lies.

Radial and tangential distortion can be included to the model using the following equations

$$s\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} r_{11} & r_{11} & r_{11} & t_1 \\ r_{11} & r_{11} & r_{11} & t_1 \\ r_{11} & r_{11} & r_{11} & t_1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = R \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} + t$$

$$x' = x/z$$
$$y' = y/z$$
$$r^2 = x'^2 + y'^2$$
$$x'' = x'\left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6\right) + 2\, p_1 x' y' + p_2\left(r^2 + 2\mathrm{x}'^2\right)$$
$$y'' = y'\left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6\right) + p_1\left(r^2 + 2\mathrm{y}'^2\right) + 2\mathrm{p}_2 x' y'$$
$$u = f_x x'' + c_x$$
$$v = f_y y'' + c_y$$

Radial distortion is modeled by the k_1 , k_2 and k_3 (which is usually zero) coefficients and tangential distortion is modeled by p_1 and p_2.

Since re sectioning the image must be done for every frame received from the usb cameras (which serve images @ 120 Hz) and since most camera chips don't offer a hardware interface for passing the distorition parameters to the local integrated circuit so it can perform this kind of image processing with out involving the main processor , a fast technique must be applied to avoid calculating all these numbers on every program loop.

Since the camera doesn't change its focus settings , the distortion parameters are always the same. We can use this knowledge to our advantage generating a precalculation frame that has pointers to the calibrated positions as its elements after acquiring the distortion parameters.
That way , the expensive task of computing the formulas mentioned above happens only once and the least possible overhead is added to the pipeline process (around 500 microseconds per frame on the main development computer , hardware details are on the second part of this text).

This tactic is followed both by the OpenCV implementation , as well as the GuarddoG RoboVision stack.

*

Mathematical Framework

1.2.0 Image Processing

Digital cameras are devices that capture the light that the universe reflects on their sensor. The general problem most vision algorithms try to solve is guessing what kind of a world reflects the light in that way. The algorithms presented here are building blocks that gradually transform the raw RGB input into more computationally meaningful representations .

TODO REPHRASE EXPLAIN Convolution is a mathematical operation applied to sets of values that "redistributes" them according to weights. The carrier of the weights is called a convolution matrix and it is typically a 3x3 kernel with one of the values , usually the middle one , called an anchor point.

The values transformed by the convolution matrix are the red , green and blue light intensities of the pixels retrieved from the image sensor. In the following example we assume a 3x3 kernel and a monochrome image sensor that captured 9x6 pixels. The kernel is passed left to right and up to down until all of the elements are changed. GuarddoG uses Gaussian Blur , Sobel and Second Derivative Convolution kernels that follow with example images.

1	1	1
1	**1**	1
1	1	1

3X3 Convolution Kernel
Divisor 9

As the anchor of the kernel passes from each element of the image array the value (marked blue) gets replaced by the addition of the neighboring elements multiplied with the corresponding kernel element.

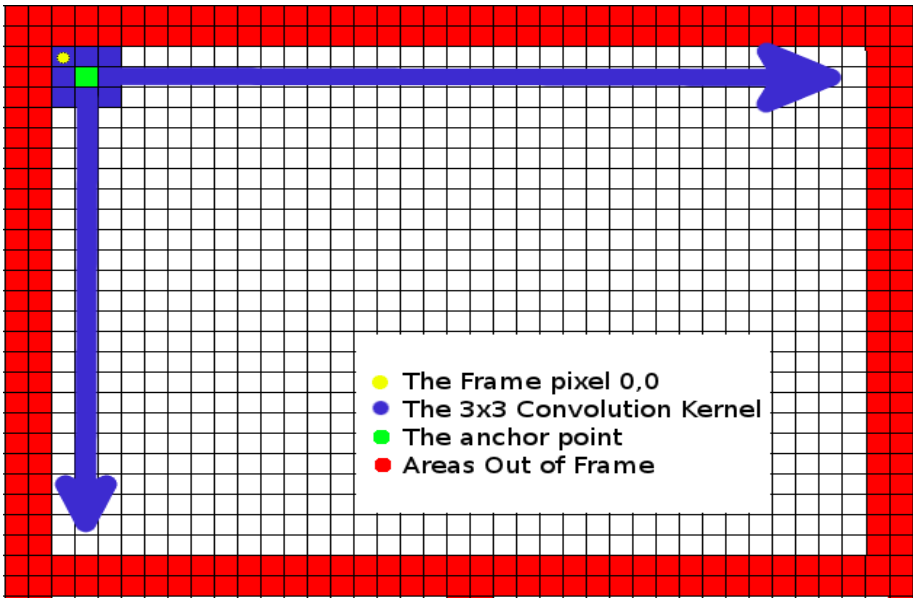
$$H(x,y)=\sum_{i=0}^{Mi-1}\sum_{j=0}^{Mj-1} I(x+i-a_i, y+j-a_j)G(i,j)$$

The anchor element on the light intensities array will become
(1x90+1x80 +1x70+1x90+1*80+1*70+
1x90 + 1x80 + 1x70) / 9 which is 80

9 x 6 Original Light Intensities Captured

90	80	70	90	80	70	90	80	70
90	**80**	70	90	80	70	90	80	70
90	80	70	90	80	70	90	80	70
90	80	70	90	80	70	90	80	70
90	80	70	90	80	70	90	80	70
90	80	70	90	80	70	90	80	70

An important thing to be noted Is that values on the edges of the array (marked orange) can not be correctly calculated as not all neighboring elements exist , common solutions for this is zero padding , using a different divisor to compensate for the missing elements or skipping the elements that can not be calculated correctlly .



BLUR FILTER

1	1	1
1	**1**	1
1	1	1

Divisor 9

FIRST-ORDER DERIVATIVE

1	-2	1
2	**-4**	2
1	-2	1

Divisor 1

SECOND-ORDER DERIVATIVE

-1	0	1
0	**0**	0
1	0	-1

Divisor 3

Blur filters evens out the colors on an input image using taking the median color value of the surrounding area . Blurring is a common operation by vision software mainly used since due to the fact that image sensors retrieve pixels that suffer from noise , these noise spikes are reduced therefore leading to better edge and corner detection.

The First-order derivative operator acts as a differentiation operator , resulting in an output that only responds to “change” of colors and ignores similar colored areas. Thus it is very useful as it reduces the image to its more unique parts , its edges .

The second-order derivative operator acts also as a differentiation operator , resulting in an output that only responds to “change” of “change” of colors (second order) and ignores similar colored areas while also having a better reaction to sudden spikes on the color frequency. Its output also reveals the image edges but is much more stable than the sobel operator .

Palette reduction reduces the total number of possible tones that one pixel can take from 16581375 on an 24bit color depth (255 * 255 * 255) to an other given number. Reducing the total possible colors causes similarly colored pixels to fall into the same color bin. This can be leveraged to make the datasets more resistant to noise. Conversion from a full color palette to a monochrome image , is a very common operation on computer vision algorithms.

Thresholding can be used as a filter extension to apply a high (or low) pass bound on an incoming signal and discard pixels that do not match the criteria. This is generally done after edge detection operations to reduce false output caused by noise.

Histograms are produced by counting the total instances of the different colors on an area of an input. They can give a good general idea about an image , such as its brightness, color distribution and are used in guarddog as a fast discarding mechanism for regions of the image when performing disparity mapping .

The miscellaneous image processing operations used by GuarddoG are mentioned in the table that follows

NAME	OPERATIONS	DESCRIPTION
Gaussian Blur	9 * Height*Width	Blurring input image to reduce noise
Sobel edge det.	9 * Height*Width	Edge Detection
Second-order de.	4 * Height*Width	Edge Detection
Palette Reduce	Height*Width	Group color frequencies together to reduce them
Threshold Image	Height*Width	Discard information that may be subject to noise
Histogram	Height*Width	Calculate number of pixels having the same color

*

Mathematical Framework

1.2.1 Corner and Feature Detection

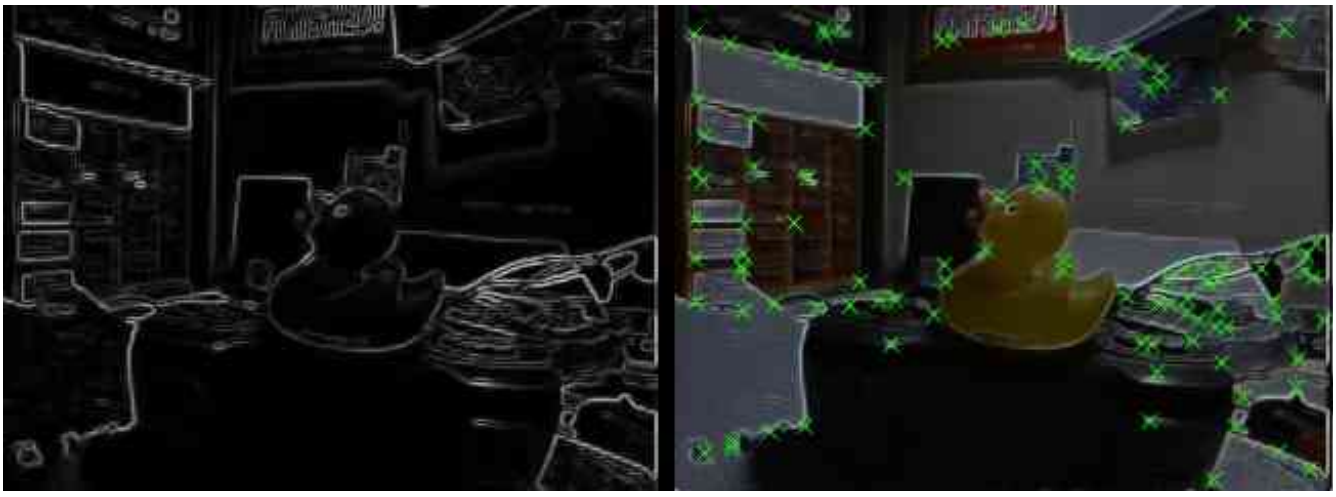


Illustration 10: Left : An incoming image after passing through First-order Derivative Edge detection , Right : The corners detected , highlighted with green X marks

After performing the various image processing steps mentioned above , to start moving away from the image as a raw array of color frequencies and into a better representation , we must focus on specific points on it that stand out and have unique characteristics . These points are called features or salient points and can be picked using a multitude of criteria. The features used by GuarddoG are corners and offer a good performance and quality trade-off. They are both relatively inexpensive computationally to extract and also exhibit persistence between frames produced from small movement of the camera , in normal lighting conditions.

Some feature detectors such as SURF [citation needed] pick points that not necessarily lie in a corner , but nevertheless have a large eigen value and are scale and rotation resistant.The feature detector used by GuarddoG is called FAST [citation needed] and it detects corners by casting a circle of 16 points on edge points **TODO** and comparing them to a threshold. **TODO EXPLAIN FAST**

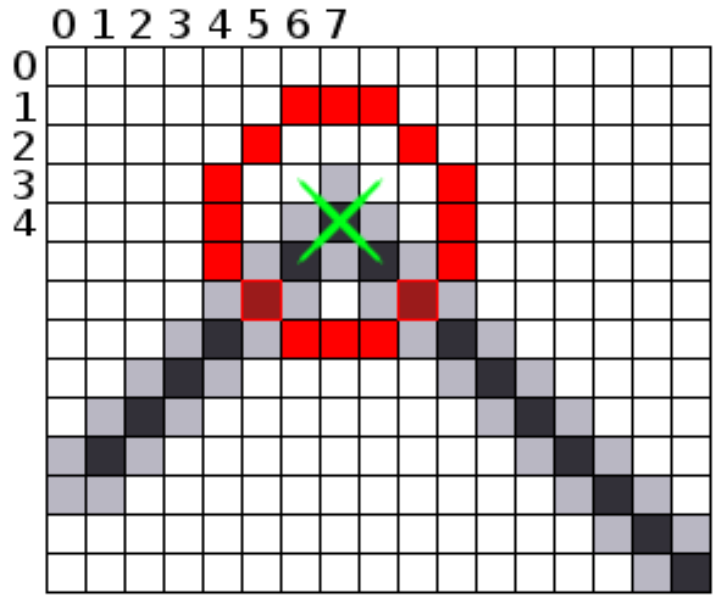


Illustration 11: An instance of the algorithm detecting a feature (corner) by sampling the 16 points of the circle casted around the point 7,4 using the FAST algorithm. The detector finds two similar colored points and succeeds in detecting the corner.

A second feature detector that is used as an alternative and performs adequately is the OpenCV cvGoodFeaturesToTrack method by Shi and Tomasi [citation needed]. which utilizes a second-derivative filtered image and then calling cvCornerMinEigenVal and cvCornerEigenValsAndVecs picks the minimum eigen values under a threshold and provides again a list of good features on a reasonable computational cost.

TODO explain SHI AND TOMASI Image motion , eigenvalues etc here :P

$$M = \begin{pmatrix} \sum (dI/dx)^2 & \sum (dI/dx * dI/dy) \\ \sum (dI/dx * dI/dy) & \sum (dI/dy)^2 \end{pmatrix}$$

$$\text{The minimal eigenvalue is then picked since:}$$
$$x_1, y_1 \text{ corresponds with } \lambda_1$$
$$x_2, y_2 \text{ corresponds with } \lambda_2$$

TODO explain Sub pixel corners here.. :P

*

Mathematical Framework

1.2.2 Template Matching and Integral Images

MAJOR TODOS HERE :'(: The image processing pipeline receives raw input from the cameras and produces images that are

After image processing is finished producing "versions" of the data that reveal different aspects of the input images , the next technique performed by guarddog is called Template Matching.
There are numerous criteria that can be used to compare two image parts and decide if they match.

GuarddoG uses a combination of pyramid segmentation , feature and template based matching across different templates to achieve high performance without sacrificing result quality. To this end the use of integral images speeds up and greatly improves the algorithm (performance-wise) .

The most simple and computationally efficient method for comparing two blocks of pixels is named SAD (Sum of Absolute Differences) and is basically the following equation.

$$SAD = \sum_{x=0}^{width} \sum_{y=0}^{height} \left| (image1[x][y] - image2[x][y]) \right|$$

This operation can be hardware accelerated on MMX and SSE2 instruction capable CPUs and thus is very lite weight. Although there are other metrics to find out if two image blocks match (and how similar they are) such as MSE (Mean Squared Error) , SATD (Sum of absolute transformed differences) , Normalized Cross Correlation (NCC) and other even more complex methods.

To make up for quality loss , while keeping the increased performance that SAD offers guarddog compares different "versions" of the patches that resulted mainly from convolution operations on the original data. That way the computational cost is moved from the block matching operation that can be performed millions of times (especially in large images) and does not take a guaranteed time to converting the image itself which has a fixed sized.

The different SAD results are then combined into a single value according to weights to compensate for the different range of values in each of the sub images. In order to further skip unneeded calculations a local histogram is used as a threshold that can completely avoid calculations if the 2 image blocks bear no resemblance at all (i.e. one is completely white and the other completely black)

*

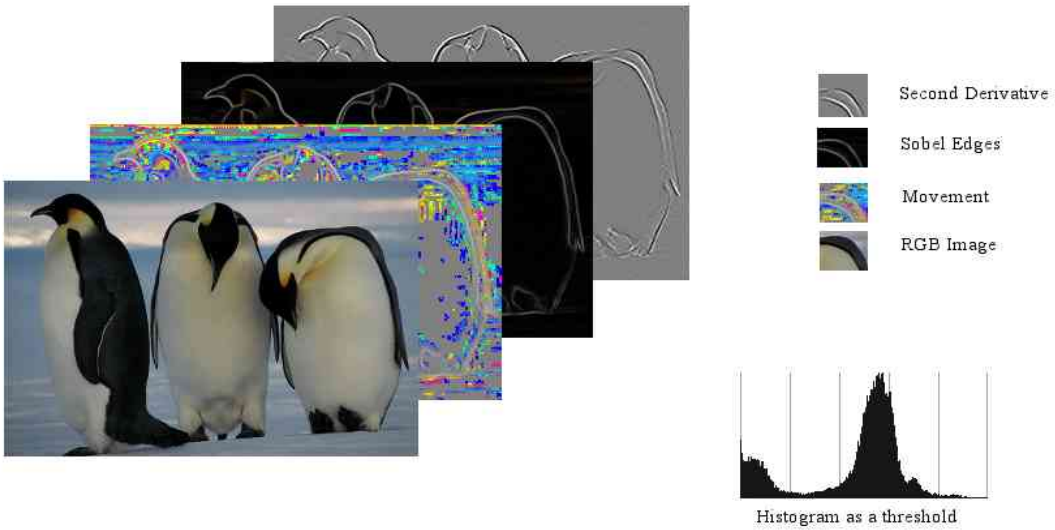


Illustration 12: The things taken into account when comparing patches

Before going into more detail about the template matching function , another useful representation for massive calculations on images is called integral images , or summed area tables.

$$I(x', y') = \sum_{x=0}^{x'} \sum_{y=0}^{y'} (image[x][y])$$

Once the table that every x,y has as an element the I(x,y) is created . We can skip a huge number of adding operations for an arbitrary area of the image (limited only by the maximum value of an integer on the machine). Any block addition operation is thus reduced to 4 operations.

$$\sum_{x=x1}^{x2} \sum_{y=y1}^{y2} image[x][y] = I(x1, y1) + I(x2, y2) - I(x2, y1) - I(x1, y2)$$

The resulting operation is not SAD because the subtraction does not produce an absolute difference on each pixel , the resulting operation is a plain Sum of Differences which is an even worse metric than SAD but it has such a big performance impact , that when used in conjunction with the sub images mentioned before it can make dense disparity mapping feasible , and when used in small enough areas provides good overall results.

Instead of
|image1[x1][y1] – image2[x1][y1]| + |image1[x2][y1] – image2[x2][y1]| + … + |image1[xN][yN] – image2[xN][yN]| we have

image1[x1][y1] + image1[x2][y1] + … + image1[xN][yN] - (image2[x1][y1] + image2[x2][y1] + … + image2[xN][yN]) which is the same with

image1[x1][y1] – image2[x1][y1] + image1[x2][y1] – image2[x2][y1] + … +
image1[xN][yN] – image2[xN][yN]

*

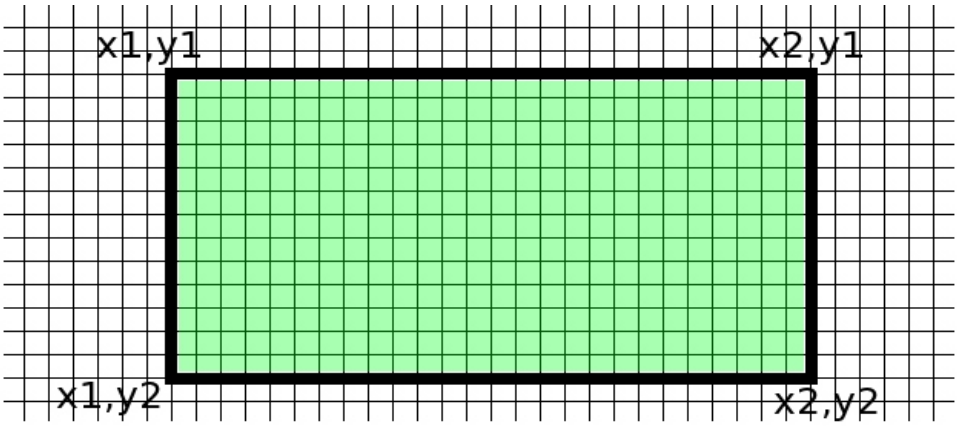


Illustration 13: Typically , we find the sum of the green area by adding all the pixels in it performing (x2-x1)(y2-y1) operations*

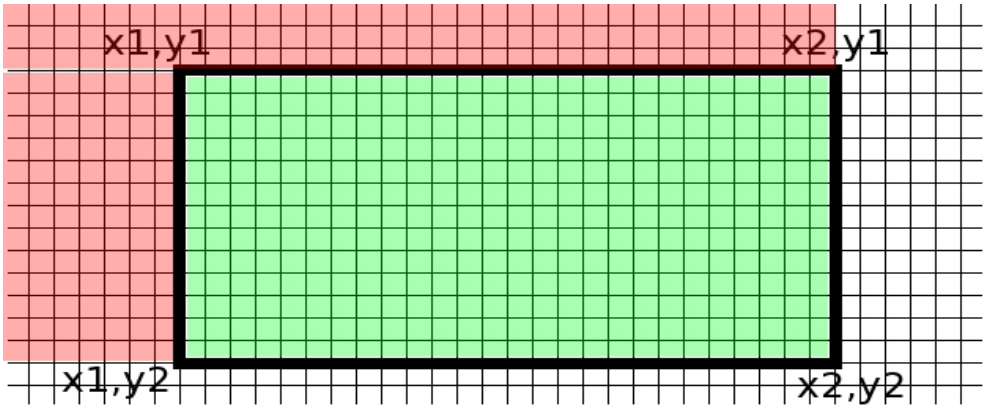


Illustration 14: We can find the sum of the green area by performing 4 operations , I(x1,y1)+I(x2,y2)-I(x1,y2)-I(x2,y1) provided we have first calculated the integral array I

Illustration 15: A SAD metric returns total mismatch of these two blocks. An addition of differences metric (not absolute) such as the integral imaging technique described before returns a total match of the two image blocks

*

Mathematical Framework

1.2.2 HAAR Wavelet Face Detection

Haar-like features are digital image features used in object recognition. Their similarity with Haar wavelets is what gave them their name and they were used in the first real-time face detector. GuarddoG uses the OpenCV implementation of a haar cascade detector with an appropriate training file , while the implementation is largely based on the Viola Jones Face detection algorithm (Robust Real-time Object Detection) .

There are many approaches to face detection and as a refinement recogniton , including eigen faces (M. Turk and A. Pentland (1991). "Face recognition using eigenfaces") , image pyramids (Neural Network-Based Face Detection, 1998), and mixed methods (W. Kienzle, G. Bakir, M. Franz and B. Scholkopf: Face Detection - Efficient and Rank Deficient. In: Advances in Neural Information Processing Systems 17, pg. 673-680, 2005.) , each of which have their own pros and cons .

The reason for choosing a Haar Wavelet based face detection is that it is again accelerated by integral images and thus it can fit in nicely in the pipeline of the vision processor algorithm while performing incredibly well for upright faces that are the only kinds of faces that a small indoor robot should respond to.

A Haar Wavelet is a small region that consists of two areas , one black (low value) and one white (high value) . As a pattern it can have a lot of iterations , and the ones displayed bellow are the most common ones.

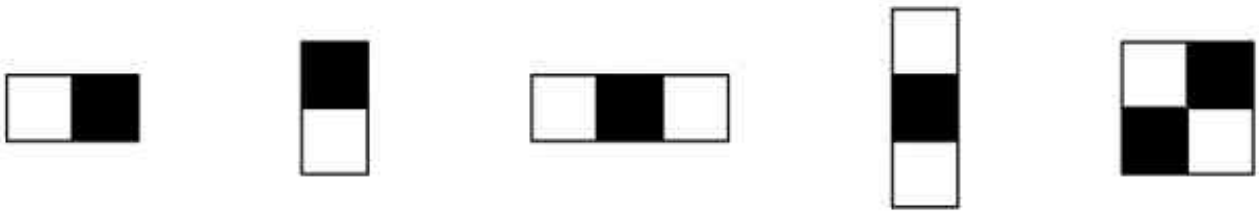


Illustration 16: Common Haar Wavelets

As the patterns , which may well be more complicated , pass over the image they are compared to the underlying color frequencies . **TODO Is a multiscale function basis and frequency is generally determined by its scale, not the direction. As many image bases, it forms a laplacian pyramid where its subscale is the subsampled low-resolution version (pre-filtered) of the signal plus a number of basis-projected versions of the signal for the high freq components of that level. Fir instance the HWT of an image at a given (frquency) level produces a low freq image (LL, smoothed and subsampled) + a LH, a HL and a HH component corresponding to the 1st, 2nd and 5th pattern you describe in the image**. , if a color frequency is low and it is under the low (black) part of the pattern it is also registered. That way the image is transformed to an array of numbers and using a correct cascade of haar wavelets appropriate to the size , and orientation of detection this can be used as a tool for generic object detection (Papageorgiou, Oren and Poggio, "A general framework for object detection")

The viola and jones detector basically works by discarding portions of the image as “non-faces”. To achieve this it uses a haar cascade constructed by an adaptive boosting machine learning algorithm , popularly coined as AdaBoost [citation needed] to pick the best features from training images and drive the face detector.

A sample detection image is the following , using the OpenCV cvHaarDetectObjects implementation and the haarcascade_frontalface_alt.xml cascade. The good response rate of this method was also confirmed by realtime usage on the International Fair of Thessaloniki 2011 where GuarddoG collected over 4500 faces on a course of a week with a very low false detection rate.

Illustration 17: Left : Sample face detected (marked by purple circle) , features detected by the corner detector explained at topic 1.2.1 (marked with yellow dots). Right : a possible HAAR cascade manually created for dramatization of the way HAAR Cascades digitize images and thus serve well for two dimensional face and object detection.

Illustration 18: Random faces out of a 4500+ faces collection gathered during IFT 2011

*

Mathematical Framework

1.3.0 World Coordinate System

UNDER CONSTRUCTION :P

*

Mathematical Framework

1.3.1 Epipolar Geometry

Assuming a rectified input of two pinhole cameras with a known alignment , viewing a 3D scene, there are some geometric relations about the projections of 3D points among them.

Both cameras see the world from a different viewpoint , and while the projected image is different there are some geometric constraints that can be leveraged to be used for disparity mapping , which will be analyzed later.

GuarddoG's cameras are positioned in parallel so the epipolar plane forms a parallel line from frame to frame. This configuration is used to reduce errors caused by incorrect calibration and reduce the overall complexity of the algorithms that are based on matching parts from one image to the other.

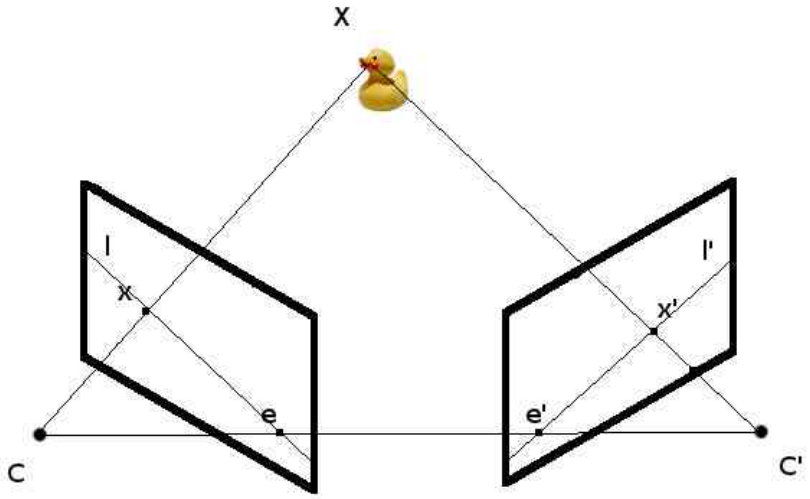


Illustration 19: A non parallel allignment where we can see highlighted the camera centers C and C' and the baseline that goes through both of them , the epipoles e and e' are the intersections of the image planes with the base line , the projection of the point X at x and x' when connected to the epipoles gives us the epipolar lines l and l'

*

With the parallel setup the two projection images are essentially beeing produced by a translation of the camera center parallel to the image plane. This results in the points e and e' beeing in infinity , and the baseline never touching the image plane (since it is parallel to it)

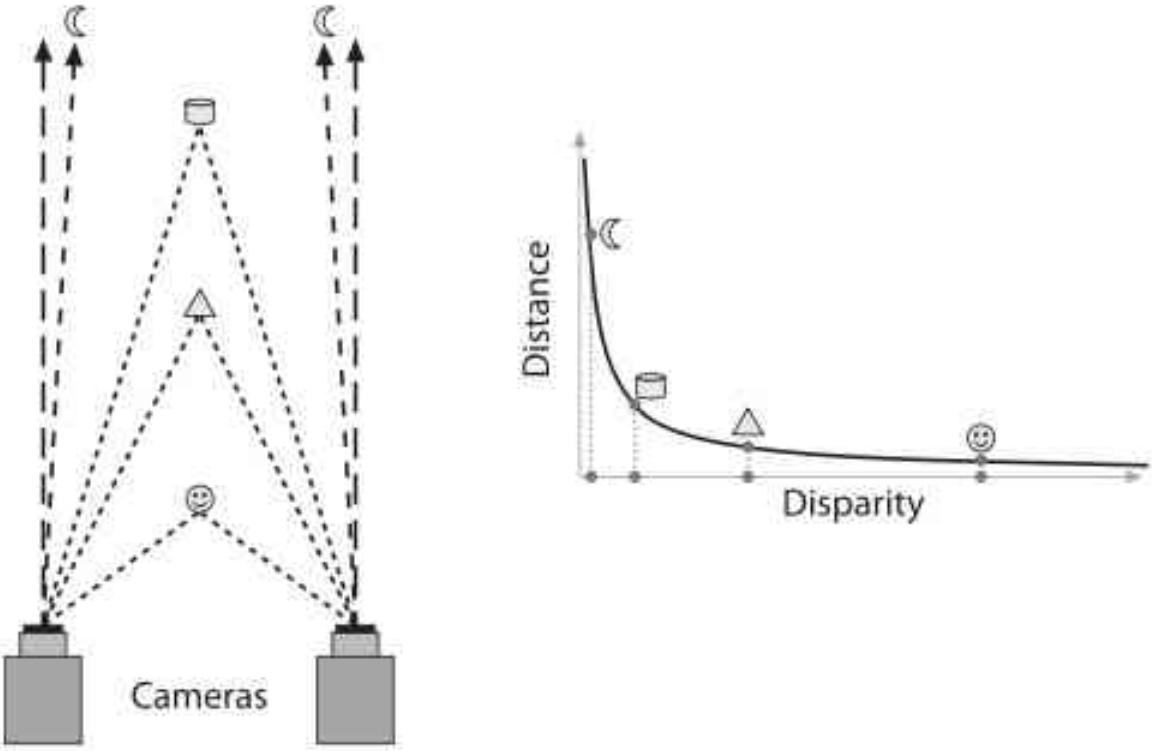
Since the projection of all the points on the line from C to X lay on the l' epipolar line , to find out the projection of the ducks head on the right image we can search only the area having the same height coordinates from image to image and that makes disparity mapping practical for computation.

From frame to frame (in the axis of time and not space) since we have lots of different types of movements and combinations of rotations and translations epipolar geometry is once again useful for the calculation of the fundamental matrix since it provides the fundamental matrix equation constraint. GuarddoG though uses homographies and not the fundamental matrix for camera pose estimation since the 3d points typically have a higher error rate and lower coverage than the corners that are used for the homographies.

*

Mathematical Framework

1.3.2 Disparity Mapping



UNDER CONSTRUCTION :P

*

Mathematical Framework

1.4.0 RANSAC

UNDER CONSTRUCTION :P

*

Mathematical Framework

1.4.1 Optical Flow

UNDER CONSTRUCTION :P

*

Mathematical Framework

1.1.0 Homography

UNDER CONSTRUCTION :P

*

Mathematical Framework

1.1.0 Simultaneous localization and mapping

UNDER CONSTRUCTION :P

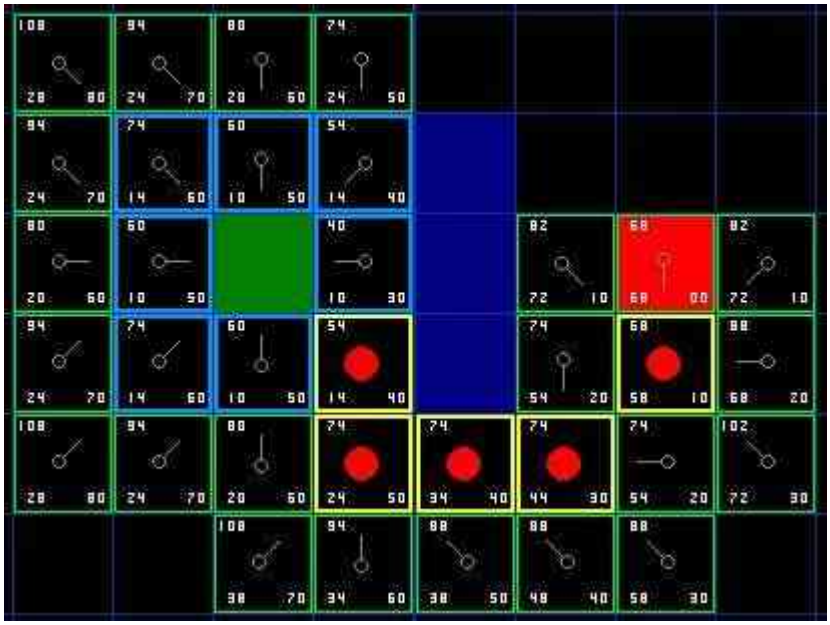
*

Mathematical Framework

1.1.0 A* Path Finding

Assuming a two dimensional map acquired by the operations above , and a stable track of the position of the robot , there is a need for an algorithm to perform path finding , in order for the robot to be able to reach a target position and dynamically change its course when new obstacles are detected. The algorithm used by this project for this kind of functionality is A* , an extension of Dijkstra's graph search algorithm. Successful path finding is very critical because it means less battery drain due to unnecessary movements and better performance as a guard.

A* uses a heuristic that has to never over-estimate the route cost , and such a heuristic is the Manhattan distance that is commonly used by many implementations.
The complexity of the algorithm is $| h(x) - h^*(x) | = O(\log h^*(x))$ where h is the heuristic used.
The cost of the algorithm for each new node is calculated using $f(n) = g(n) + h(n)$ where g is the cost of the transition to the new node and h the heuristic for the transition to the goal node.
A* is thus admissible since adding g which is an exact estimation of the distance from the source node to the optimistic heuristic since will always make the algorithm seek the solution with the lowest possible cost.



A* Algorithm

```
OPEN SET =  START NODE
CLOSED SET = EMPTY
while the node with the lowest cost in OPEN SET is not the GOAL NODE:
  current = remove lowest rank item from OPEN SET
  add current to CLOSED SET
  for neighbors of current:
    cost = g(current) + movementcost(current, neighbor)
    if neighbor in OPEN and cost less than g(neighbor):
      remove neighbor from OPEN, /*new path is better*/
    if neighbor in CLOSED SET and cost less than g(neighbor):
      remove neighbor from CLOSED SET
    if neighbor not in OPEN SET and neighbor not in CLOSED SET:
        set g(neighbor) to cost
        add neighbor to OPEN SET
        set priority queue rank to g(neighbor) + h(neighbor)
        set neighbor's parent to current

Reconstruct path following parent pointers from goal to start
```

One of the shortcomings of a raw implementation of an uncustomized A* algorithm is that in the real world diagonal movement is a little further away than than horizontal (pythagorean theorem) . The result is that returned paths can be “non optimal” for a real world moving robot. Added to this problem comes the fact that in physical movement one tends to hold a course turning as little as it is possible. A* can provide an optimal solution that has many turns , but this will take more time for the robot to be traversed. The solution to this problem is keeping the heading of the robot as an information vector on every opened node and adding an extra weight when turns are made , while also adding an extra weight when performing diagonal movement to balance them.

The final element needed is a way to represent uncertainty about the mapped obstacles since there may be errors in the input , not only caused by “mis-detection of obstacles” but also by the the lack of detail of the map since an area of 200 m^2 quantized at a scale of 10 cm^2 per block results in an array sized 2000x1000 that cannot reflect the full complexity of the scene.

Using these modifications , the output becomes better but there is a further improvement that can be achieved by using the largest possible straight paths to connect sub regions of the A* paths. Doing that the turning maneuvers of the robot are reduced to the fewest possible. To achieve that , after a path has been extracted ,instead of reconstructing the path following the parnet pointers we use a second pass algorithm runs which casts a line (using Bresenham's line algorithm) from the last step of the path to all the previous ones until an obstacle is detected. The previous point before the obstacle is then marked as connected to the first one and the algorithm continues until the source node is connected. This improves the operation of the robot . This could also be improved in the future to use odometer based curves instead of point to point turning , something that would also make the movement of GuarddoG seem more life like.

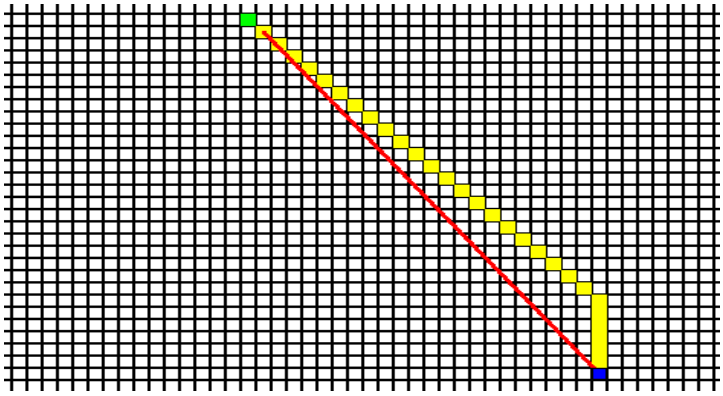


Illustration 20: The problems that may occur using an uncustomized A Algorithm , and how they are corrected*

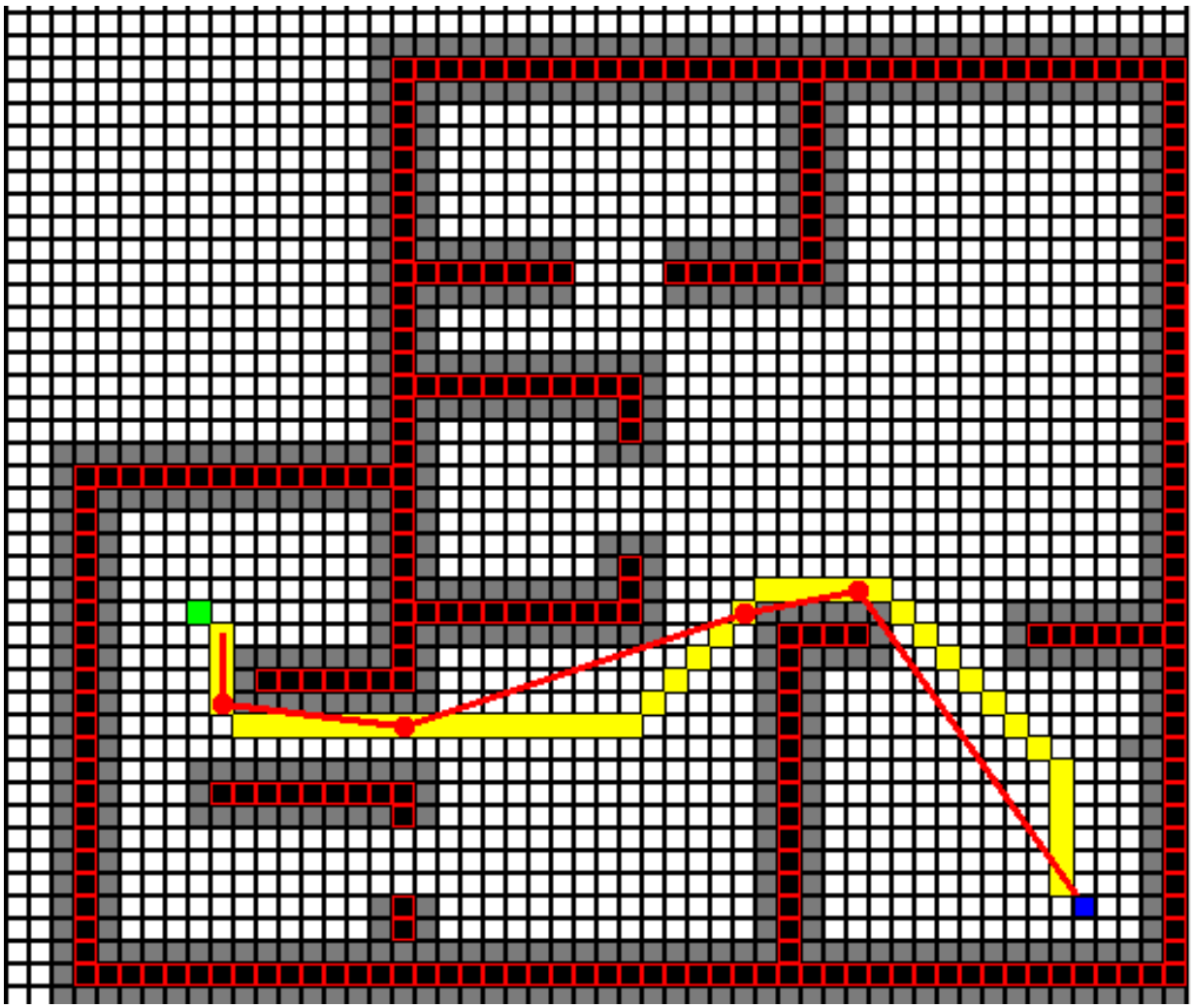


Illustration 21: The green block is the source , the blue the target , red/black blocks are obstacles and gray areas , areas of uncertainty. The yellow path is the one that A returns and the red line the compressed path for as little turning as possible.*

*

Mathematical Framework

1.1.0 First-order logic and a Wumpus like world

Or a Shakey one also taken from Russel & Norvig

UNDER CONSTRUCTION :P

*

Hardware

2.1.0 Camera Sensors

An active-pixel sensor (APS) is an image sensor consisting of an integrated circuit containing an array of pixel sensors, each pixel containing a photodetector and an active amplifier. There are many types of active pixel sensors including the CMOS APS used most commonly in cell phone cameras, web cameras and in some DSLRs. Such an image sensor is produced by a CMOS process (and is hence also known as a CMOS sensor), and has emerged as an alternative to charge-coupled device (CCD) imager sensors.

The term active pixel sensor was coined by Tsutomu Nakamura who worked on the Charge Modulation Device active pixel sensor at Olympus,[4] and more broadly defined by Eric Fossum in a 1993 paper.[5]

Image sensor elements with in-pixel amplifiers were described by Noble in 1968,[6] by Chamberlain in 1969,[7] and by Weimer et al. in 1969,[8] at a time when passive-pixel sensors – that is, pixel sensors without their own amplifiers – were being investigated as a solid-state alternative to vacuum-tube imaging devices. The MOS passive-pixel sensor used just a simple switch in the pixel to read out the photodiode integrated charge.[9] Pixels were arrayed in a two-dimensional structure, with access enable wire shared by pixels in the same row, and output wire shared by column. At the end of each column was an amplifier. Passive-pixel sensors suffered from many limitations, such as high noise, slow readout, and lack of scalability. The addition of an amplifier to each pixel addressed these problems, and resulted in the creation of the active-pixel sensor. Noble in 1968 and Chamberlain in 1969 created sensor arrays with active MOS readout amplifiers per pixel, in essentially the modern three-transistor configuration. The CCD was invented in 1970 at Bell Labs. Because the MOS process was so variable and MOS transistors had characteristics that changed over time (Vt instability), the CCD's charge-domain operation was more manufacturable and quickly eclipsed MOS passive and active pixel sensors. A low-resolution "mostly digital" N-channel MOSFET imager with intra-pixel amplification, for an optical mouse application, was demonstrated in 1981.[10]

Another type of active pixel sensor is the hybrid infrared focal plane array (IRFPA) designed to operate at cryogenic temperatures in the infrared spectrum. The devices are two chips that are put together like a sandwich: one chip contains detector elements made in InGaAs or HgCdTe, and the other chip is typically made of silicon and is used to readout the photodetectors. The exact date of origin of these devices is classified, but by the mid-1980s they were in widespread use. By the late 1980s and early 1990s, the CMOS process was well established as a well controlled stable process and was the baseline process for almost all logic and microprocessors. There was a resurgence in the use of passive-pixel sensors for low-end imaging applications,[11] and active-pixel sensors for low-resolution high-function applications such as retina simulation[12] and high energy particle detector.[13] However, CCDs continued to have much lower temporal noise and fixed-pattern noise and were the dominant technology for consumer applications such as camcorders as well as for broadcast cameras, where they were displacing video camera tubes.

In 1995, personnel from JPL founded Photobit Corp., who continued to develop and commercialize APS technology for a number of applications, such as web cams, high speed and motion capture cameras, digital radiography, endoscopy (pill) cameras, DSLRs and of course, camera-phones. Many other small image sensor companies also sprang to life shortly thereafter due to the accessibility of the CMOS process and all quickly adopted the active pixel sensor approach.

The cameras used by GuarddoG are based on the OV7720/OV7221 CMOS VGA (640x480) CAMERACHIP

Sensor , and are cheap and easy to find as they are the camera system used by the Playstation 3 Gaming Console

Camera Sensor Key Specifications

Array Size	640 x 480
Power Supply Digital Core Voltage	1.8VDC + 10%
Power Supply Analog Voltage	3.0V to 3.3V
Power Supply I/O Voltage	1.7V to 3.3V
Power Requirements - Active	120 mW typical (60 fps VGA, YUV)
Power Requirements - Standby	< 20 µA
Temperature Range	-20°C to +70°C
Output Format (8-bit)	• YUV/YCbCr 4:2:2 • RGB565/555/444 • GRB 4:2:2 • Raw RGB Data
Lens Size	1/4"
Max Image Transfer Rate	60 fps for VGA
Scan Mode	Progressive
Electronic Exposure	Up to 510:1 (for selected fps)
Pixel Size	6.0 µm x 6.0 µm
Fixed Pattern Noise	< 0.03% of VPEAK-TO-PEAK
Image Area	3984 µm x 2952 µm
Package Dimensions	5345 µm x 5265 µm

*

Hardware

2.1.0 Camera Synchronization

Stereo vision on a mobile robot is a non-trivial problem that traditionally requires expensive hardware-synchronized cameras. Because standard stereo reconstruction assumes that the images from the left and right cameras are captured from a common scene , any motion that occurs between the left and right cameras capturing frames is equivalent to a change in the stereo camera`s baseline. This change in baseline invalidates the system`s extrinsic calibration , causing the quality of the rectification to decrease and the distances to be distorted by the robot`s velocity.

Hardware synchronization , the process of forcing two or more cameras to share a common hardware clock , has been traditionally limited to the professional stereo vision systems such as Point Grey`s Bumblebee product line . Thankfully , the inexpensive Playstation Eye camera is built on the same high-end OmniVision OV7720 chipset that is comparable to those found in many machine vision cameras. These cameras can be hardware-synchronized using the exposed frame clock input (FSIN) and output (VSYNC) pins . By shorting one camera`s VSYNC pin to the others cameras FSIN pins the cameras are forced to share a common clock . To reduce the risk of a difference in ground potentials damaging the OV7720 delicate circuitry , each camera was also modified to share a common ground .

This hardware synchronization guarantees that all three cameras capture images simultaneously , but does not guarantee that the frames will travel retaining their synchronization on the USB .

Each camera has its own hardware clock and that means that in addition to the small distortion in space (due to optics) we have a small distortion in the fourth dimension , the axis of time. To tackle this problem guarddog uses cameras that have a very fast refresh rate of 120fps @ 320x240 pixels with a rewired shutter (FSIN , VSYNC pins) in order for synchronization on the hardware side of the camera snapshots. A secondary problem is that there is non uniform latency over the USB cable and the USB host controller . This is problem is combated using direct frame grabbing via V4L2 and zero-copy passing by pointer to the beginning of the image pipelining and static linkage of the libraries consisting of the project to reduce delays and overheads.

*

Hardware

2.1.0 USB Host

A USB system has an asymmetric design, consisting of a host, a multitude of downstream USB ports, and multiple peripheral devices connected in a tiered-star topology. Additional USB hubs may be included in the tiers, allowing branching into a tree structure with up to five tier levels. A USB host may have multiple host controllers and each host controller may provide one or more USB ports. Up to 127 devices, including hub devices if present, may be connected to a single host controller.
USB devices are linked in series through hubs. There always exists one hub known as the root hub, which is built into the host controller.
A physical USB device may consist of several logical sub-devices that are referred to as device functions. A single device may provide several functions, for example, a webcam (video device function) with a built-in microphone (audio device function). Such a device is called a compound device in which each logical device is assigned a distinctive address by the host and all logical devices are connected to a built-in hub to which the physical USB wire is connected. A host assigns one and only one device address to a function.
Diagram: inside a device are several endpoints, each of which is connected by a logical pipes to a host controller. Data in each pipe flows in one direction, although there are a mixture going to and from the host controller.
USB endpoints actually reside on the connected device: the channels to the host are referred to as pipes
USB device communication is based on pipes (logical channels). A pipe is a connection from the host controller to a logical entity, found on a device, and named an endpoint. Because pipes correspond 1-to-1 to endpoints, the terms are sometimes used interchangeably. A USB device can have up to 32 endpoints: 16 into the host controller and 16 out of the host controller. The USB standard reserves one endpoint of each type, leaving a theoretical maximum of 30 for normal use. USB devices seldom have this many endpoints.
There are two types of pipes: stream and message pipes depending on the type of data transfer.
isochronous transfers: at some guaranteed data rate (often, but not necessarily, as fast as possible) but with possible data loss (e.g., realtime audio or video).
interrupt transfers: devices that need guaranteed quick responses (bounded latency) (e.g., pointing devices and keyboards).
bulk transfers: large sporadic transfers using all remaining available bandwidth, but with no guarantees on bandwidth or latency (e.g., file transfers).
control transfers: typically used for short, simple commands to the device, and a status response, used, for example, by the bus control pipe number 0.

*

Hardware

2.1.0 Embedded System

V4L2 mmap()
Name
v4l2-mmap -- Map device memory into application address space
Synopsis

#include <unistd.h>
#include <sys/mman.h>

void *mmap(void *start, size_t length, int prot, int flags, int fd, off_t offset);

Arguments

start

Map the buffer to this address in the application's address space. When the MAP_FIXED flag is specified, start must be a multiple of the pagesize and mmap will fail when the specified address cannot be used. Use of this option is discouraged; applications should just specify a NULL pointer here.

uATX (6.75 inches by 6.75 inches [171.45 millimeters by 171.45 millimeters]) (ITX compatible)
Integrated Intel® Celeron® 220 processor (1.2 Ghz) with a 533 MHz system bus
One 240-pin DDR2 SDRAM Dual Inline Memory Module (DIMM) sockets
Support for DDR2 677/533/400 MHz DIMMs
Support for up to 1 GB of system memory
SiS* SiS662 Northbridge
SiS* SiS964L Southbridge
ADI* AD1888 audio codec
Integrated SiS Mirage* 1 graphic engine
Winbond* W83627DHG-B based Legacy I/O controller for hardware management, serial, parallel, and PS/2* ports

Booting from a USB STICK

Hardware

The Energy – Weight – Heat – Cost Problem

Hardware

GuarddoG Part List / Specifications

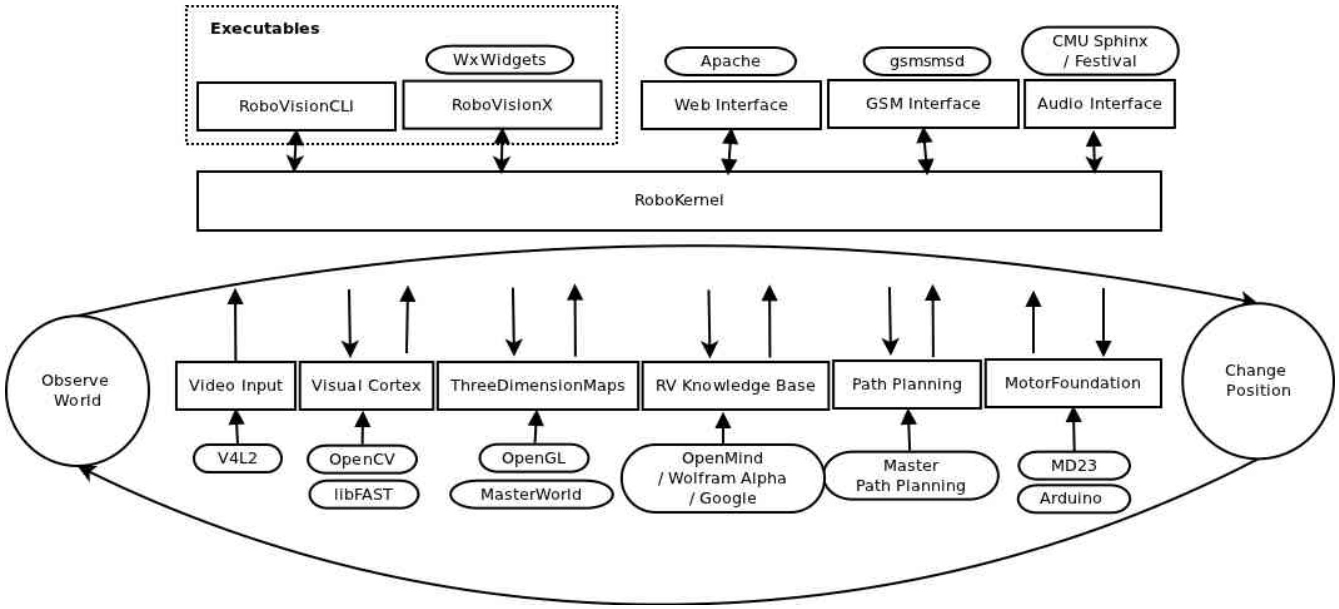
Embedded Electronics
1x Arduino = 25 euro (Uno)
3x Infrared Led = 3 euro
1x RD-01 (or RD-02 Devantech motors) = 130 euro
2x Buttons (power -on) = 2 euro
2x Switches (power supply) = 2 euro
2x LED HeadLights = 10 euro
2x Ultrasonic Devantech SRF-05 with mounting = 40 euro
1x Dual Axis Accelerometer (memsic 2125) = 30 euro
Total : 252 euro

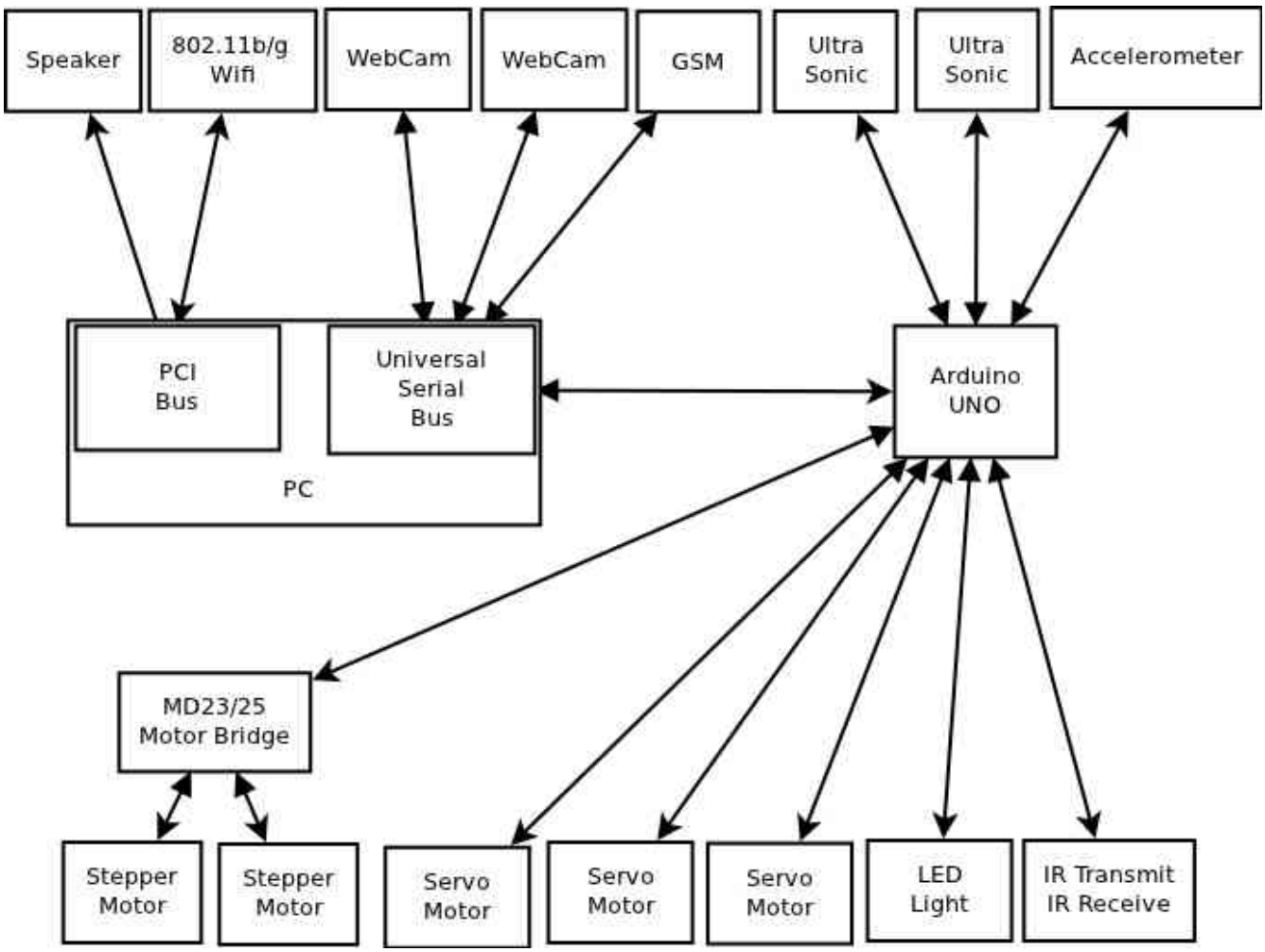
Computer Hardware

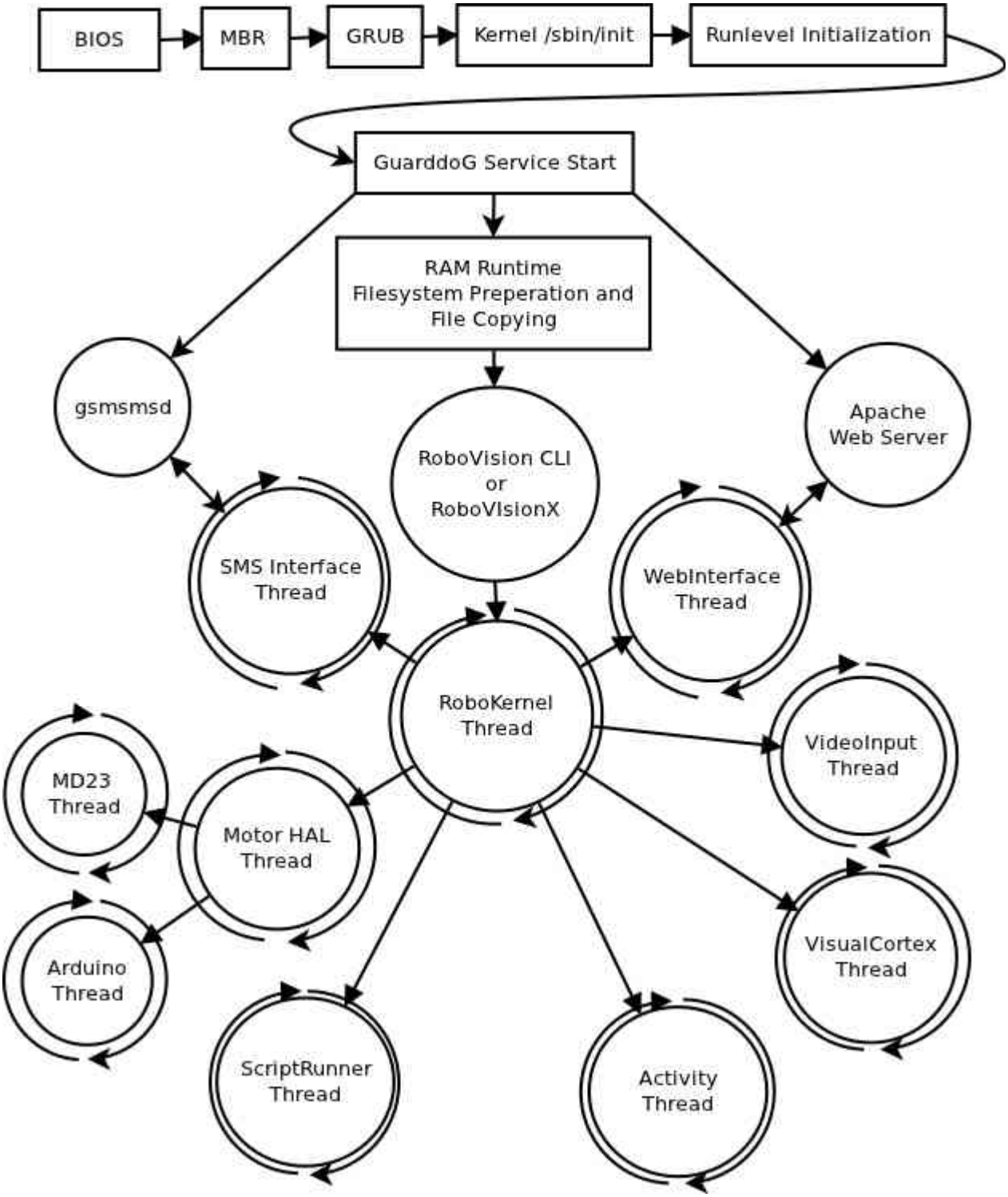
1x Fan = 5 euro
1x Mini-Itx Motherboard = 65-75 euro (Currently on guarddog Intel D201GLY2)
1x PicoPSU 90W = 45 euro
1x AC-DC 12 V Converter = 30 euro
2x Webcams (On guarddog MS VX-6000) = 92 euro , LOGITECH C510 HD , PS3 Eyes
1x WIFI PCI card (WG311T) = 30 euro
1x USB Flash Drive 8GB + = 20 euro
1x 512-2048MB RAM DIMM (on guarddog 512MB DDR2) = 30 euro
Total : 327 euro

Software Stack

Pipeline Outline


Executables
WxWidgets
RoboVisionCLI
RoboVisionX
Apache
Web Interface
gsmsmsd
GSM Interface
CMU Sphinx
/ Festival
Audio Interface
RoboKernel
Observe
World
Video Input
Visual Cortex
ThreeDimensionMaps
RV Knowledge Base
Path Planning
MotorFoundation
Change
Position
V4L2
OpenCV
libFAST
OpenGL
MasterWorld
OpenMind
/ Wolfram Alpha
/ Google
Master
Path Planning
MD23
Arduino


Speaker
802.11b/g
Wifi
WebCam
WebCam
GSM
Ultra
Sonic
Ultra
Sonic
Accelerometer
PCI
Bus
Universal
Serial
Bus
PC
Arduino
UNO
MD23/25
Motor Bridge
Stepper
Motor
Stepper
Motor
Servo
Motor
Servo
Motor
Servo
Motor
LED
Light
IR Transmit
IR Receive



Software Stack

Performance Hypervisor

Software Stack

Implementation Framework

Software Stack

Statistics

The System in Practice

Installation

The System in Practice

Data Sets and Test Results

The System in Practice

Commercial Value

The System in Practice

Weaknesses

Future Work

Network Connectivity

NLP – AI Knowledge Base

Speech Recognition

Commercial Robots

CUDA / VLSI acceleration

Acknowledgements

*

Bibliography / References

[1]
[2]
[3]A Flexible New Technique for Camera Calibration (1998) by Zhengyou Zhang , Zhengyou Zhang
[4]Edward Rosten , Fusing points and lines for high performance tracking , (**YEAR HERE**) Machine learning for high-speed corner detection.
[5] Herbert Bay, Andreas Ess, Tinne Tuytelaars, Luc Van Gool, (**YEAR HERE**) SURF: Speeded Up Robust Features
[6]Jianbo Shi , Carlo Tomasi , (YEAR HERE) Good Features to Track
[7]Yoav Freund, Robert E. Schapire. "A Decision-Theoretic Generalization of on-Line Learning and an Application to Boosting", 1995