Implementation and test of the Time Of Flight scanner # 03D201

by
Kristian Villien
s020320

Introduction

The purpose of this project is to implement the O3D201 - Time Of Flight camera from IFM on the DTU designed SMR (Small Mobile Robot). The camera will be implemented so it can be interface with the DTU designed SMR.

After the camera has been implemented, it will be tested. The test will both be targeted towards general camera parameters, and towards robot specific assignments.

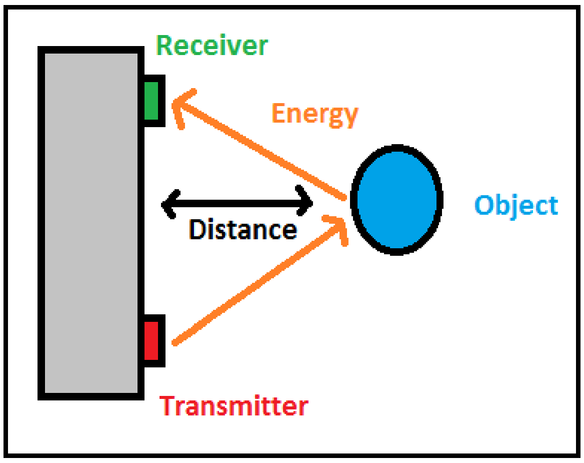
This document will first have a section on the basic theory and operations of the camera, as well as a description of the different camera setting available to the user. The next section will be a description of the software implementation. The last section will be a description of the tests done on the camera, as well as the results.

At the end of this project report a number of suggestion for improvement of the software will be described, and the source code can also be found here.

Index

Camra theory

Time Of Flight theory



The term 'Time Of Flight' (TOF) refers to a measurement scheme where some energy is transmitted from the measurement device, and this energy reflects off one or more objects, thereby returning to the measuring device. The receiver is waiting for the return energy, and when this is measured, the time from transmission to reception is recorded.

The time of flight principle is widely used for object detection, but the most well know areas are sonar and radar, using either sound or electromagnetic waves as the energy form.

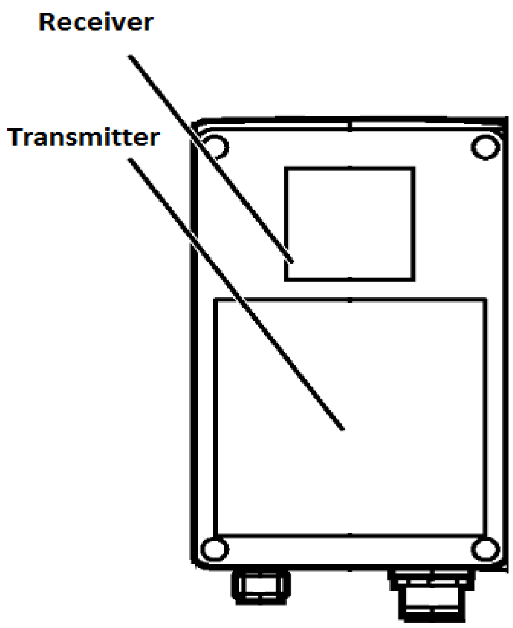
Using light as an energy medium is very much like a radar, as they are both electromagnetic radiation, however with a radar, the distance precision is normally not required to be much more then a meter, where as the visual distance measurements will normally require much high precision.

The distance from the TOF measurement device to the target can be calculated with the following equation:

*Distance [meters] = Speed [m/s] * ½ * Flight time [s]*

A half is multiplied on because the the energy must move both to the object and back again, meaning it moves twice the distance. The speed refers to the speed of the energy.

The speed of sound is different for various travel mediums, and varies with temperature. The speed of sound is normal air at room temperature is approximately 343 [m/s], and in water approximately 1497 [m/s].



The speed of electro magnetic radiation is less medium depended, and is normally simplified to 300,000,000 [m/s] in all mediums. This is a simplification as the speed of electromagnetic radiation does vary with different mediums, but it is close enough for all practical purposes.

From the equation above, it can be calculated that for a 1m distance precision, the time measurement precision needs to be 5.8ms for sound in air and 6.67ns for light.

Basic camera function

The Time Of Flight device is referred to as a camera because is uses a lens and a two dimensional sensor array to capture the image.

The camera has a number of infrared LEDs, and these are used to make a short illumination of the scene in front of the camera. The sensor array not only measures the time is takes for each sensor to receive the infrared light, but also measures how much is returned. Measuring the amount of light, also called intensity, that is reflected of the objects can give an indication of the brightness of the objects.

Because the return time for the energy is short, the energy pulse itself must be really short to have a decent resolution. Having a very short light pulse severely limits the amount of energy transmitted, and therefore also the signal-noise ratio of the reflected pulse. There are basically two ways to boost the signal noise ratio, either repeat the pulse a number of times, or modulate the light intensity with a frequency and measure the phase shift of the returning pulse. Which of these solution is actually used in this camera is not described in the documentation, however it does not actually matter as the result is the same.

Because the transmission is modulated with at a frequency, be it pulsed at that frequency or frequency modulated, it is only possible to measure a distance equal to the wave length of the modulation frequency. Anything further away then the wave length will be as being closer by one wave length, for example, if an object is 8 meters away and the wave length is 6 meters, the object will be seen as being 2 meters away.

The camera has an integration time, where it continuously transmits and receives. It does this integration four times.

Camera features

Transmission modulation channels

As mentioned, the transmission is modulated with a certain frequency, this is done to increase the reflected energy.

There are several transmission frequencies available to the robot, these channels are 23MHz, 20.4MHz and 20.6MHz. These frequencies have the wave lengths 13.04, 14.7 and 14.56 respectively. These ranges have to be divided by two since the light needs to reach the object causing the reflection and back again. This give the unambiguous range of 6.52 meters, 7.35 meters and 7.28 meters respectively.

There is also the possibility of using two different frequencies for the modulation, and match the data from these two modulations, thereby increasing the unambiguous range to 45 meters.

It should be noted that using two modulation frequencies will result in using the double time to acquire the image.

Dual integration

When there is a high integration time, scanning of close objects will max out the signal, which will make it impossible to get a precise distance. When there is a low integration time, the scanning of far away objects will have too low signal, which will result in the objects not being detected. The amount of signal being reflected is proportional to the distance squared. This means doubling the distance will give a quarter of the signal.

On top of the distance problem, it should also be noted that black objects reflect much less energy then white objects, and reflective surfaces, like a mirror, can reflect even less back to the robot.

To counter the problem of having to choose between detecting high signal objects or low signal objects, the camera has the possibility of having two integration time.

When using two integration time, called double sampling, the camera will take an image with both integration times, and chose the best pixel value for the final image. The best value will normally be the long integration time unless this value is maxed out.

The cameras documentation recommend using a factor of 100 between the two integration times, this can however be optimized for static scenes.

Heartbeat

Once a network session has been started on the camera, the camera is blocked from other session. If a software error where to occurs, the camera would need a reboot before being used again.

To counter this, it is possible to set a watch dog on the network session, meaning is the communication stops for a given time, the camera assumes the connection has been lost and ends the session.

To keep the session going when the watch dog has been enabled, it is necessary to keep communicating with the camera. If the user has nothing to send to the camera, a heartbeat function has been implemented which can be called. Calling the heartbeat function does nothing except keeps the network session active.

Temperature protection & Mute time

The camera outputs a lot of energy in infra red light, and since an LED is not very effective, something like 90% or more of the energy put into the LED will be transformed into heat. To prevent the camera from over heating and being damaged, the camera will stop generating images when it gets too hot, and outputs an error until it cools down again.

When the camera is running in free running mode, it is impractical for the camera to shut down all the time to cool off, it is therefore possible to use a mute time, meaning a time when the camera is not doing anything.

A camera period is define by three time, first an integration time where the picture is generated, then a calculation time where the image is being processed, and then the mute time to allow the camera to cool off.

The need for a mute time can be minimized by having heat dissipation in mind when mounting the camera, thereby supplying extra air flow and such.

Filters

The camera has build in filters to increase the quality of the image. There are three filters available, and they can all be used at the same time.

- Multiple images averaging – Taking several images and averaging the data.
- Single image median filtering – For each pixel in the image, the surrounding pixel values are looked though, and the median value is used.
- Single image mean filtering – For each pixel in the image, the value is set to the average of the surrounding values.

Filtering the image increases processing time.

Coordinate calculations

The camera has a unit vector for each pixel in the image, by multiplying this unit vector onto the distance, the camera can output the picture values in (X, Y, Z) coordinates, where Z is the plane facing away from the camera.
The unit vector used for this calculation can also be read as an image output.

Camera parameters

The following parameters are used in this software, and can each be set by a variable.

Name	*Description*	*Default value*
ModulationFrq	Sets the modulation frequency, 0=23MHz, 1=20.4MHz, 2=20.6MHz , 3=double frequency.	0
DoubleSampling	Enables double integration, using both integration times	0
IntegrationTime1	Sets the first integration time in [us].	100
IntegrationTime2	Sets the second integration time in [us].	1000
FrameMuteTime	Sets the idle time between pictures. Can be used to control the frame rate. Also used to prevent overheating. In [ms].	70
FreeRunning	Sets the update mode to free running or manual. Use “ImageGet” in manual mode	0
NumberOfImagesAverage	Sets the amount of images to average over for each output.	1
MeanFilter	Sets the mean filter to on or off.	0
MedianFilter	Sets the median filter to on or off.	0

Software design

Software plug-in Interface

The job of the software is to output the images generated by the camera. The images are dumped into the image pool, and plug-in's waiting for the images get an image updated event as notification when the image has been updated.

The software can run in two modes, free running mode where the images are updated constantly, and manual mode where the image is only updated when a function is called.

The software is controlled by three functions, “open”, “close” and “ImageGet”.

- '“open” is called to connect to the camera and set it up.
- “close” is called to disconnect from the camera.
- “ImageGet” is called to trigger the camera and update the images. When using the camera in free running mode, this function is not used.

A number of accessible variables are created to control the settings of the camera. The software makes sure to updated the camera setting when one of these variables are changed.

A variable is also created for each image type the camera can output, setting one or more of these to '1' will output the image. An image count is updated each time an image is received with a new time stamp, meaing when several images are requested they will have the same image count when belonging to the same frame.

Camera Interface

There are two IP port used to interface the camera, one for setting up the camera, and one for getting the image data. The two ports are interfaced completely separate of each other, with the exception of the camera trigger in manual mode is done through the settings port.

As default the settings port is set to 8080, and the data port is set to 50002, but these can be changed with little difficulty, as can the IP address. The network setting can either be changed manually using the buttons on the camera, or through the settings port. When changed through the settings port, the camera needs a hardware reset to be in effect.

In addition to the two network interface, the camera also has a manual trigger and several IO ports with other purposes, these have not been used in this project and will therefore not be described.

XML interface

The settings port is interfaced with XML-RPC commands. Details on this standard can be found at: http://en.wikipedia.org/wiki/XML-RPC. The XML-RPC commands must be precluded by a HTTP header.

The camera documentation refers to a standard library available for free download, however since this is for windows, and the XML-RPC is very simple, this has not been used.

Some mistakes and missing informations in the camera documentation should be noted.

1. The command “`MDAXMLHeartBeat`” (as it is called in the camera documentation) should be written with a lower case 'B', “`MDAXMLHeartbeat`”. Using a upper case 'B' will result in an unknown command error.
2. It is essential to send the “encoding” parameter in the XML tag, without this the camera will not respond. This is not documented anywhere. Example of an acceptable XML start tag:

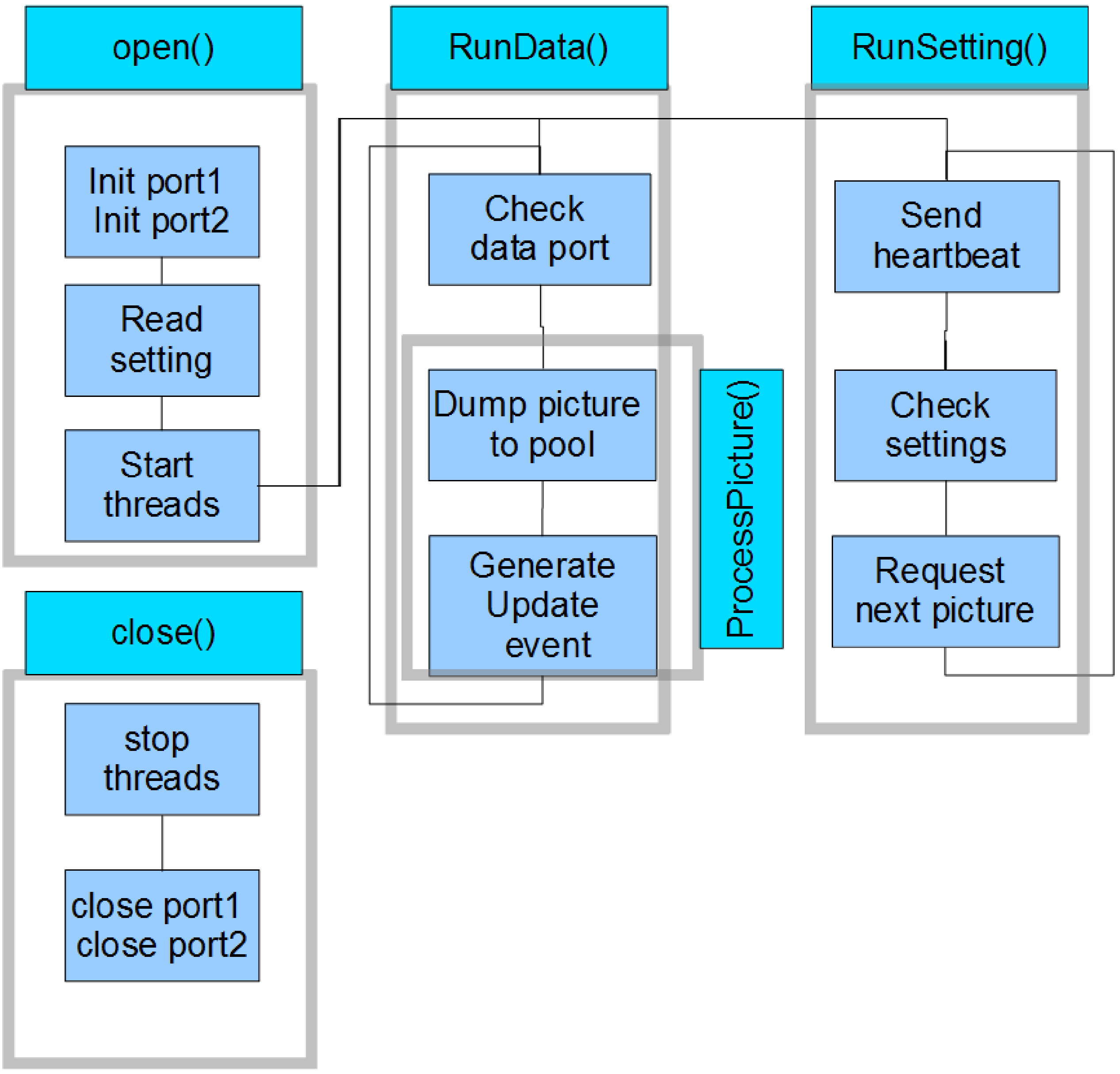
   ```
   <?xml version="1.0" encoding="UTF-8" ?>
   ```

Software build up

The basic software structure can be seen below.

Calling the "open" command initializes the two IP ports, reads the cameras current setting and starts the data thread and the command thread.

The camera saves its latests settings even after a power down, however it has been requested that the camera is always initialized with the default setting. While this initialization is not done doing the open command, it is done very quickly in the settings thread. The default values are set in the "void UfuncTOF::createResources()" function.



Settings thread

The settings thread has several jobs.

- The first job is to keep the heartbeat going, meaning at least once every ten seconds a Heartbeat XML command must be send otherwise the camera assumes that the connection with the client has been lost. A five second interval has been chose.
- The second job is to make sure that the current camera setting is the same setting requested by the user. To do this, two sets of variables are kept, one holding the requested user setting, and one holding the current camera setting. If the two variable sets differ, the user requested setting is send to the camera, and read back to make sure it is set correctly.

- The third job of the settings thread is to request new image data when needed. The request of new image data is placed in the setting thread since, when using the camera in manual mode, the trigger must be send over the settings port.

Data thread

The data threads only job is to keep checking for image data in the data port. The reason this was given a separate thread is that XML-RPC communication through the settings port waits for a reply, and this may take time, and doing that time the data port buffer might fill up.

When enough data for a full image has been accumulated, the image is processed and dumped to the image pool.

In addition to checking for data on the data port, the data thread also handles a time out of the image request, since it has been found that switching between free running mode and manual mode can cause the requested image data to get lost, but not always.

Initialization

When the module is added to the ucam server, the constructor is called, located in the header file, the constructor calls the "init()" function. Adding the module to the ucamserver also calls the "createResources()" function which initializes all the U-variables used by this plug in.

The "init()" function does very little since most of the initialization happens when the user does does an "tof open" command.

The "tof open" command initializes internal variables and connects to the two IP ports, setting and data. It also reads the current camera setting and starts the two threads.

XML-RPC communication

description

The XML-RPC communication is preceded by a HTTP header. This header is always the same, and is included in both the call and the reply.

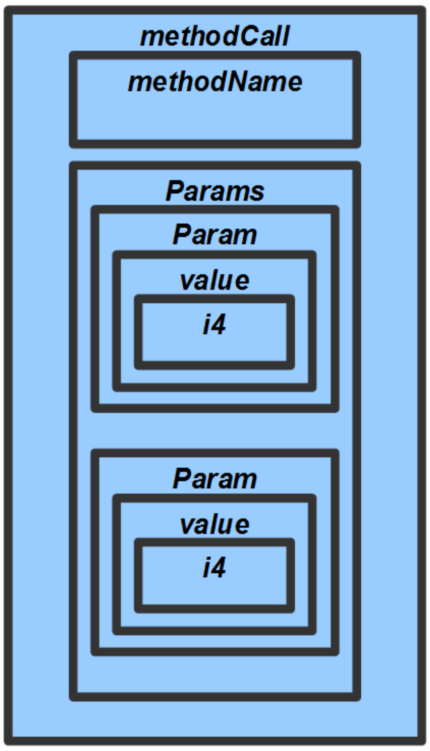
The HTTP header has the following format:

HTTP/1.1 200 OK
Server: XMLRPC++ 0.7
Content-Type: text/xml
Content-length: (insert size of XML data)

The XML-RPC data is packed into XML tags in a very simple structure, an XML tag in this case refers to both a begin tag and an end tag in the following way:

<xml_tag> (xml_tag data) </xml_tag>

the xml tag have several levels, and a method call using the XMLRPC protocol has the build up as seen to the right, with the parameter values inside the <i4> tags. The number of parameters



depend on the method call. The <i4> tag is the parameter type, and is only used when the parameter is an integer. When the parameter is a string, such as an IP address, the <i4> is not included.

The XML response has a similar build up, with <methodResponse> instead of <methodCall>.

It should be noted that the XML tags in the XML response does not always include end tags.

Generating call

An XML call is generated by a simple switch statement, with each parameter either linked to a variable or set to a constant value.

Parsing the response

The response is handles by simply removing the XML tags, and saving all values inside the <value> tags. After having the values isolated, a simple switch assign these values to the correct variable.

XML-RPC commands

The following commands have been implemented and tested:

- MDAXMLConnectCP : Must be the first function called, starts the camera session.
- MDAXMLDisconnectCP : Ends the camera session.
- MDAXMLHeartbeat : Used to indicated the camera session is still ongoing.
- MDAXMLSetWorkingMode : Unknown function - "turn on the image server".
- MDAXMLSetFrontendData : Set up the basic camara parameters.
- MDAXMLGetFrontendData : Gets the current samera setting. This data is also returned by the SetFrontedData function.
- MDAXMLGetTrigger : Gets the current trigger mode.
- MDAXMLSetTrigger : Sets the new testing mode.
- MDAXMLTriggerImage : Triggers an image is the trigger is set to software trigger.
- MDAXMLGetAverageDetermination : Gets the current number of images to combine for each output.
- MDAXMLSetAverageDetermination : Sets the number of images to combine for each output.
- MDAXMLSetMedianFilterStatus : Turns the median filter on or off.
- MDAXMLGetMedianFilterStatus : Gets the current status of the median filter.
- MDAXMLSetMeanFilterStatus : Turns the mean filter on and off.
- MDAXMLGetMeanFilterStatus : Gets the current status of the mean filter.

The following commands have not been implemented:

- MDAXMLGetIP
- MDAXMLSetIP
- MDAXMLGetSubNetmask
- MDAXMLSetSubNetmask
- MDAXMLGetGatewayAddress
- MDAXMLSetGatewayAddress
- MDAXMLGetDHCPMode
- MDAXMLSetDHCPMode
- MDAXMLGetXmlPortCP
- MDAXMLSetXmlPortCP
- MDAXMLGetTCPPortCP
- MDAXMLSetTCPPortCP
- xmlOPS_GetIORegister

- xmlOPS_SetIORegister
- MDAXMLGetDebounceTrigger
- MDAXMLSetDebounceTrigger
- MDAXMLGetProgram
- MDAXMLSetProgram

Data requests

Image data is requested by writing a simple ASCII string to the data port. Each picture requested is represented by a character:

- d/D – distance image
- i/I - Intensity image
- e/E - x coordinate of line of sight vector
- f /F - y coordinate of line of sight vector
- g/G - z coordinate of line of sight vector
- x/X - cartesian x coordinates
- y/Y - cartesian y coordinates
- z/Z - cartesian z coordinates

The uppercase characters means that the image will not be a repeat of the last image sent, and the camera will wait for the next image. Since the image data is returned in the order requested, making the first character in the string an upper case letter will result in all images being returned after the next camera image is ready.

The image data returned has two parts, an image header and image data. Everything return is in floating point variables.

image processing

All image data is in floating points variables, both the image header and the image pixel data. Since the image pool works with integers, the data must be converted to such. The image pool is set to 16 bit signed data, and the float is converted to millimeter resolution. This allows for points in an area of ±32meters. The precision of the camera is set to 10mm in the data sheet.

It is possible to get 8 different images, these can be seen in the previous section. Each image has a different number in the image pool. The image pool numbers start at 70 and end at 77.

The output of the camera is actually rotated 90°, so it is rotated back when writing to the image pool. No image processing is done beyond what is done internally in the camera.

It should be noted that if the signal is too strong or too low, the data for that pixel is set to a negative value, which is why it is possible to have negative distances or intensities.

Connecting the camera

The camera has two wires connected to it, a standard Ethernet cable and an power/IO cable. The camera must be connected to a 24V power supply to be able to work. The power/IO cable connector is missing, but below the color code can be seen below.

Signal	***Pin number***	***color***
U+ (24V)	1	brown
trigger input	2	white
0 V (Gnd)	3	blue
OUT 1 / analogue output	4	black
Ready	5	grey
OUT 2	6	pink
IN 1 / Switching input 1	7	lilac
IN 2 / Switching input 2	8	orange

Camera test

All tests are done any filters enabled. While the filters can reduce the noise, the limited amount of pixels makes the data loss too great.

Two different types of test are done, data tests and environment test. Data test is do get an idea of the quality of the camera while environment test are to test the camera in typical robot uses.

Data tests

The following test are used to determine the quality of the image-data output from the camera.

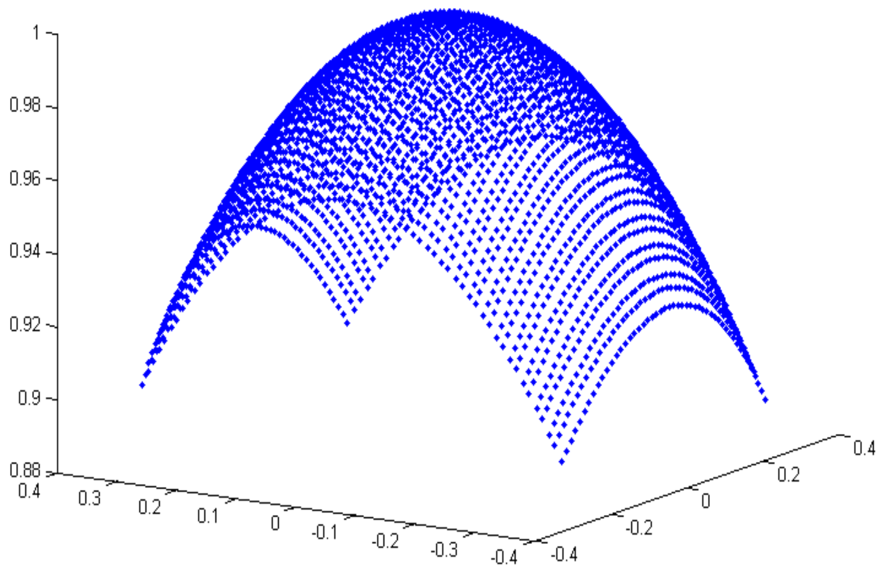
Image angle

The purpose of this test to to determine the cameras viewing angle.

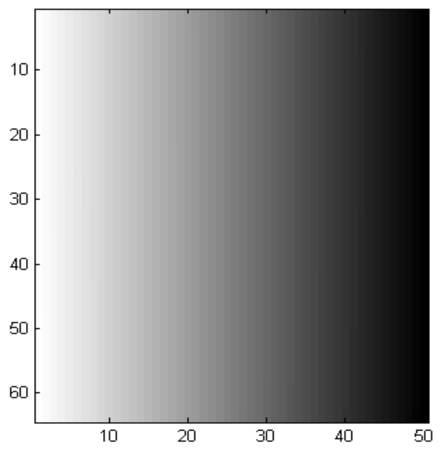
Doing this test double sampling was used and the integration times where set to 100/1000.

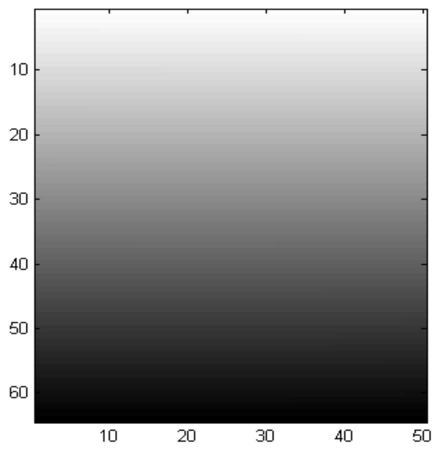
Unit vector

The camera can output a unit vector for each pixel. A 3D plot of the unit vectors can be seen below:



This unit vector can be calculated into two angles, one along the X axis (left to right), and one along the Y axis (up to down). The unit vectors outputet from the camera can be seen bolow. To the left is the X vector and to the right is the Y vector.





The corner and centre angles are as seen below.

position	X angle	Y angle
Top left (1,1)	17	21
Top right (1,50)	-16,3	21
Bottom left (64,1)	17,1	-21,8
Top right (64,50)	-16,4	-21,8
Center (26,32)	0,08	-0,07

Resulting angle between pixels:

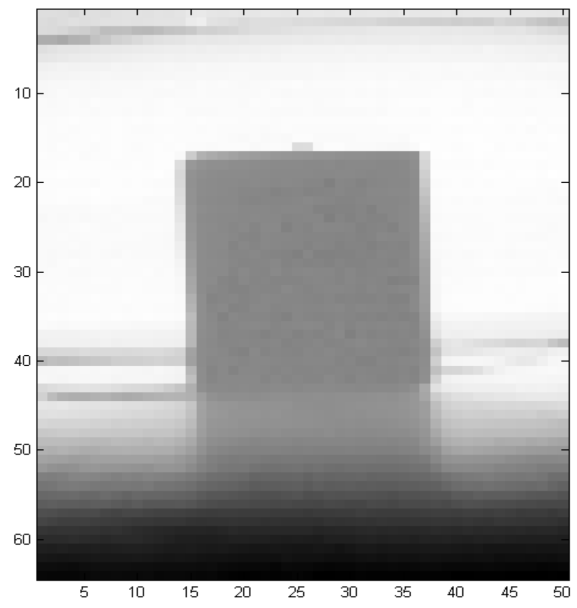
X angle = 0,67

Y angle = 0,67

Testing the angle

placing a box on the floor 1m away from the camera, with a box width of 27cm, should result in a pixel width of 19,5.

Resulting Z axis image:



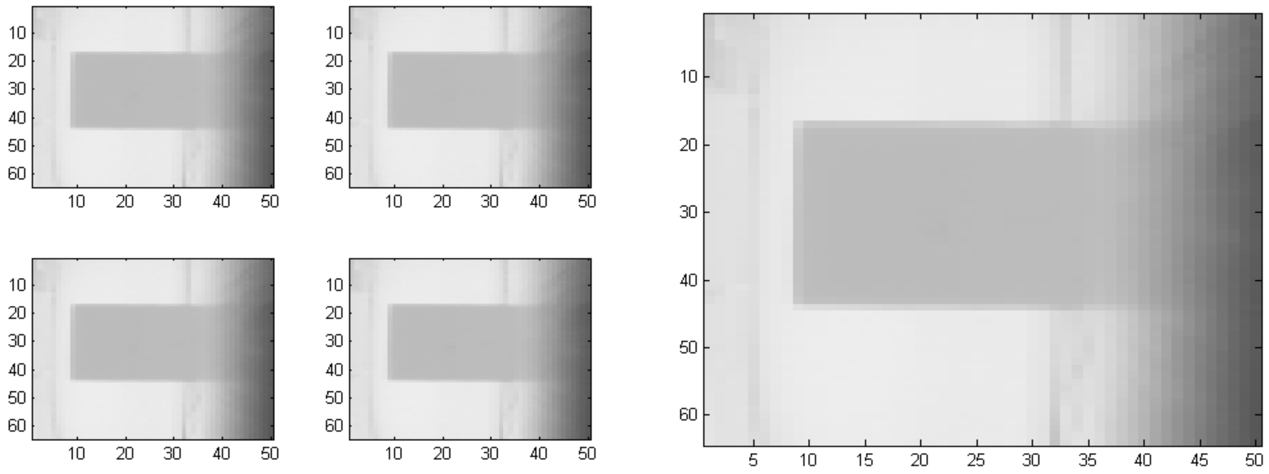
The width of the box is from pixel 16 to pixel 37, resulting in a width of 21 pixels. Pixel 15 and 38 has a value of somewhere in between the box distance and the wall distance.

Accuracy test

The purpose of this test is to test the accuracy of the camera. The test is done by placing a box a distance away from the camera and generating a number of images. The camera is place on its side as mounting it like this is easier. Double sampling is used, integration times: 100/1000. four pictures are taken, and the average of these is used.

Test 1 – 1.5 meter

The four pictures and the combined picture:

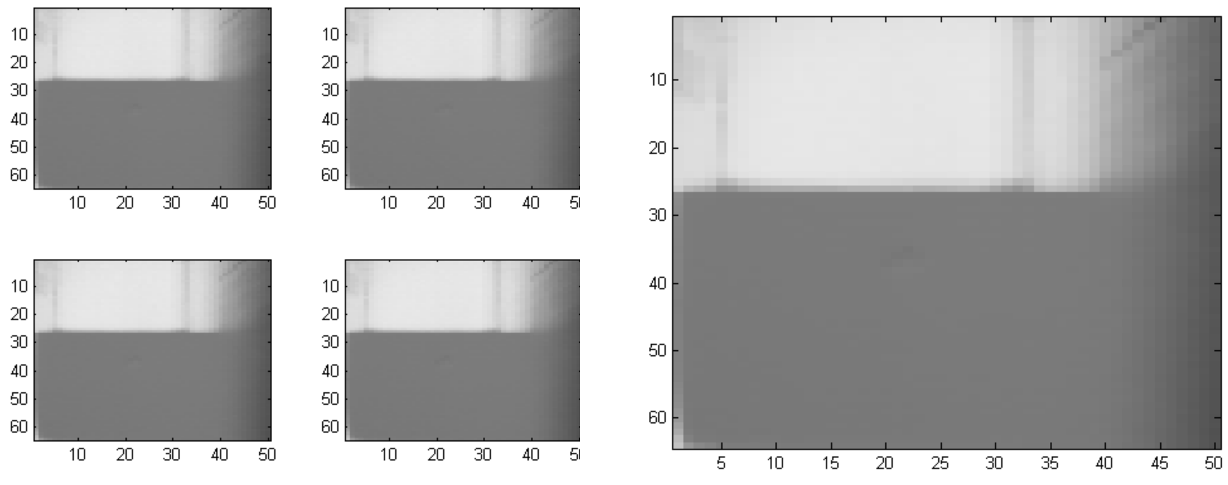


The average distance over the area of the box was 1473mm, and the pixel variance of each individual picture was 2.4036mm, 2.5990mm, 1.8556mm and 2.0214mm.

Test 2 – 1.0 meter

Same test with a box 1 meter away, the integration times was set to: 50/500 to prevent overflow from a direct reflection.

The four pictures and the combined picture:

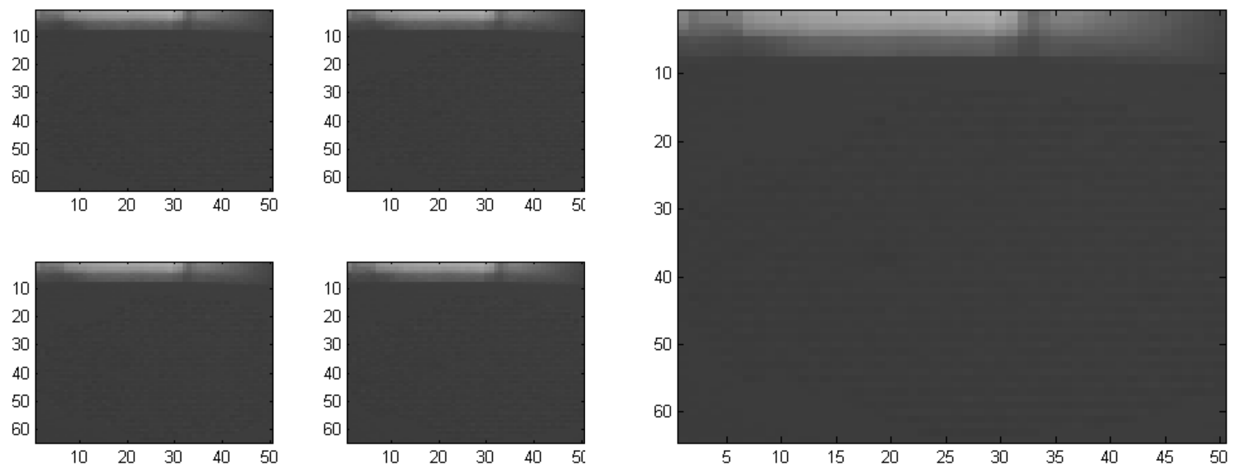


The average distance over the area of the box was 961mm, and the pixel variance of each individual picture was 2.9567mm, 3.4594mm, 2.8928mm and 4.0468mm.

Test 3 – 0.5 meter

Same test with a box 0.5 meter away, the integration times was set to: 50/500 to prevent overflow from a direct reflection.

The four pictures and the combined picture:



The average distance over the area of the box was 477mm, and the pixel variance of each individual picture was 45.6619mm, 45.6619mm, 49.7247mm and 48.3315mm.

Results

The results show that the camera distance was around 3cm short of the manually measured distance. This difference could be cause by either the manual measurement or the camera measurement.

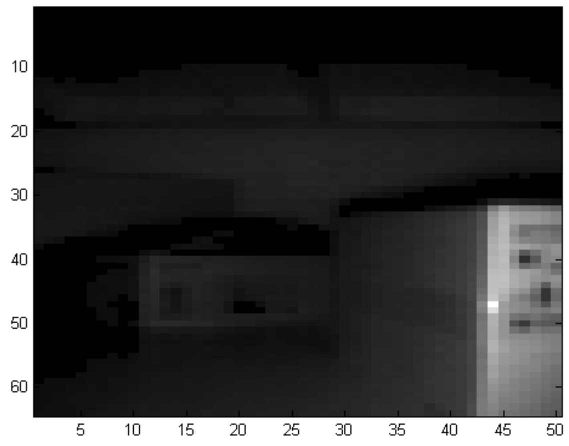
At close quarters, the noise greatly increased. The big variance in the last test was not a big difference between pictures, but rather a large amount of random pixel noise. The noise could be up to several cm.

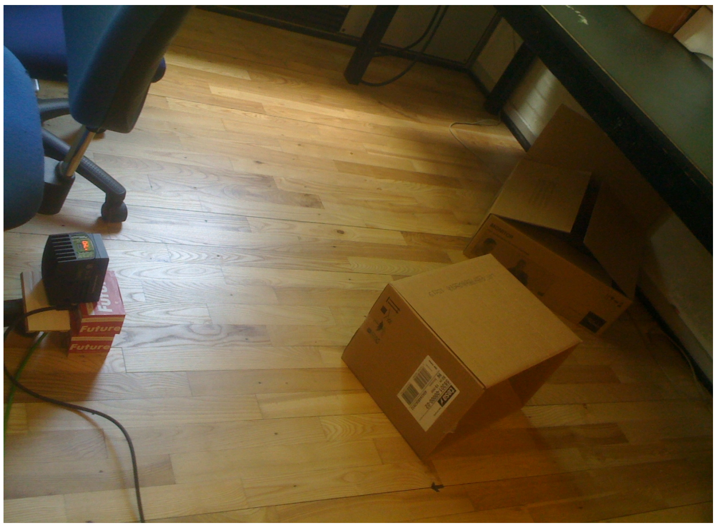
The great increase in noise is assumed to be caused by switching from long integration time to short integration time.

Noise test

The purpose of this test is to get an idea of how noisy the measurements are.

The noise test is conducted with two boxes in front of the camera at different distances. The boxes have their corners facing the camera to avoid direct reflections. This is show in below picture, and also the intensity measurement is shown.





The distance measurements are done with single integration to see if the integration time affect the noise level. A total of 20 images are taken with every integration time, and the distance noise is evaluated. (Please note that the unsigned error distance is used and not the variance).

Pixels that are either too low intensity or too high intensity to be measured are ignore. Each test shows 3 images

The left image is the distance image, black is zero distance, meaning unmeasurable, and white is 2.1m.

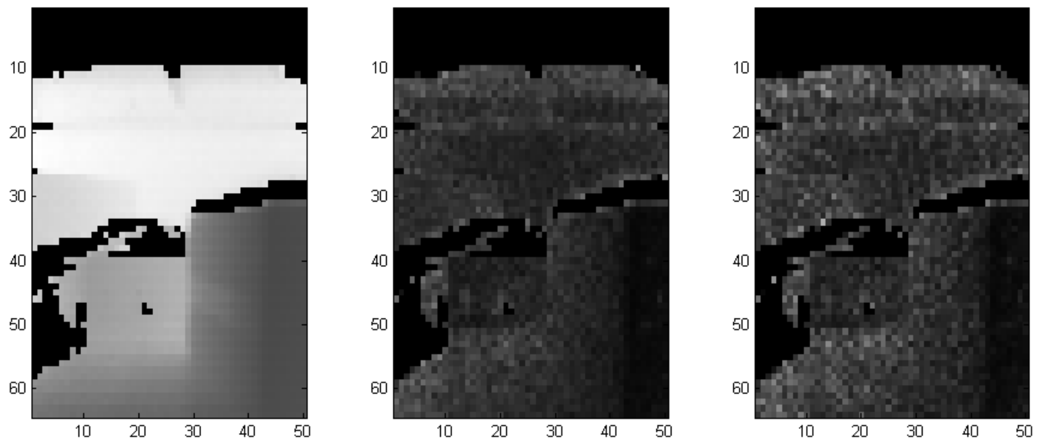
The centre image is the average pixel noise, black is zero noise and white is 100mm.

The right image is the maximum noise for that pixel for the 20 picture series, black is zero and white is 200mm

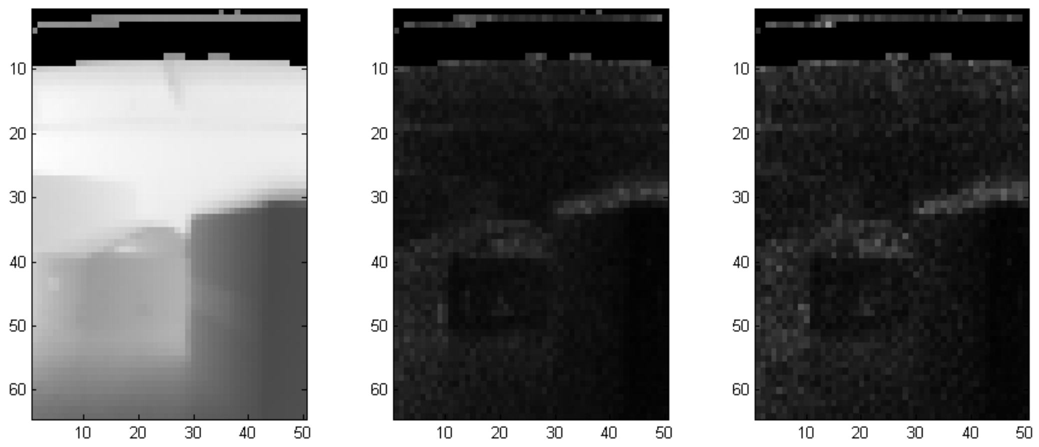
The following integration times are tested:

- 100
- 250
- 500
- 1000

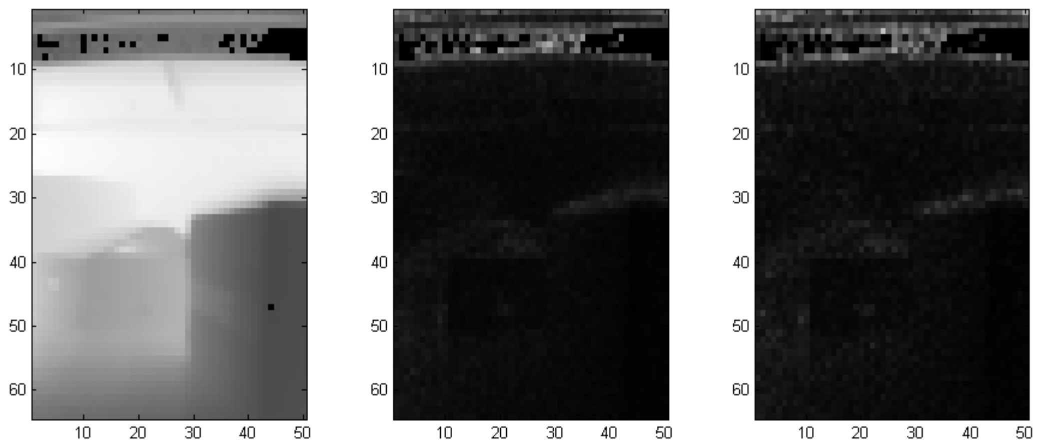
Integration time 100



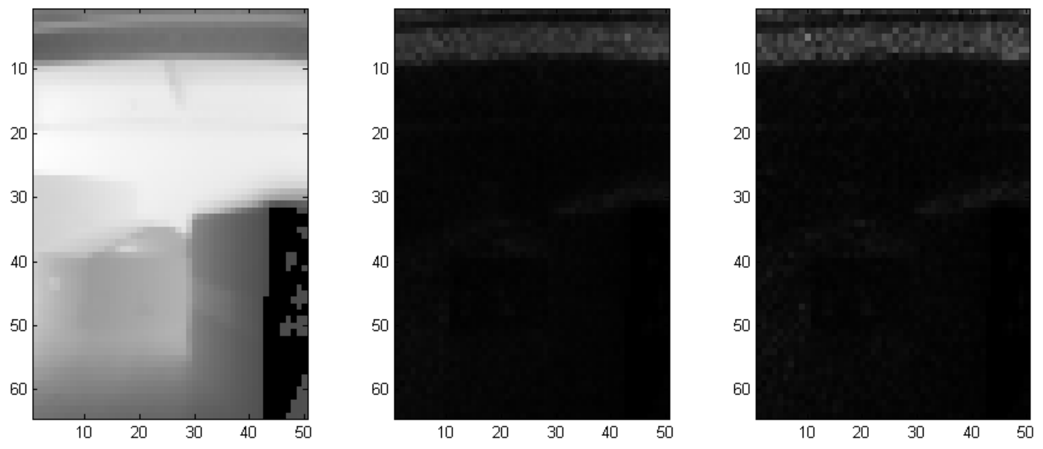
Integration time 250



Integration time 500



Integration time 1000



Results

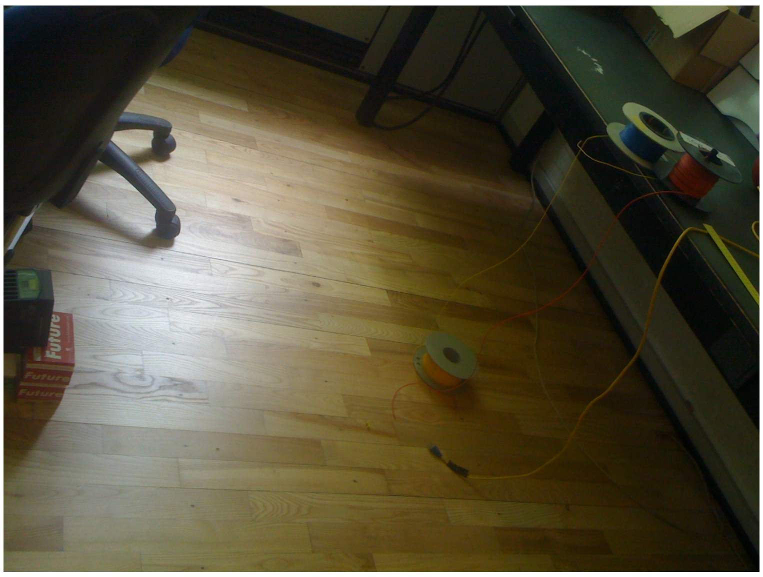
The average pixel noise of the valid pixels can be seen below, although it should be noted that the number of invalid pixels varies with the integration time:

Time: 100	Time: 250	Time: 500	Time:1000
17 mm	8.6 mm	6.1 mm	4.4 mm

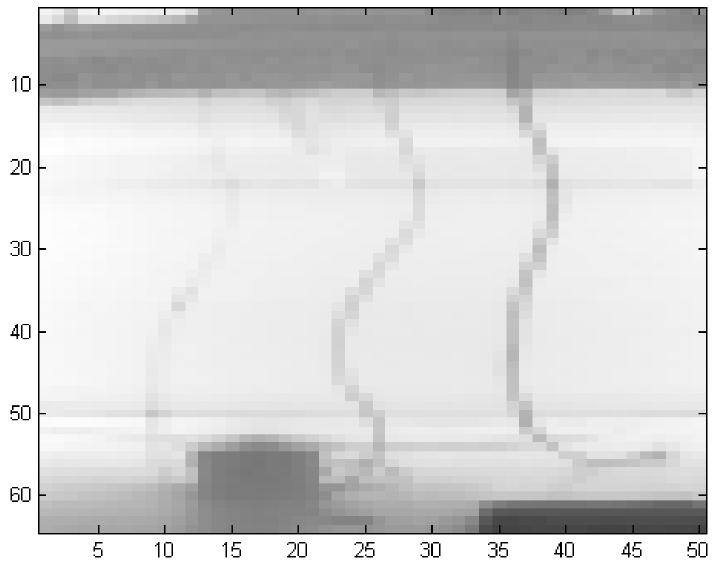
With the low integration time several areas could not be seen due to too low intensity, while with the high integration time some areas could not be seen due to too high intensity. This test shows the advantage of using double integration.

Minimum object test

The purpose of this test is to measure how small objects are measured by the camera. The test is conducted with a three wires hanging off a table approximately 1 meter away from the camera, as seen on the picture below. Starting from the left, the wires have a diameter of approximately 1mm, 2mm and 5mm.



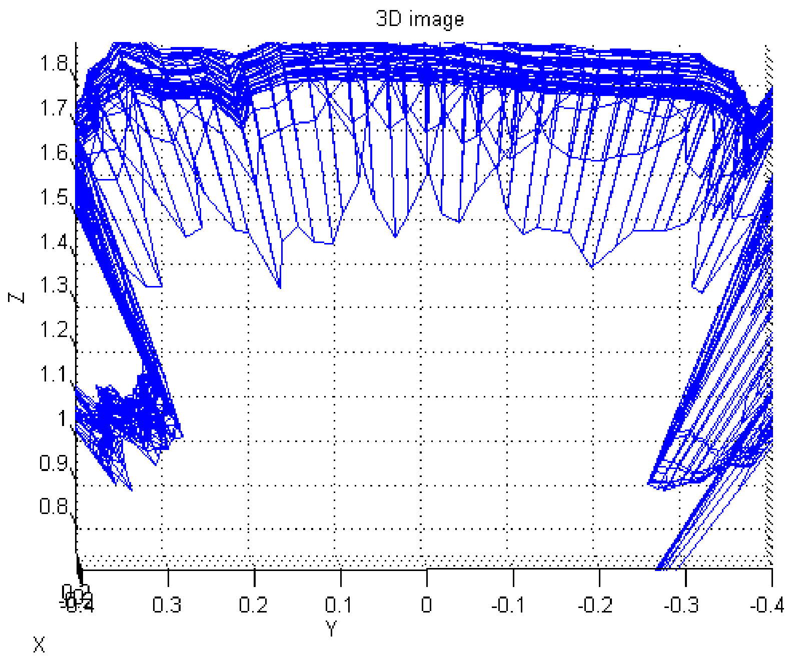
The resulting distance measurement can be seen below.



The three wires can be seen, but it can also be seen that they are at different distances, which is not correct as they are all hanging off the same table edge.

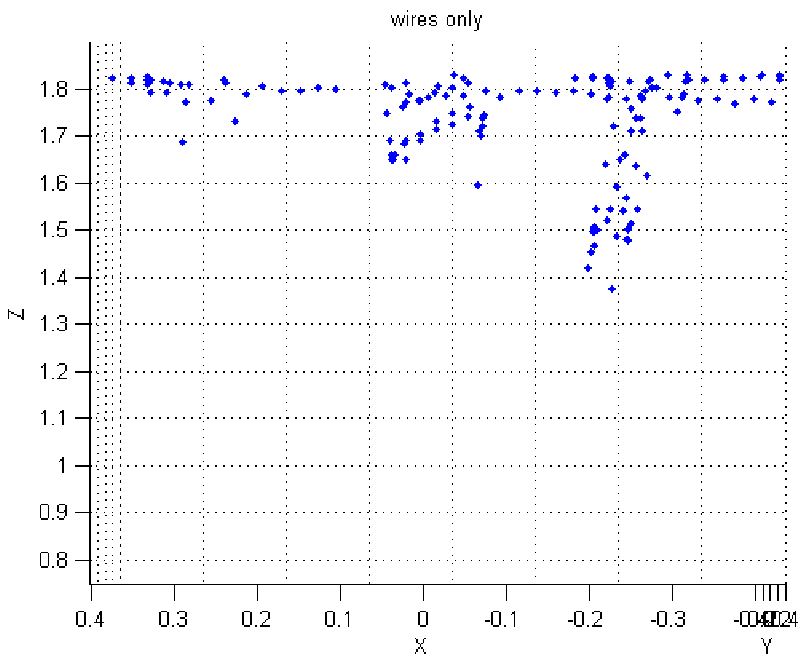
The next figure is a 3D image of the scan.

The floor can be seen to the right of the image, the table edge can be seen to the left, and the back wall can be seen in the top of the image, with the wires sticking out.



It can be seen that the wires are not at the same distance as the table edge.

Isolating the wires and looking at them from the top can be seen in the next figure. The left dots are the thin wire, the middle dots are the middle wire and the right dots are the thick wire



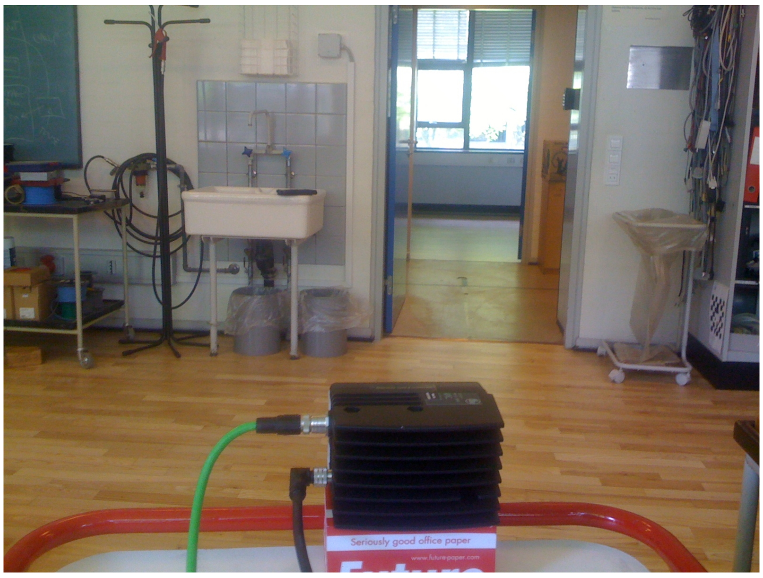
All wires are located at about 0.8meter away from the back wall. The wire with one millimetre diameter is seen as between 0.0 and 0.1 meters. The middle wire with a diameter of 2 millimetre is centred at about 0.1 meter away from the wall. The thick wire, with a diameter of 5mm is seen as about 0.3 meter away from the wall.

With an opening angle of 0.67° for each pixel, at one meter distance it should cover a width of 1.2cm. The width of the thick wire is 0.5cm which results in the wire filling roughly about 40% of the scan area. The wire is seen as being away from the wall with about 40% of the distance from the wire to the wall, so an indication of a linear relationship between the resulting distance measurements and the actual distance of the different objects captured within one pixel can be found, although more testing is required to confirm this.

Modulation frequency's test

The purpose of this is to test the distance wrap around and the double frequency modulation function.

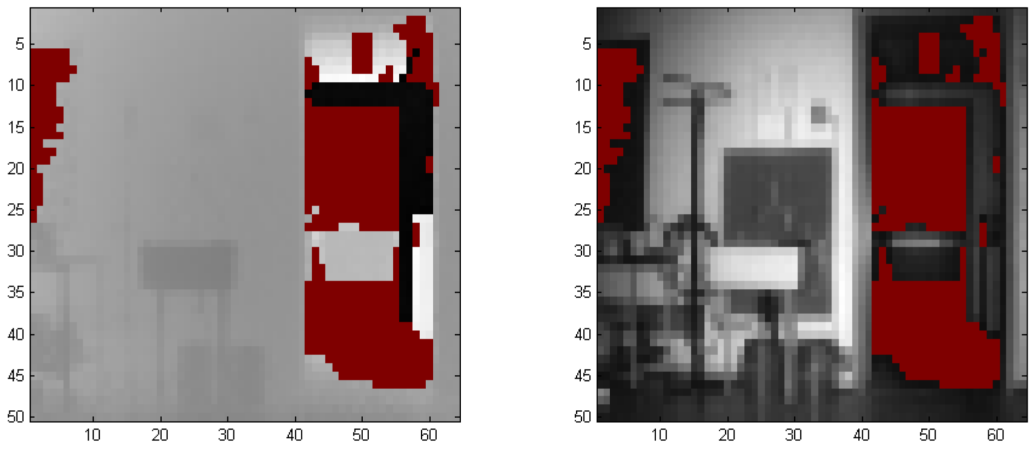
The camera is set-up to take an image out a doorway as show in the picture below.



The integration time is set to 200/2000, this is the highest integration time that can be supported with the power supply used. One image is taken with each modulation frequency, and one image is taken with dual modulation frequencies.

The left image is the distance and the right is the intensity. Any pixel with too low (or too high) intensity to be measured is draw as dark red.

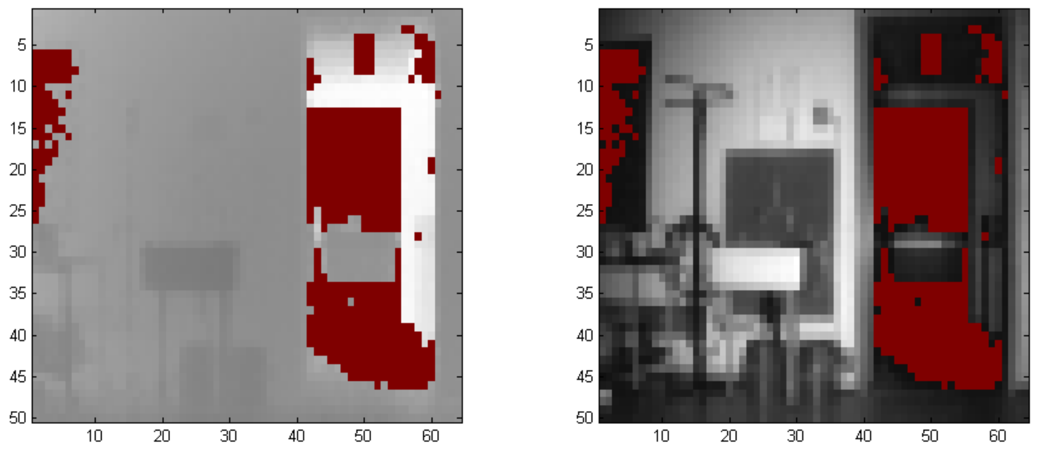
Frequency 0 : 23MHz



it can be seen that the far wall in the corridor can be seen as being really close, around 0.2 meter away from the camera.

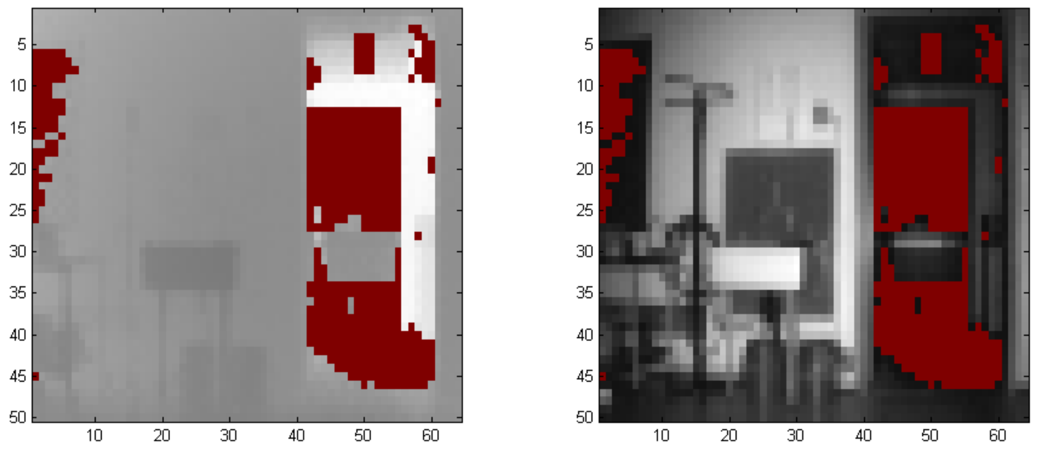
The far wall in the far room is seen as being about 4 meters away, about the same distance as the far wall in the room with the camera.

Frequency 1 : 20.4MHz



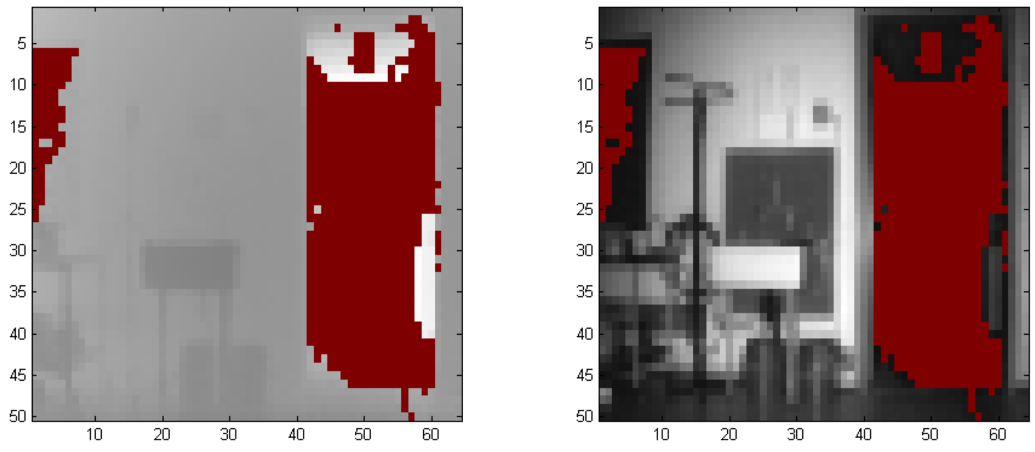
Changing to a lower frequency increases the measurable distance before it wraps around. In this measurement the far wall in the corridor can correctly be seen as being far away, however the far wall in the far room is still seen as being closer then it is.

Frequency 2 : 20.6MHz



The result with 20.6MHZ is very close to the same as the result with 20.4MHz.

Frequency 3 : Dual frequencies



The dual frequency result shows only what can be seen within the wrap around limit of the 23MHz result, which indicates that this is one of the frequencies used.

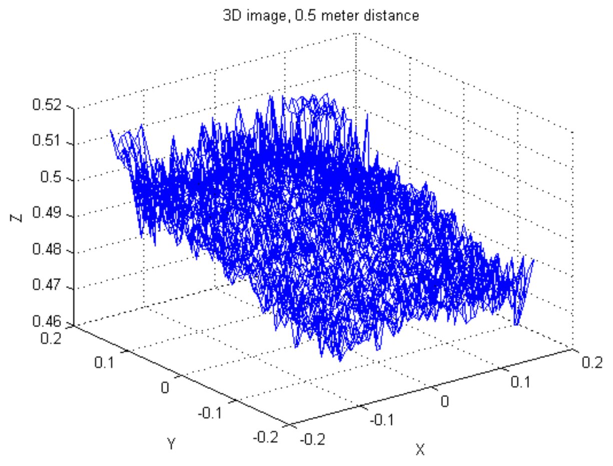
The result shows nothing which is beyond the distance possible with 23MHz modulation, indicating that this is the measurable limit in the test. The maximum distance measured in this test was 6.44 meters, just short of the theoretical 23MHz limit of 6.52 meters.

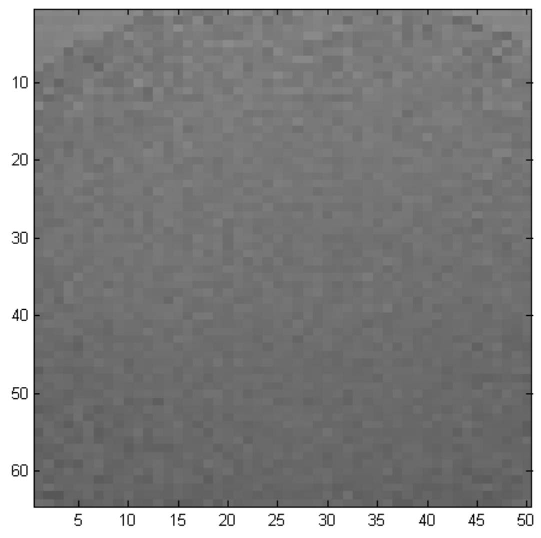
Curved edge test

The purpose of this test is to see how good the unit vector is on a straight surface, since doing the tests some curving has been seen at the camera edges. The test is conducted with the camera being placed in front of a brick wall, the test is done at a distance of approximately 0.5 meter, 1 meter, 1.5 meter and 2 meters.

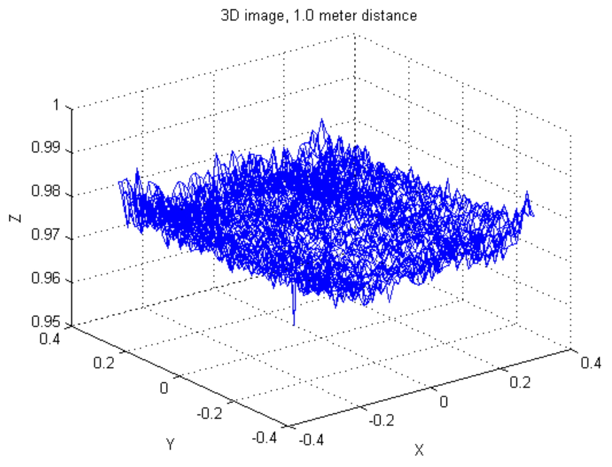
Two images are shown for each test, the left one is a 3D image, the right one is a 2D image where the Z axis is shown as a grey colour, white being +10cm and black being -10 cm.

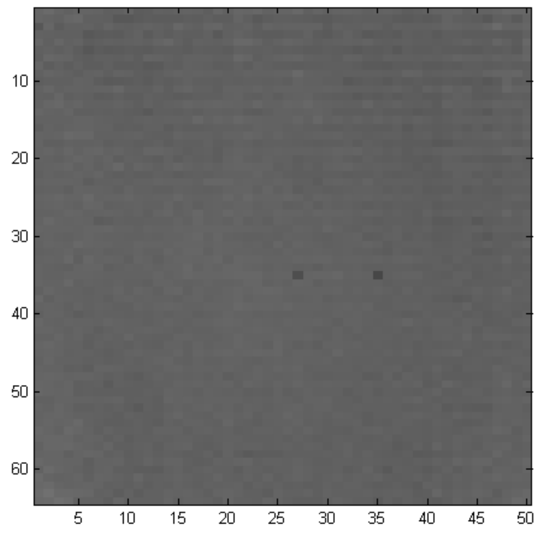
Distance 0.5 meter



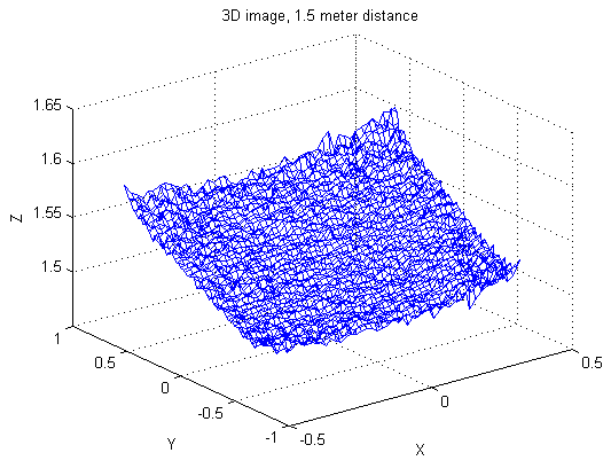


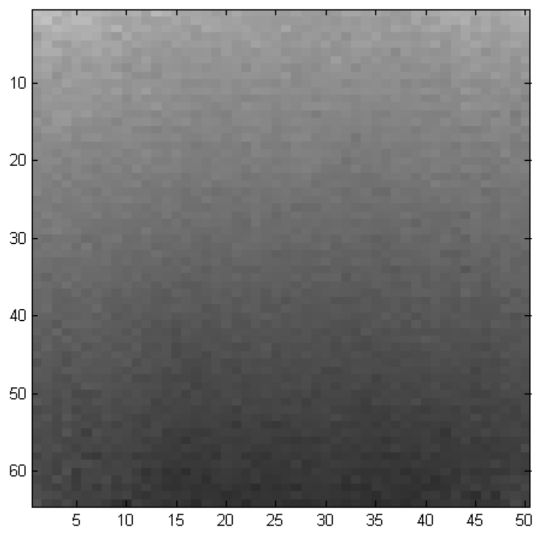
Distance 1.0 meter



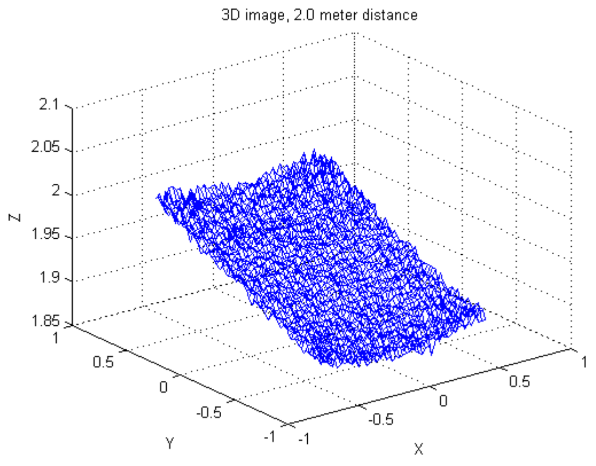


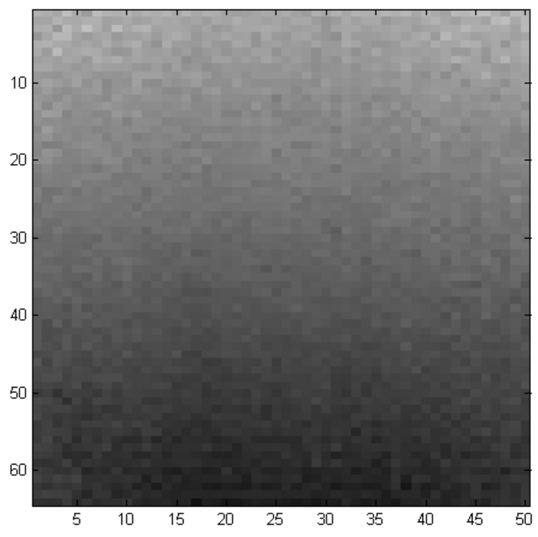
Distance 1.5 meter





Distance 2.0 meter





It can be seen that especially the top two corners have a slight curving away form the camera, how ever this is minor and should not greatly affect the measurements.

Environment tests

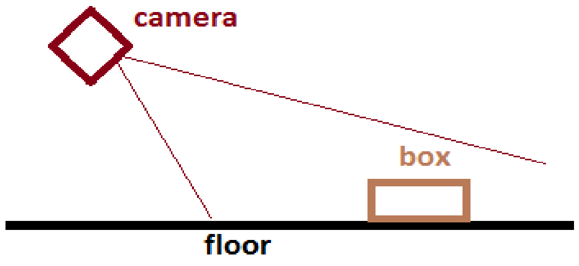
A number of tests are done to see how the camera does at tasks ofeten required at robots.

Floor plane extraction

The purpose of this test is to genera a floor plane so it is possible to identify any objects blocking the robots path.

Test description

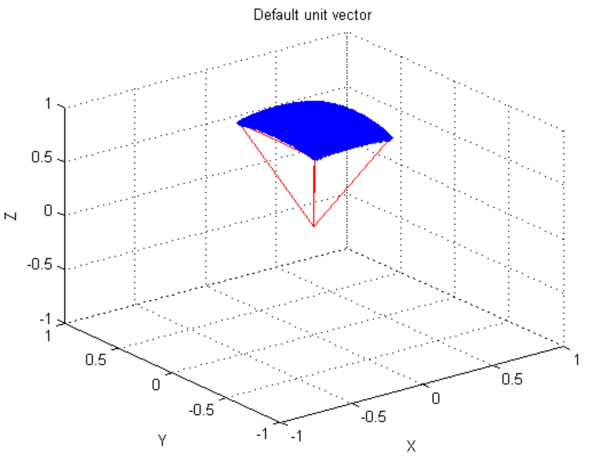
The floor-plane extraction is done using the camera generated unit vectors.



The test is done placing a wooden block on the floor with the dimensions 15cm X 15cm X 7 cm. The camera is then hand held at an angle of approximately 45° at waist height.

The floor-plane extraction is done by turning the angle of the unit vector so it is pointing in the direction the camera is pointing, by doing this, the measurements of the floor should be position parallel to the Y axis. By afterwards subtracting the hight of the camera position, the Y axis should position on the Y axis.

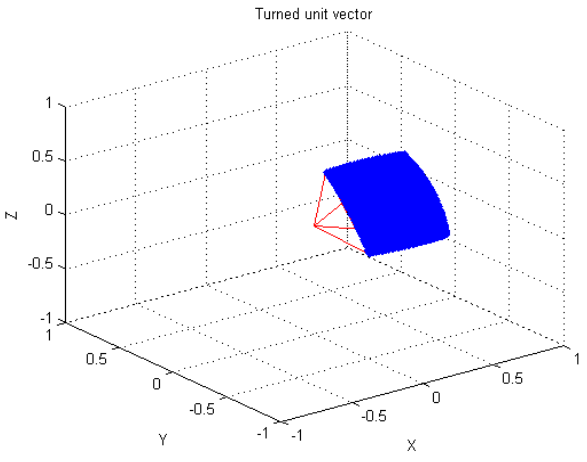
Unit vector direction



The centre of the unit vector is directly out the Z axis. This would be correct if the camera was pointing straight forward, but since it is being held at an angle, the unit vectors must be turned to match that angle.

There are three direction to turn the unit vector, around the X-axis, around the Y-axis and around the Z-axis.

Since the camera is held towards the floor, the direction must move from out the Z-axis, which is forward, to out the Y-axis, which is down, this means it must be turned around the X axis.

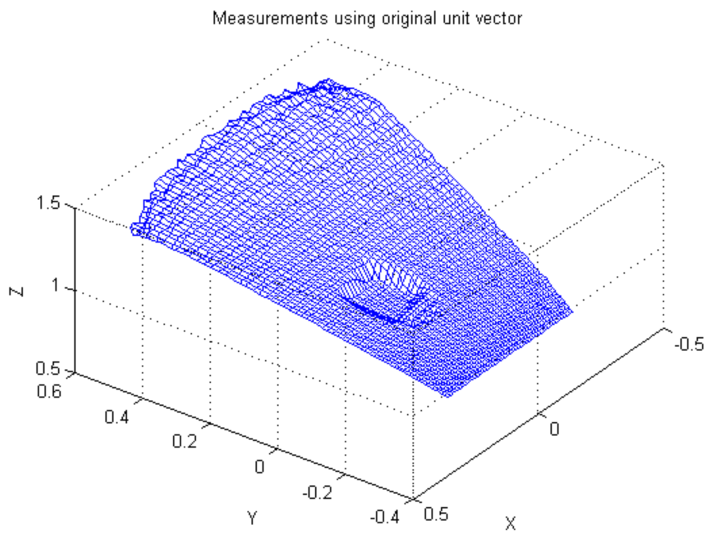


Trail and error has shown the the camera was held at an angle of 52° around the X-axis, and slightly crooked, at an angle of 3° around the Z-axis. Turning the unit angle by this amount can be seen here.

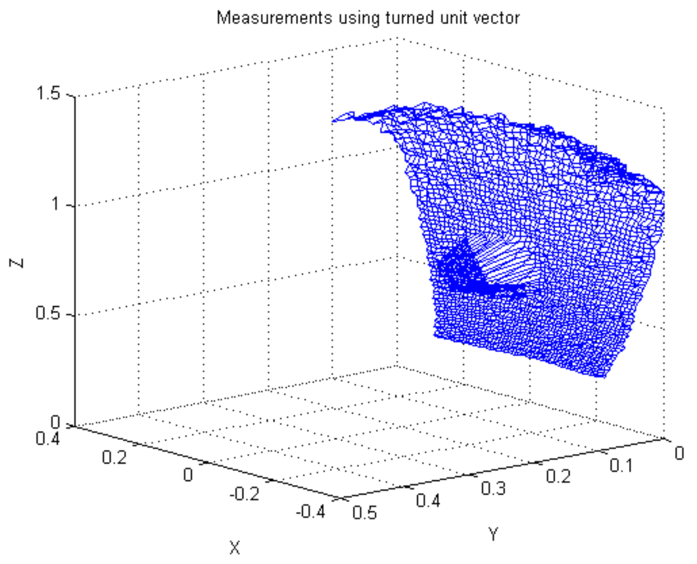
It has also been found that the camera was held at a high of around 0.85 meter.

Measurements

The measurements as a 3D image can be seen here:



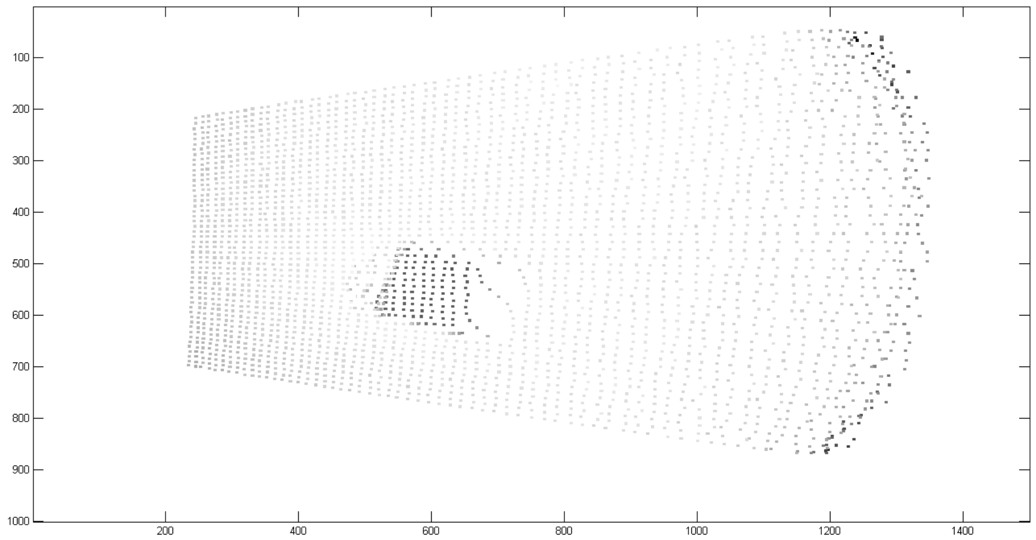
A floor can clearly be seen, but it is also hard to determine where this is position. The result of the turned unit vector can be seen below:



The result is that the floor-plane is located on the Y plane.

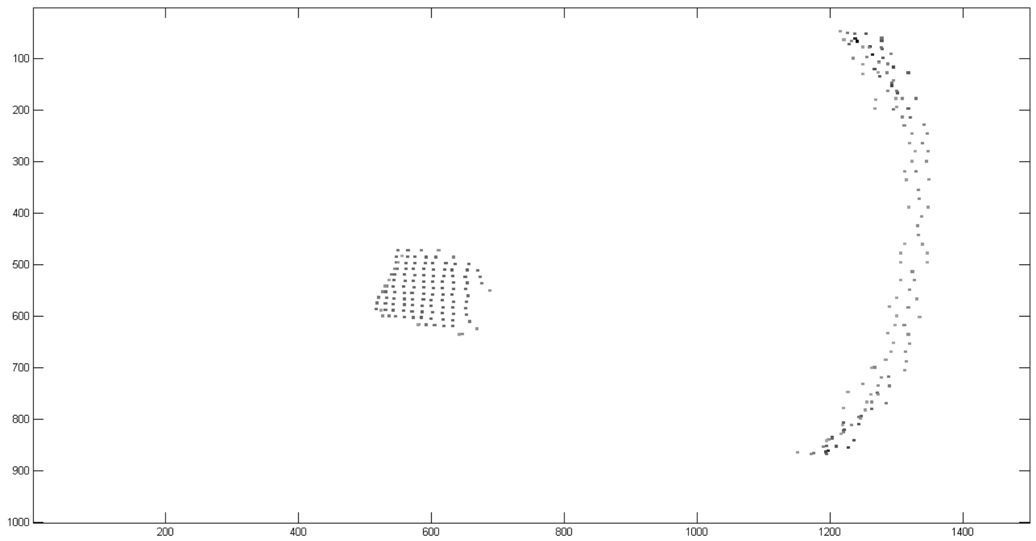
Mapping the objects

with the floor-plane located on the Y axis, the data can now easily be placed onto a 2D image, with the pixel intensity representing the hight above the Y-plane.



It can be seen that the result is not perfect, and some curving can be seen on the floor.

As the object is to find objects on the floor, a simple threshold is done to remove the floor plane. The threshold value is set to 4cm, since the block is 7cm high, this will always be seen.



The result is that only the block remains, as well as the curving at the edge of the camera, which is accentually higher then the block.

Matlab code

```
clear all
% close all
clc
angx = 52/180*pi;
angy = 0/180*pi;
angz = -3/180*pi;
high = 0.88;

VecX=load('angle/exportdata5_166.dat');
VecY=load('angle/exportdata6_166.dat');
VecZ=load('angle/exportdata7_166.dat');
figure(1)
plot3(VecX,VecY,VecZ,'b.',[0 VecX(1,1) VecX(64,1) 0 VecX(64,50) VecX(1,50) 0],[0 VecY(1,1) VecY(64,1) 0 VecY(64,50) VecY(1,50) 0],[0
VecZ(1,1) VecZ(64,1) 0 VecZ(64,50) VecZ(1,50) 0],'r')
axis([-1 1 -1 1 -1 1])
title('Default unit vector');
grid on
xlabel('X');
ylabel('Y');
zlabel('Z');
%turning along the X axis
VecT = (VecX.^2+VecY.^2+VecZ.^2).^0.5;
VecY2 = VecY*cos(angx)-VecZ*sin(angx);
VecZ2 = VecY*sin(angx)+VecZ*cos(angx);
VecX2 = VecX;
%turning along the Y axis
VecX3 = VecX2*cos(angy)-VecZ2*sin(angy);
VecZ3 = VecX2*sin(angy)+VecZ2*cos(angy);
VecY3 = VecY2;
%turning along the Z axis
VecX4 = VecX3*cos(angz)-VecY3*sin(angz);
VecY4 = VecX3*sin(angz)+VecY3*cos(angz);
VecZ4 = VecZ3;

VecX2 = VecX4;
VecY2 = VecY4;
VecZ2 = VecZ4;

VecT2 = (VecX2.^2+VecY2.^2+VecZ2.^2).^0.5;
figure(2)
plot3(VecX2,VecY2,VecZ2,'b.',[0 VecX2(1,1) VecX2(64,1) 0 VecX2(64,50) VecX2(1,50) 0],[0 VecY2(1,1) VecY2(64,1) 0 VecY2(64,50)
VecY2(1,50) 0],[0 VecZ2(1,1) VecZ2(64,1) 0 VecZ2(64,50) VecZ2(1,50) 0],'r')
axis([-1 1 -1 1 -1 1])
title('turned unit vector');
grid on
xlabel('X');
ylabel('Y');
zlabel('Z');

%loading the data and plotting it in a 3D format
img = load('exportdata1_46.dat');
img2X = img .* VecX2;
img2Y = img .* VecY2;
high = min(min(img2Y));
img2Y = img2Y -min(min(img2Y));
img2Z = img .* VecZ2;
figure(3)
plot3(img2X,img2Y,img2Z,'b',img2X',img2Y',img2Z','b')
title('Measurements using turned unit vector');
xlabel('X');
ylabel('Y');
zlabel('Z');
axis([-0.4 0.4 0 0.5 0 1.5])
grid on
figure(4)
grid on
plot3(img.*VecX,img.*VecY,img.*VecZ,'b',(img.*VecX)',(img.*VecY)',(img.*VecZ)','b')
title('Measurements using original unit vector');
grid on
xlabel('X');
ylabel('Y');
zlabel('Z');

%mapping data
map1 = zeros(1000,1500);
map2 = map1;
for i = 1:size(img2X,1)
    for j = 1:size(img2X,2)
        x = round(img2X(i,j)*1000)+500;
        y = round(img2Z(i,j)*1000);
        map1(x-2:x+2,y-2:y+2) = ones(5,5)*img2Y(i,j);
       if(img2Y(i,j) > 0.05)
            map2(x-2:x+2,y-2:y+2) = ones(5,5)*img2Y(i,j);
        end
    end
end

map1 = map1/max(max(map1));
map2 = map2/max(max(map2));
figure(10)
image(1-cat(3,map1,map1,map1))
figure(11)
image(1-cat(3,map2,map2,map2))
```

Shape detection

The purpose of this test is to see if round objects can be identified as round.

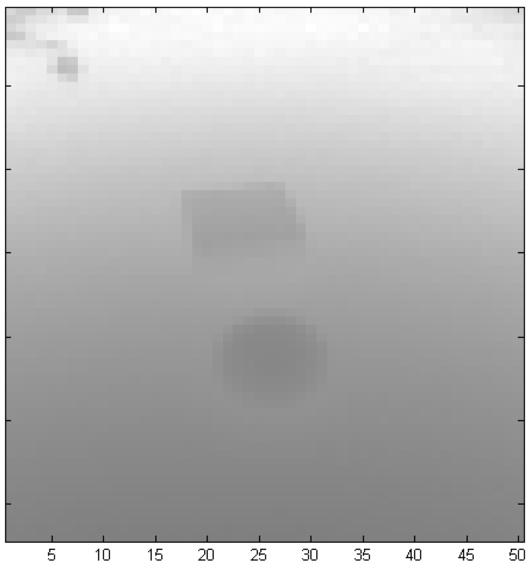
The test is done by placing a ball on the floor, and preforming the test used in the floor-plane extraction above.

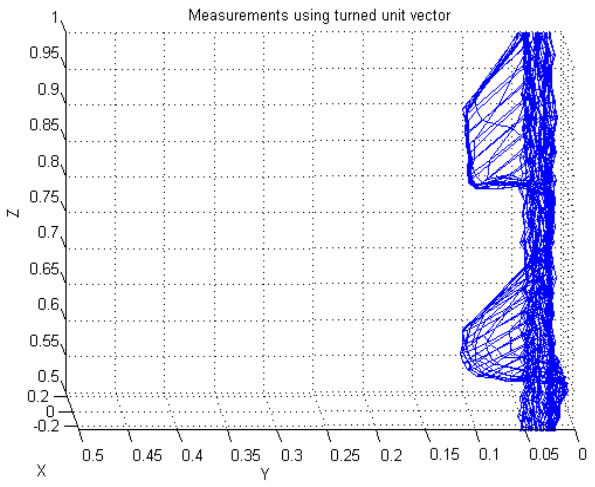
First test

The distance image can be seen to the right.

In the image there is a ball, a box and in the upper left corner part of a chair can be seen.

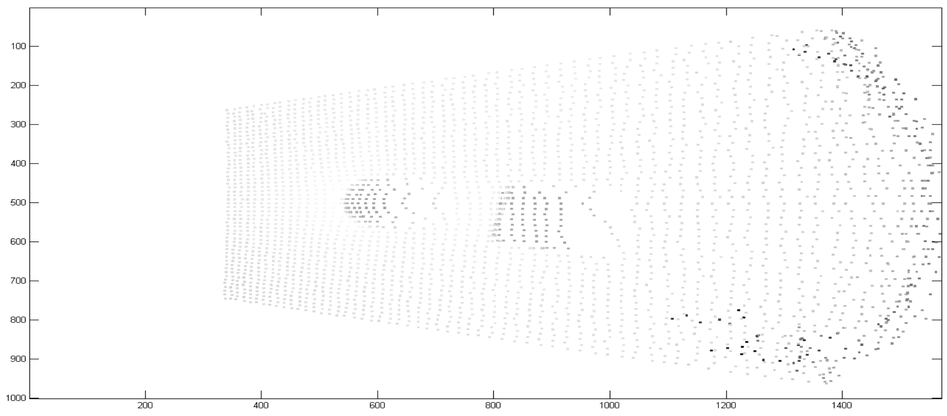
Moving the floor plane to the Y plane and generating a 3D picture. The objects seen from the side can be been below.





Some curving of the ball can be seen, but the object is not capture by enough pixels to really identify the form. It should be noted that it is easy to see the difference between a block and a ball.

Moving the data to a 2D image where colour represents the height above ground gives the following result. The curving of the ball can clearly be seen.



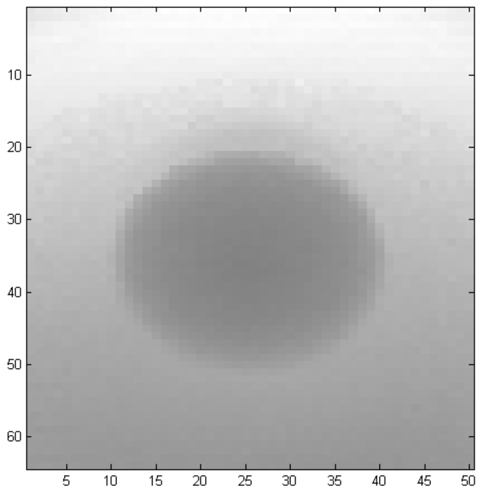
Second test

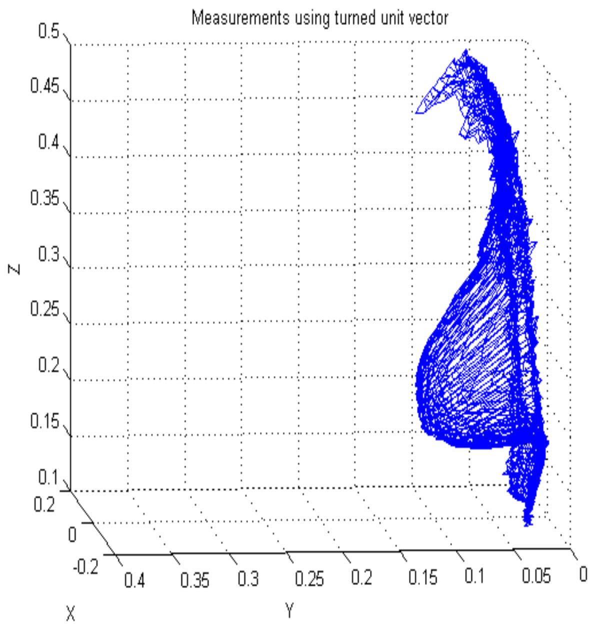
the second test is a close up of the ball. The distance picture can be seen below.

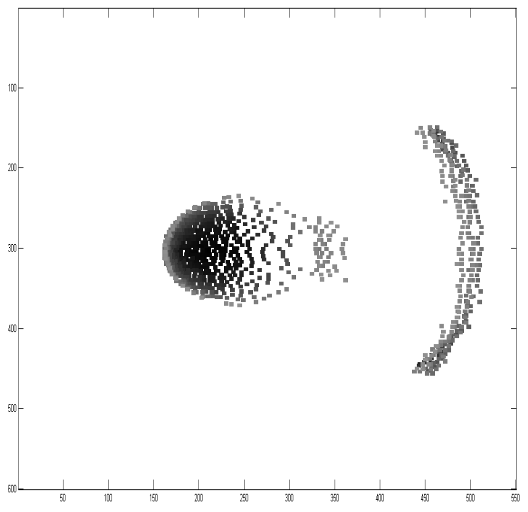
Translating this into a 3D image clearly shows the round shape, although the area behind the all is shades, which gives it a tail.

Below is the 3D image of the ball as well as a 2D threshold map.

The picture was taken at a hight of 35cm at an angle of 50°.



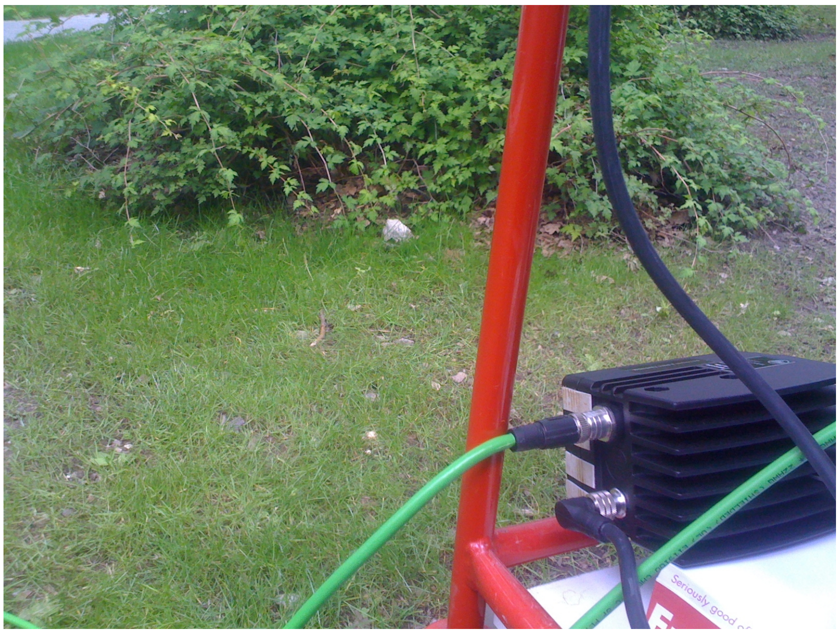




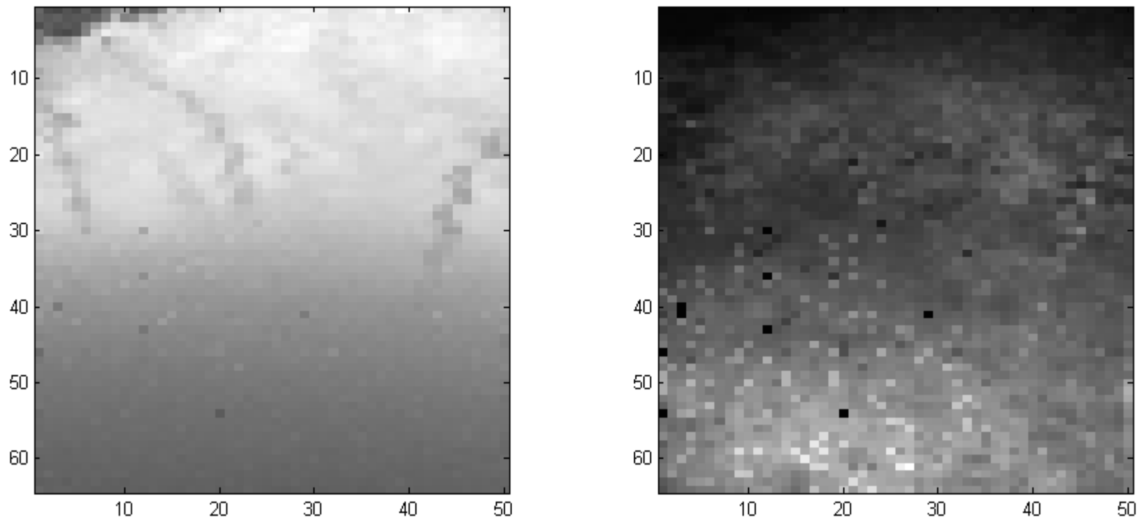
Vegetation measurement

The purpose of the test is to see how the camera reacts to vegetation. The camera was placed at a hight of about 0.45 meter with an angle of about 20°, and a picture was taken of grass and a bush.

Below is a picture of the test set up.

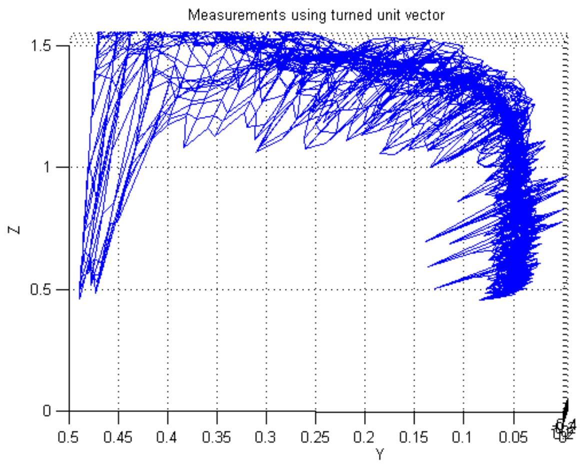


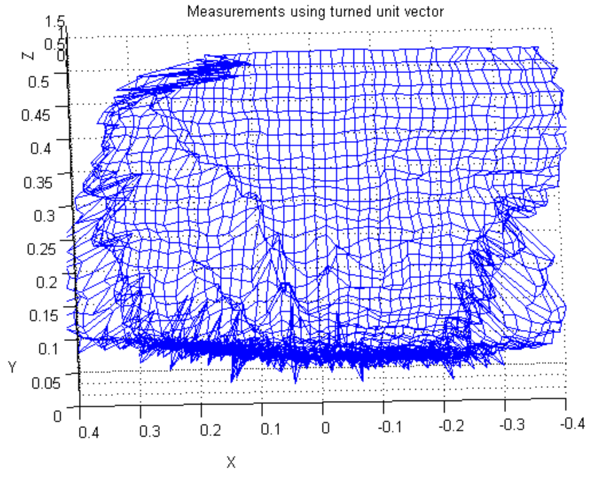
The measurement can be seen below. The left picture is the distance while the right is the intensity. The picture was taken with integrationtime 200/2000.



Using the pixel unit vectors, the following 3D graphs was generated, the left from the side, and the right for the cameras point of view.

In the side image, the grass can be seen to the right, the bush can be seen in the top and some overhanging branches can be seen to the left.





The vegitation did not hinder a picture from being generated.

Sunshine vs. cloudy weather

No direct test has been preformed on what the effects of sunshine is on the camera, however the previous test, vegetation, was preformed outside. The amount of sun shine did not appear to greatly affect the measurements.

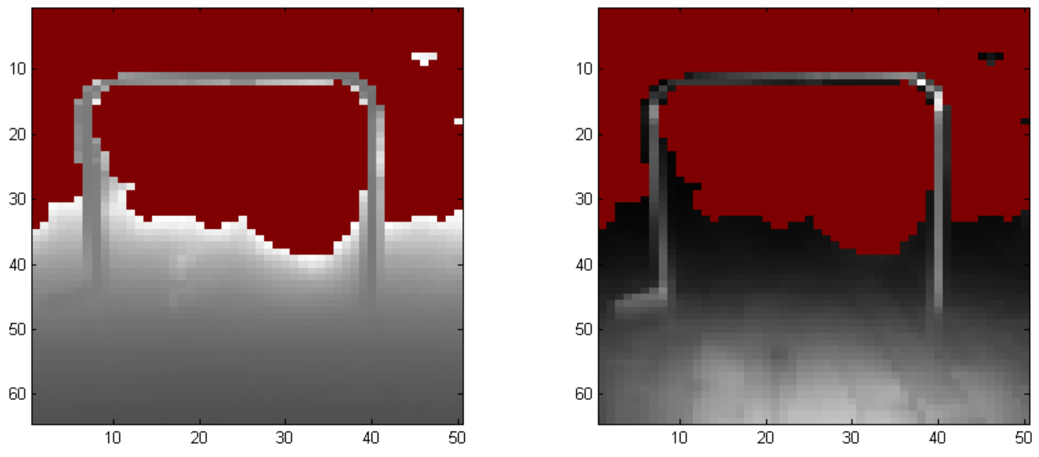
Several pictures was taken, both with direct sunshine, and with clouds in front of the sun, all using the same camera setting.

Gateway

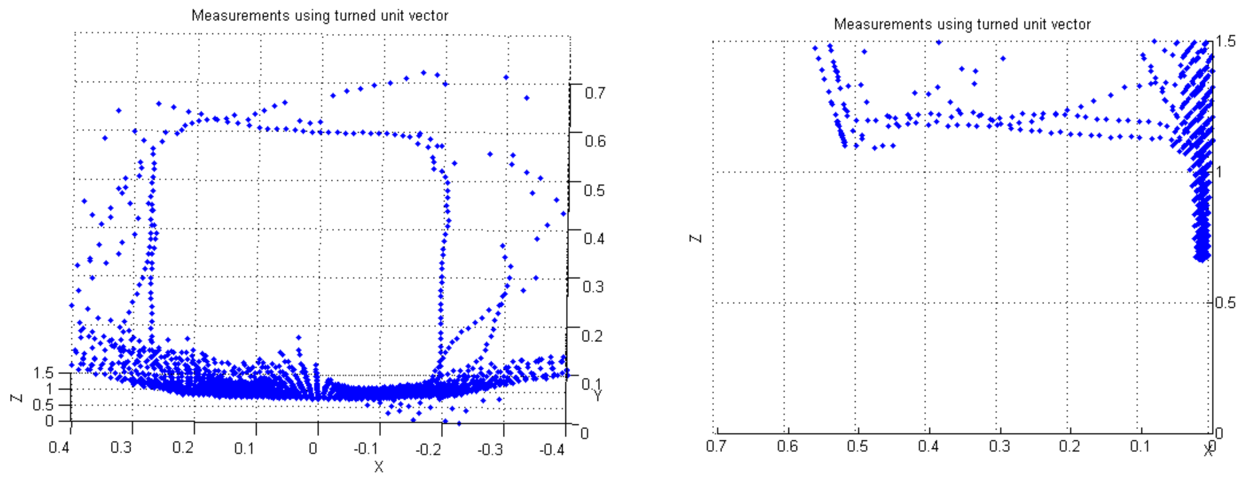
The purpose of this test is to see if the robot can spot a gateway and estimate its hight and position. he camera was placed at a hight of about 0.45 meter with an angle of about 20°. A gateway is placed in front of the camera at a distance of about 1.2 meter.

In this test, anything further away from the camera then 2,5 meter was ignored, as it is irrelevant for this test. Dual modulation was used to prevent far away object from appearing close.

Below is the image generated from the camera. To the left is the distance and to the right is the intensity.



This is converted into 3D below. The left is from the cameras point of view and the right is from the side.



For the images the gate is estimated to width 0.45m, height 0,50m and distance 1,1m.

The gate is measured to width 0.46m, height 0.49m and distance 1.2m.

Surgestions for software improvement

Image rotation

The default image rotation is with the camera buttons facing up and the the camera wires facing down. Since default direction is not always the best way to mount the camera, and the camera therefore may be mounted in any direction, the image should be rotated to match the mounted direction.

A variable should be implemented to indicated the rotation direction, and the image pool should be updated acordingly. This implementation would cause the image size to vary based on the rotation direction, and the plugin using the relevant image should be aware of this.

Heartbeat

The heartbeat is currently sendt every 5 sec, regardless of any other communication. It is only nessesary to have communication with the camera every 10 seconds, which means the hartbeat should only be sendt if there has been no other communication within the last 10 seconds.

This is a small optimization, and the gain would be minimal.

Line of sight vector time stamp

At the moment, the software compares the image header time stamp to see if it needs to update the image count, resulting in all images being taken at the same time having the same time stamp.

The line of sight vector X, Y and Z outputtet from the camera does not have the same time stamp as the rest of the images, and since it is currently being requested in the middle of the data (if it is requestet) the resulting image count for the different images type will be three different values. One value for all images loaded before the LOS vectors, One value for the LOS vectors and One value for the images requested after the LOS vector. Not only will it result in the picture counter counting three times faster then it is taking the pictures, the frame rate will also be completely wrong.

The LOS images should not have their timestamp checked.

It should be noted that the LOS vectors have been repeatetly checked, they remain the same regardslesss of setting or senary.

Increase detection range

It has been shown that the dual frequency function removed anything that is beyound the range of the 23MHz wrap around limit, it would be possible to manually use two different frequencies and merge these to get an image with ranges far beyound this.

It should be noted that a bigger power supply would have to be used to get a better signal.

Software code

there are two file, a header file and a C++ file. The two files are a modified version of the aukinect object.

Ufunctof.h

```
#ifndef UFUNC_AUTOF_H
#define UFUNC_AUTOF_H

#include <stdint.h>

#include <ugen4/uimg3dpoint.h>
#include <urob4/ufuncplugbase.h>
#include <ucam4/uimagelog.h>

#include <ugen4/ulock.h> // include the TCP/IP port thingy


// time stamp structure used in image header
typedef struct HeaderTimeStamp
{
float Seconds;
float Useconds;
};
//image header structure
typedef struct ImageHeaderInformation {
/** @brief Imagedata size in Bytes without header */
float DataSize;
/** @brief Size of the header */
float HeaderSize;
/** @brief type of image cf. IMAGE_HEADER::ImageTypes */
float ImageType;
/** @brief consecutive version number */
float Version;
/** @brief single or double integration */
float SamplingMode;
/** @brief illu status 0,1,2,3 bit coded */
float IlluMode;
/** @brief frequency mode cf. ModulationFrequency */
float FrequencyMode;
/** @brief unambiguous range of current frequency */
float UnambiguousRange;
/** @brief time needed by image evaluation [ms] */
float EvaluationTime;
/** @brief first integration time single sampling mode [ms] */
float IntegrationTime_Exp0;
/** @brief second integration time double sampling mode [ms] */
float IntegrationTime_Exp1;
/** @brief timestamp */
HeaderTimeStamp TimeStamp;
/** @brief median filter status */
float MedianFilter;
/** @brief mean filter status */
float MeanFilter;
float internal_a[4];
/** @brief displays if image is valid or not */
float ValidImage;
float ErrorCode;
float internal_b[3];
/** @brief configured trigger mode */
float CurrentTriggerMode;
float internal_c[4];
/** @brief Inter Frame Mute time*/
float IfmTime;
float internal_d[64];
/*picture data*/
```

```
float Data[50*64];
};

// the possible image types
enum ImageTypes
{
INVALID_IMAGE = 0,
DISTANCE_IMAGE,
INTERNAL_DATA_A,
AMPLITUDE_IMAGE,
INTERNAL_DATA_B,
NORMAL_X_IMAGE,
NORMAL_Y_IMAGE,
NORMAL_Z_IMAGE,
KARTESIAN_X_IMAGE,
KARTESIAN_Y_IMAGE,
KARTESIAN_Z_IMAGE,
INTERNAL_DATA_C,
SEGMENTATION_IMAGE
};

//the different commands







/**
 * This is a simple example plugin, that manipulates global variables, both of its own, and those of
other modules.
@author Christian Andersen
*/
class UFuncTOF : public UFuncPlugBasePush
{ // NAMING convention recommend that the plugin function class
  // starts with UFunc (as in UFuncKinect) followed by
  // a descriptive extension for this specific plug-in
public:
  /**
  Constructor */
  UFuncTOF()
  { // command list and version text
    setCommand("tof", "tof", "3D Time Of Flight camera");
    init();
  }
  /**
  Destructor */
  ~UFuncTOF();
  /**
   * Called by server after this module is integrated into the server core structure,
   * i.e. all core services are in place, but no commands are serviced for this module yet.
   * Create any resources that this modules needs. */
  virtual void createResources();
  /**
   * Handle incomming commands flagged for this module
   * \param msg pointer to the message and the client issuing the command
   * \return true if the function is handled - otherwise the client will get a 'failed' reply */
  virtual bool handleCommand(UServerInMsg * msg, void * extra);
  /**
  Called from push implementor to get the push object.
  should call to 'gotNewData(object)' with the available event object.
  if no object is available (anymore), then call with a NULL pointer. */
  virtual void callGotNewDataWithObject();
  //
private:
  /**
  Initialize other variables */
  void init();
  /**
  produce disparity image.
  \returns true if successful - i.e. source images available*/
  bool processImages(ImageHeaderInformation *Image);
  /**
   * check if setting is up to data, if not, update it
   */
  bool processSetting(void);
  /**
   * reads all the settings from the camera
   */
  bool getAllSetting(void);
  /**
   * call XML-RPC command and related functions
   */
  bool callXMLRPC(int command);
  bool ParseResponse(char* Response, char* Result);
  bool GenerateRequest(int type,char* Request);
  char* GetNextValue(char* reply);
  bool WaitForReply(char* Reply,int timeout, int limit);
  /**
  Handle stereo-push commands */
  bool handleTOF(UServerInMsg * msg);
  /// start read thread
  bool start();
  /** stop read thread
  \param andWait waits until thread is terminated, when false, then the call returns
  when the stop flag is set (i.e. immidiately). */
  void stop(bool andWait);


  /**
  Open or close camera stream */

public:
  /**
  Run receive thread */
  void runSetting();
  void runData();
private:
// variables used for settings
  char CameraName[15];
  char CameraVersion[8];
  int ModulationFrequencySetting;
  int DoubleSampling;
  int IntegrationTime1;
  int IntegrationTime2;
  int FrameMuteTime;
// free running
  int FreeRunning;
  int FreeRunningValue;
// image filters
  int NumberOfImagesAverage;
  int MeanFilter;
  int MedianFilter;

// variables used to pass internal data.
  volatile bool TriggerImage;
  volatile int ImagesMissing;

//time stamp of last image
  HeaderTimeStamp LastTime;
  UTime CurrentImageTime;
  UTime ImageRequestTime;
  /**
  Pointers to "own" global variables. */
  UVariable * varIsOpen;
// get images U-variables
  UVariable * varGetDistance;
  UVariable * varGetIntensity;
  UVariable * varGetNormalX;
  UVariable * varGetNormalY;
  UVariable * varGetNormalZ;
  UVariable * varGetKartesianX;
  UVariable * varGetKartesianY;
  UVariable * varGetKartesianZ;
//setting U-variables
```

```
  UVariable * varModulationFrequency;
  UVariable * varDoubleSampling;
  UVariable * varIntegrationTime1;
  UVariable * varIntegrationTime2;
  UVariable * varFrameMuteTime;
  UVariable * varFreeRunning;
  UVariable * varNumberOfImagesAverage;
  UVariable * varMeanFilter;
  UVariable * varMedianFilter;

  //image variables
  UVariable * varCamDeviceNum;
  UVariable * varFramerate;
  UVariable * varUpdateCnt;
  //debug
  UVariable * varExportData;

  /// thread runnung flag
  bool threadRunningSetting;
  bool threadRunningData;
  /// stop thread flag
  bool threadStop;

  /**
  Thread handle for frame read thread. */
  pthread_t threadHandleData;
  pthread_t threadHandleSetting;

  /*network ports*/
  UClientPort DataPort;
  UClientPort SettingPort;

};


#endif
```

Ufunctof.cpp

```
char CAMERA_IP[] = "192.168.0.69";
char ROBOT_IP[] = "192.168.0.99";
int SETTINGS_PORT = 8080;
int DATA_PORT = 50002;

#include <urob4/uvariable.h>
#include <urob4/uresposehist.h>
#include <urob4/uvarcalc.h>
#include <urob4/uimagepool.h>
#include <ucam4/ucammount.h>
#include <ucam4/ucampool.h>

#include "ufunctof.h"

#include <pthread.h>

#define MDAXMLConnectCP     0
#define MDAXMLHeartbeat     1
#define MDAXMLDisconnectCP  2
#define MDAXMLGetIP     3
#define MDAXMLSetIP     4
#define MDAXMLGetSubNetmask 5
#define MDAXMLSetSubNetmask 6
#define MDAXMLGetGatewayAddress 7
#define MDAXMLSetGatewayAddress 8
#define MDAXMLGetDHCPMode   9
#define MDAXMLSetDHCPMode   10
#define MDAXMLGetXmlPortCP  11
#define MDAXMLSetXmlPortCP  12
#define MDAXMLGetTCPPortCP  13
#define MDAXMLSetTCPPortCP  14
#define MDAXMLHeartBeat     15
#define xmlOPS_GetIORegister    16
#define xmlOPS_SetIORegister    17
#define MDAXMLSetWorkingMode    18
#define MDAXMLGetFrontendData   19
#define MDAXMLSetFrontendData   20
#define MDAXMLGetTrigger    21
#define MDAXMLSetTrigger    22
#define MDAXMLTriggerImage  23
#define MDAXMLGetDebounceTrigger    24
#define MDAXMLSetDebounceTrigger    25
#define MDAXMLGetAverageDetermination   26
#define MDAXMLSetAverageDetermination   27
#define MDAXMLGetProgram        28
#define MDAXMLSetProgram        29
#define MDAXMLSetMedianFilterStatus 30
#define MDAXMLGetMedianFilterStatus 31
#define MDAXMLSetMeanFilterStatus   32
#define MDAXMLGetMeanFilterStatus   33


#ifdef LIBRARY_OPEN_NEEDED

///////////////////////////////////////////////////
// library interface
// used by server when function is loaded to create this object

UFunctionBase * createFunc()
{ // create an object of this type
  //
  /** replace 'UFuncTOF' with your classname, as used in the headerfile */
  return new UFuncTOF();
}

#endif


UFuncTOF * tofObj = NULL;
```

```
///////////////////////////////////////////////////

UFuncTOF::~UFuncTOF() // skal rettes
{
  stop(true);
}

/////////////////////////////////////////////////

void UFuncTOF::init()
{
  tofObj = this;

  threadRunningData = false;
  threadRunningSetting = false;
  threadStop = false;

}

///////////////////////////////////////////////////

bool UFuncTOF::handleCommand(UServerInMsg * msg, void * extra)
{ // message is unhandled
  bool result = false;
  //
  if (msg->tag.isTagA("tof"))
    result = handleTOF(msg);
  else
    sendDebug(msg, "Command not handled (by me)");
  return result;
}

bool UFuncTOF::handleTOF(UServerInMsg * msg)
{
 bool result;

  // check for parameter 'help'
  if (msg->tag.getAttValue("help", NULL, 0) or msg->tag.getAttCnt() == 0)
  { // create the reply in XML-like (html - like) format
    sendHelpStart("TOF");
    sendText("--- Time Of Flight camera help\n");
    sendText("help is not done yet");
    sendHelpDone();
  }
  else if (msg->tag.getAttValue("open", NULL, 0))
  {
      printf("\r\nStarting device\r\n");
      SettingPort.setHost("192.168.0.69");
      SettingPort.setPort(8080);
      printf("host set: %s:%d\r\n",SettingPort.getHost(),SettingPort.getPort());
      printf("Trying to connect...\r\n");
      result = SettingPort.tryConnect();
      if(result) printf("OH MY GOD! it worked\r\n");
      else       printf("Failed as expected\r\n");

      result = SettingPort.isConnected();
      if(result)
      {
          printf("we are connected\r\n");
          printf("host IP set: %s\r\n",SettingPort.getHostIP());
          printf("Starting camera Session...\r\n");
          callXMLRPC(MDAXMLConnectCP);
          printf("getting setting...\r\n");
          callXMLRPC(MDAXMLGetFrontendData);//get setting
          callXMLRPC(MDAXMLGetTrigger);//check setting
          callXMLRPC(MDAXMLGetAverageDetermination);//check setting
          callXMLRPC(MDAXMLGetMedianFilterStatus);//check setting
          callXMLRPC(MDAXMLGetMeanFilterStatus);//check setting
```

```
        printf("\r\nopening data port\r\n");
        DataPort.setHost(CAMERA_IP);
        DataPort.setPort(DATA_PORT);
        printf("host set: %s:%d\r\n",DataPort.getHost(),DataPort.getPort());
        printf("Trying to connect...\r\n");
        result = DataPort.tryConnect();
        if(result)
        {
            printf("OH MY GOD! it worked\r\n");
            varIsOpen->setValued(1);
            start();
        }
        else
        {
            printf("Failed as expected\r\n");
            printf("Trying to close connection...\r\n");
            SettingPort.closeConnection();
            result = SettingPort.isConnected();
            varIsOpen->setValued(0);
        }
        printf("\r\n");
    }
    else
    {
        printf("we are NOT connected\r\n");
        varIsOpen->setValued(0);
    }

}
else if (msg->tag.getAttValue("close", NULL, 0))
{
    result = SettingPort.isConnected();
    if(result)
    {
       printf("stopping");
        stop(true);
       printf("stopped?");
        callXMLRPC(MDAXMLDisconnectCP);
        printf("we are connected\r\n");
        printf("Trying to close connection...\r\n");
        SettingPort.closeConnection();
        result = SettingPort.isConnected();
        if(result) printf("we are still connected\r\n");
        else       printf("we are no longer connected\r\n");
        varIsOpen->setValued(0);
        printf("close data connection...\r\n");
        DataPort.closeConnection();
        result = DataPort.isConnected();
        if(result) printf("we are still connected\r\n");
        else       printf("we are no longer connected\r\n");
        varIsOpen->setValued(0);
    }
    else
    {
        printf("we are NOT connected\r\n");
        varIsOpen->setValued(0);
    }

}
else if (msg->tag.getAttValue("imageget", NULL, 0))
{
    result = DataPort.isConnected();
    if(result)
    {
        if(FreeRunning == 0)
        {
            printf("getting images... \r\n");
            TriggerImage = true;
            while(TriggerImage == true); // wait for trigger
            while(ImagesMissing != 0); // wait for images
            printf("done\r\n");
        }
        else
```

```
          {
              printf("free running enabled, image update constant.");
          }
      }
      else
      {
          printf("we are NOT connected\r\n");
      }

  }
  else
  { // get any command attributes (when not a help request)
        printf("\r\nCommand unknown\r\n");
  }
  return true;
}

/////////////////////////////////////////////////////////////////

bool UFuncTOF::callXMLRPC(int command)
{
    const int limit = 10000;
    int sp = 0; //string pointer
    char Call[limit];
    char Reply[limit];
    char Variables[limit];
    //bool failed = false;
    char *CurrentValue;
    char *NextValue;

    //bool result;
    if(GenerateRequest(command,Call))
        return false;
    sp = strlen(Call); //find size
    if(!SettingPort.blockSend(Call, sp)) //send the string)
        printf("ERROR in sending, oh shit");

    Reply[0] = 0;
    if(WaitForReply(Reply,1000, limit))
        printf("timeout, end not found");
    sp = strlen(Reply);
    if(sp == 0)
        printf("no reply, this sucks");
    if(ParseResponse(Reply, Variables) == false)  // get variables from reply
        printf("cant read reply");
    CurrentValue = Variables;

    // three possible senarios, 1) response was a fault. 2) command failed, 3) command succes
    // check for fault

    if(strstr(CurrentValue,"FAULT;") != NULL)
    {
        printf("There is something wrong with the XMLCall: \"%s\"\r\n",CurrentValue);
        printf("command call(%d):\r\n%s\r\n",sp,Call);
        printf("\r\nreply(%d):\r\n%s\r\n",sp,Reply);
        return false;
    }
    // find error value
    NextValue = GetNextValue(CurrentValue);
    //check if command failed
    if(CurrentValue[0] != '0')
    {
        printf("the command could not be completet, error code: %s",CurrentValue);
        printf("command call(%d):\r\n%s\r\n",sp,Call);
        printf("\r\nreply(%d):\r\n%s\r\n",sp,Reply);
        return false;
    }
  //  while
    CurrentValue = NextValue; NextValue = GetNextValue(CurrentValue);
    switch(command)
    {
    case MDAXMLConnectCP:
        sprintf(CameraVersion,"%s",CurrentValue);
        CurrentValue = NextValue; NextValue = GetNextValue(CurrentValue);
        sprintf(CameraName,"%s",CurrentValue);
        printf("camera IDed, type: %s - version: %s\r\n",CameraName,CameraVersion);
        break;
//    case MDAXMLDisconnectCP:
//    case MDAXMLGetIP:
//    case MDAXMLSetIP:
//    case MDAXMLGetSubNetmask:
//    case MDAXMLSetSubNetmask:
//    case MDAXMLGetGatewayAddress:
//    case MDAXMLSetGatewayAddress:
//    case MDAXMLGetDHCPMode:
//    case MDAXMLSetDHCPMode:
//    case MDAXMLGetXmlPortCP:
//    case MDAXMLSetXmlPortCP:
//    case MDAXMLGetTCPPortCP:
//    case MDAXMLSetTCPPortCP:
    case MDAXMLHeartBeat: // no reply
        break;
//    case xmlOPS_GetIORegister:
//    case xmlOPS_SetIORegister:
    case MDAXMLSetWorkingMode:
        DATA_PORT = atoi(CurrentValue);
        break;
    case MDAXMLSetFrontendData:
    case MDAXMLGetFrontendData: //0,Modulation frequency,double
sampling,0,integration1,integration2,20,mute time
        CurrentValue = NextValue; NextValue = GetNextValue(CurrentValue); // error hint
        ModulationFrequencySetting = atoi(CurrentValue);
        CurrentValue = NextValue; NextValue = GetNextValue(CurrentValue);
        DoubleSampling = atoi(CurrentValue);
        CurrentValue = NextValue; NextValue = GetNextValue(CurrentValue); // always 0
        CurrentValue = NextValue; NextValue = GetNextValue(CurrentValue);
        IntegrationTime1 = atoi(CurrentValue);
        CurrentValue = NextValue; NextValue = GetNextValue(CurrentValue);
        IntegrationTime2 = atoi(CurrentValue);
        CurrentValue = NextValue; NextValue = GetNextValue(CurrentValue);//always 20
        CurrentValue = NextValue; NextValue = GetNextValue(CurrentValue);
        FrameMuteTime = atoi(CurrentValue);
        printf("(setting) ModulationFreq:%d - doubsamp:%d - integr1&2:%d/%d - mute:
%d\r\n",ModulationFrequencySetting,DoubleSampling,IntegrationTime1,IntegrationTime2,FrameMuteTime);
        break;
    case MDAXMLGetTrigger:
        FreeRunningValue = atoi(CurrentValue);
        if(FreeRunningValue == 3) FreeRunning = 1;
        else            FreeRunning = 0;
        printf("freerunning: %d\r\n",FreeRunning);
        break;
    case MDAXMLSetTrigger: break;
    case MDAXMLTriggerImage: break;
//    case MDAXMLGetDebounceTrigger:
//    case MDAXMLSetDebounceTrigger:
    case MDAXMLGetAverageDetermination:
      NumberOfImagesAverage = atoi(CurrentValue);
      printf("NumberOfImagesAverage: %d\r\n",NumberOfImagesAverage);

      break;
    case MDAXMLSetAverageDetermination:
      break;
//    case MDAXMLGetProgram:
//    case MDAXMLSetProgram:
    case MDAXMLSetMedianFilterStatus: break;
    case MDAXMLGetMedianFilterStatus:
        MedianFilter = atoi(CurrentValue);
        printf("MedianFilter: %d\r\n",MedianFilter);
        break;
    case MDAXMLSetMeanFilterStatus: break;
    case MDAXMLGetMeanFilterStatus:
        MeanFilter = atoi(CurrentValue);
        printf("MeanFilter: %d\r\n",MeanFilter);
        break;
    default: break;
    }
    return true;
}

bool UFuncTOF::WaitForReply(char* Reply,int timeout,int limit)
{
    int i = 0;
    int sp = 0;
    while(strstr(Reply,"</methodResponse>") == NULL && ((i*10) < timeout))
    {
        sp += SettingPort.getDataFromLine(Reply+sp,limit,10); // wait for reply
        i++;
        Reply[sp] = 0;
    }
    if((i*10) >= timeout)
        return true;
    else
        return false;
}
char* UFuncTOF::GetNextValue(char* reply)
{
    char* NextValue = strstr(reply,";");
    if(NextValue != NULL)
    {
        NextValue[0] = '\0';
        NextValue++;
    }
    return NextValue;
}


#define NUMBER_OF_TAGS 10

#define END            0
#define METHODRESPONSE 1
#define FAULT          2
#define PARAMS         3
#define PARAM          4
#define VALUE          5
#define MEMBER         6
#define NAMEt           7
#define I4             8
#define ARRAY          9
#define DATAt           10
const char* PossibleTags[] = {
"</",
"<methodResponse>",
"<fault>",
"<params>",
"<param>",
"<value>",
"<member>",
"<name>",
"<i4>",
"<array>",
"<data>",
"...",
"...",
"..."
};

bool UFuncTOF::GenerateRequest(int type,char* Request)
{
    const int limit = 1000;
    int sp = 0,sp2; //string pointer
    char XMLCall[limit];
    bool failed = false;
    //generate XML body
    sp+=sprintf(XMLCall+sp, "<?xml version=\"1.0\" encoding=\"UTF-8\"?
>\r\n<methodCall><methodName>");
    switch(type)
    {
        case MDAXMLConnectCP: sp+=sprintf(XMLCall+sp, "MDAXMLConnectCP");break;
        case MDAXMLDisconnectCP: sp+=sprintf(XMLCall+sp, "MDAXMLDisconnectCP");break;
        case MDAXMLGetIP: sp+=sprintf(XMLCall+sp, "MDAXMLGetIP");break;
        case MDAXMLSetIP: sp+=sprintf(XMLCall+sp, "MDAXMLSetIP");break;
        case MDAXMLGetSubNetmask: sp+=sprintf(XMLCall+sp, "MDAXMLGetSubNetmask");break;
        case MDAXMLSetSubNetmask: sp+=sprintf(XMLCall+sp, "MDAXMLSetSubNetmask");break;
        case MDAXMLGetGatewayAddress: sp+=sprintf(XMLCall+sp, "MDAXMLGetGatewayAddress");break;
        case MDAXMLSetGatewayAddress: sp+=sprintf(XMLCall+sp, "MDAXMLSetGatewayAddress");break;
        case MDAXMLGetDHCPMode: sp+=sprintf(XMLCall+sp, "MDAXMLGetDHCPMode");break;
        case MDAXMLSetDHCPMode: sp+=sprintf(XMLCall+sp, "MDAXMLSetDHCPMode");break;
        case MDAXMLGetXmlPortCP: sp+=sprintf(XMLCall+sp, "MDAXMLGetXmlPortCP");break;
        case MDAXMLSetXmlPortCP: sp+=sprintf(XMLCall+sp, "MDAXMLSetXmlPortCP");break;
        case MDAXMLGetTCPPortCP: sp+=sprintf(XMLCall+sp, "MDAXMLGetTCPPortCP");break;
        case MDAXMLSetTCPPortCP: sp+=sprintf(XMLCall+sp, "MDAXMLSetTCPPortCP");break;
        case MDAXMLHeartBeat: sp+=sprintf(XMLCall+sp, "MDAXMLHeartbeat");break;
        case xmlOPS_GetIORegister: sp+=sprintf(XMLCall+sp, "xmlOPS_GetIORegister");break;
        case xmlOPS_SetIORegister: sp+=sprintf(XMLCall+sp, "xmlOPS_SetIORegister");break;
        case MDAXMLSetWorkingMode: sp+=sprintf(XMLCall+sp, "MDAXMLSetWorkingMode");break;
        case MDAXMLGetFrontendData: sp+=sprintf(XMLCall+sp, "MDAXMLGetFrontendData");break;
        case MDAXMLSetFrontendData: sp+=sprintf(XMLCall+sp, "MDAXMLSetFrontendData");break;
        case MDAXMLGetTrigger: sp+=sprintf(XMLCall+sp, "MDAXMLGetTrigger");break;
        case MDAXMLSetTrigger: sp+=sprintf(XMLCall+sp, "MDAXMLSetTrigger");break;
        case MDAXMLTriggerImage: sp+=sprintf(XMLCall+sp, "MDAXMLTriggerImage");break;
        case MDAXMLGetDebounceTrigger: sp+=sprintf(XMLCall+sp, "MDAXMLGetDebounceTrigger");break;
        case MDAXMLSetDebounceTrigger: sp+=sprintf(XMLCall+sp, "MDAXMLSetDebounceTrigger");break;
        case MDAXMLGetAverageDetermination: sp+=sprintf(XMLCall+sp,
"MDAXMLGetAverageDetermination");break;
        case MDAXMLSetAverageDetermination: sp+=sprintf(XMLCall+sp,
"MDAXMLSetAverageDetermination");break;
        case MDAXMLGetProgram: sp+=sprintf(XMLCall+sp, "MDAXMLGetProgram");break;
        case MDAXMLSetProgram: sp+=sprintf(XMLCall+sp, "MDAXMLSetProgram");break;
        case MDAXMLSetMedianFilterStatus: sp+=sprintf(XMLCall+sp,
"MDAXMLSetMedianFilterStatus");break;
        case MDAXMLGetMedianFilterStatus: sp+=sprintf(XMLCall+sp,
"MDAXMLGetMedianFilterStatus");break;
        case MDAXMLSetMeanFilterStatus: sp+=sprintf(XMLCall+sp, "MDAXMLSetMeanFilterStatus");break;
        case MDAXMLGetMeanFilterStatus: sp+=sprintf(XMLCall+sp, "MDAXMLGetMeanFilterStatus");break;
        default: failed = true; break;
    }
    sp+=sprintf(XMLCall+sp, "</methodName>\r\n");
    switch(type)
    {
        case MDAXMLConnectCP: //first is the robot IP address, second is 1 for watchdog, 0 for no.
            sp+=sprintf(XMLCall+sp, "<params><param><value>%s</value></param><param><value><i4>
%d</i4></value></param></params>\r\n", ROBOT_IP,0);
            break;
        case MDAXMLDisconnectCP:
            sp+=sprintf(XMLCall+sp, "<params><param><value>%s</value></param></params>\r\n",
ROBOT_IP);
            break;
        case MDAXMLGetIP: break;
        case MDAXMLSetIP:
            sp+=sprintf(XMLCall+sp, "<params><param><value>%s</value></param></params>\r\n",
ROBOT_IP);
            break;
        case MDAXMLGetSubNetmask: break;
//        case MDAXMLSetSubNetmask: break;
        case MDAXMLGetGatewayAddress: break;
//        case MDAXMLSetGatewayAddress: break;
        case MDAXMLGetDHCPMode: break;
//        case MDAXMLSetDHCPMode: break;
        case MDAXMLGetXmlPortCP: break;
//        case MDAXMLSetXmlPortCP: break;
        case MDAXMLGetTCPPortCP: break;
//        case MDAXMLSetTCPPortCP: break;
        case MDAXMLHeartBeat: break;
//        case xmlOPS_GetIORegister: break;
//        case xmlOPS_SetIORegister: break;
        case MDAXMLSetWorkingMode: //1 arg, 0 for off, 1 for on
            sp+=sprintf(XMLCall+sp, "<params><param><value><i4>%d</i4></value></param></params>",1);
            break;
        case MDAXMLGetFrontendData: break;
        case MDAXMLSetFrontendData:
            sp+=sprintf(XMLCall+sp, "<params><param><value><i4>
%d</i4</value></param><param><value><i4>%d</i4</value></param><param><value><i4>
%d</i4</value></param><param><value><i4>%d</i4</value></param><param><value><i4>
%d</i4</value></param><param><value><i4>%d</i4</value></param><param><value><i4>
%d</i4</value></param><param><value><i4>%d</i4</value></param></params>",
0,ModulationFrequencySetting,DoubleSampling,0,IntegrationTime1,IntegrationTime2,20,FrameMuteTime);
            break;
        case MDAXMLGetTrigger: break;
        case MDAXMLSetTrigger:
            if(FreeRunning) FreeRunningValue = 3;
            else            FreeRunningValue = 4;
            sp+=sprintf(XMLCall+sp,"<params><param><value><i4>
%d</i4></value></param><param><value><i4>%d</i4></value></param><param><value><i4>
%d</i4></value></param></params>",0,0,FreeRunningValue);
            break;
        case MDAXMLTriggerImage: break;
        case MDAXMLGetDebounceTrigger: break;
//        case MDAXMLSetDebounceTrigger: break;
        case MDAXMLGetAverageDetermination: break;
        case MDAXMLSetAverageDetermination:
            sp+=sprintf(XMLCall+sp, "<params><param><value><i4>
%d</i4></value></param><param><value><i4>%d</i4></value></param><param><value><i4>
%d</i4></value></param></params>",0,0,NumberOfImagesAverage);
            break;
        case MDAXMLGetProgram: break;
//        case MDAXMLSetProgram: break;
        case MDAXMLSetMedianFilterStatus:
            sp+=sprintf(XMLCall+sp, "<params><param><value><i4>
%d</i4></value></param></params>",MedianFilter);
            break;
        case MDAXMLGetMedianFilterStatus: break;
        case MDAXMLSetMeanFilterStatus:
            sp+=sprintf(XMLCall+sp, "<params><param><value><i4>
%d</i4></value></param></params>",MeanFilter);
            break;
        case MDAXMLGetMeanFilterStatus: break;
        default: failed = true; break;
    }
    sp+=sprintf(XMLCall+sp, "</methodCall>\r\n");
    //generate header
    sp2 = sprintf(Request, "POST /RPC2 HTTP/1.1\r\nUser-Agent: XMLRPC++ 0.7\r\nHost: %s:
%d\r\nContent-Type: text/xml\r\nContent-length: %d\r\n\r\n",CAMERA_IP,SETTINGS_PORT,sp);
    sp2 += sprintf(Request+sp2,"%s",XMLCall);
    Request[sp2] = 0;
    return failed;
}

bool UFuncTOF::ParseResponse(char* Response, char* Result)
{
    char* RR;
    char* Rmin,*Rnow, *Rstop;
    int i, Imin = 0;
    int sp = 0;


    Rstop = strstr(Response,"</methodResponse>"); // find the end tag
    RR = strstr(Response,"<methodResponse>");//find the start tag
    if(Rstop == NULL || RR == NULL)
        return false;
    while(1)
    {        // keep looking for tags until no more is found
        Rmin = Rstop;
        for(i=0;i<NUMBER_OF_TAGS;i++)// look for the next tag by cycling through all the tags
        {
           Rnow = strstr(RR,PossibleTags[i]);
           if(Rnow != NULL && Rnow < Rmin)
           {
               Rmin = Rnow;
               Imin = i;
           }
        }
         // if no tag is found, stop loop
        if(Rmin == Rstop)
            break;
        // if next tag is a tag, find out what to do
        else
        {
            RR = strstr(Rmin,">")+1;//go to end of tag
//            printf("\r\n\"%s\"(%d) found[%c]: ",PossibleTags[Imin],Imin,RR[0]);
            switch(Imin)
            {
            case FAULT:// if fault write this in result
                sp += sprintf(Result+sp,"FAULT;");// +1;
                break;
            case VALUE: // write value to result
            case NAMEt: // write fault name to result
            case I4:    // write value to result
                if(RR[0] != '<')
                {
                 Rnow = strstr(RR,"<");
                 i = Rnow-RR;
                 memcpy(Result+sp,RR,i);
                 sp += i;
                 sp += sprintf(Result+sp,";");// finish with ';'
                }
                break;
            default: break;
            }
        }
        Result[sp] = 0;
    }
    //printf("\"%s\"\r\n",Result);
    return true;
}
/////////////////////////////////////////////////////////////////

void UFuncTOF::createResources()
{
    varIsOpen = addVar("open", 0.0, "d", "(r) is the camera open or closed");
//get images variables
    varGetDistance = addVar("GetDistance", 1.0, "d", "Requests the distance measurement.");;
    varGetIntensity = addVar("GetIntensity", 1.0, "d", "Requests the intensity measurement.");;
    varGetNormalX = addVar("GetNormalX", 0.0, "d", "Requests the direction vector X.");;
    varGetNormalY = addVar("GetNormalY", 0.0, "d", "Requests the direction vector Y.");;
    varGetNormalZ = addVar("GetNormalZ", 0.0, "d", "Requests the direction vector Z.");;
    varGetKartesianX = addVar("GetKartesianX", 0.0, "d", "Requests the distance converted to
Kartesian X.");;
    varGetKartesianY = addVar("GetKartesianY", 0.0, "d", "Requests the distance converted to
Kartesian Y.");;
    varGetKartesianZ = addVar("GetKartesianZ", 0.0, "d", "Requests the distance converted to
Kartesian Z.");;
//settings variables
    varModulationFrequency = addVar("ModulationFrq", 0.0, "d", "Sets the modulation frequency,
0=23MHz, 1=20.4MHz, 2=20.6, 3=double frequency.");
    varDoubleSampling = addVar("DoubleSampling", 1.0, "d", "enables double integration, using two
different integration times.");
    varIntegrationTime1 = addVar("IntegrationTime1", 100.0, "d", "Integration time 1, the
integration time sets the maximum length.");
    varIntegrationTime2 = addVar("IntegrationTime2", 1000.0, "d", "Integration time 2, the
integration time sets the maximum length.");
    varFrameMuteTime = addVar("FrameMuteTime", 70.0, "d", "Sets the non-transmition time, used for
frame frequency control, and to limit camera heating.");
    varFreeRunning = addVar("FreeRunning", 0.0, "d", "Set to 0 to only get pictures with
the \"ImageGet\" command. Set to 1 to continuously take pictures.");
    varNumberOfImagesAverage = addVar("NumberOfImagesAverage", 1.0, "d", "Sets the number of images
to combine to a finished image.");
    varMeanFilter = addVar("MeanFilter", 0.0, "d", "Sets the mean filter numbers. (temporal
filter)");
    varMedianFilter = addVar("MedianFilter", 0.0, "d", "Sets the median filter numbers. (spartial
filter)");
  //image variables
    varCamDeviceNum = addVar("CamDeviceNum", 20.0, "d", "Camera device number.");
    varFramerate = addVar("Framerate", 0.0, "d", "Time between updates.");
    varUpdateCnt = addVar("updCnt", 0.0, "d", "(r) Number of updates.");

    varExportData = addVar("ExportData", 0.0, "d", "export picture data to a datafile");
}
```

```
///////////////////////////////////////////

void UFuncTOF::callGotNewDataWithObject()
{
  gotNewData(NULL);
}

/////////////////////////////////////////////////////
bool UFuncTOF::processImages(ImageHeaderInformation *Image)
{
    int i,j,x,y;
    int PoolNumber;
    int temp[4];
    float TimeDif;
    char* Ip = (char*)Image;

    //change big indian to small indian
    for(i = 0;i < (int)sizeof(*Image);i +=4)
    {
       for(j=0;j<4;j++)
         temp[j] = Ip[i+j];
       for(j=0;j<4;j++)
         Ip[i+j] = temp[3-j];
    }

//    printf("image time: %f / %f \r\n",Image->TimeStamp.Seconds,Image->TimeStamp.Useconds);

    //check for new frame
    if(LastTime.Seconds != Image->TimeStamp.Seconds || LastTime.Useconds != Image-
>TimeStamp.Useconds)
    {
        CurrentImageTime.now();
        TimeDif = (Image->TimeStamp.Seconds - LastTime.Seconds) + (Image->TimeStamp.Useconds -
LastTime.Useconds) / 1000000;
        TimeDif = 1/TimeDif;
        varFramerate->setValued((int)TimeDif);
        varUpdateCnt->setValued(varUpdateCnt->getInt() + 1);
        LastTime.Seconds = Image->TimeStamp.Seconds;
        LastTime.Useconds = Image->TimeStamp.Useconds;
    }
    switch((int)Image->ImageType)
    {
    case 1://distance image
        PoolNumber = 70;
        ImagesMissing &=~0x01;
        break;
    case 3://intensity image
        PoolNumber = 71;
        ImagesMissing &=~0x02;
        break;
    case 5://normal x image
        PoolNumber = 72;
        ImagesMissing &=~0x04;
        break;
    case 6://normal y image
        PoolNumber = 73;
        ImagesMissing &=~0x08;
        break;
    case 7://normal z image
        PoolNumber = 74;
        ImagesMissing &=~0x10;
        break;
    case 8://kartesian x image
        PoolNumber = 75;
        ImagesMissing &=~0x20;
        break;
    case 9://kartesian y image
        PoolNumber = 76;
        ImagesMissing &=~0x40;
        break;
    case 10://kartesian z image
        PoolNumber = 77;
        ImagesMissing &=~0x80;
        break;
    default:
        return true;
    }

    //initialize image
    UImagePool * imgPool = (UImagePool *)getStaticResource("imgPool", false, false);
    UImage * depthBW = imgPool->getImage(PoolNumber, true);
    int16_t * DP;
    if (depthBW != NULL)
    {
        if (depthBW->tryLock())
        {
            depthBW->setSize(64,50, 1, 16, "BW16S");
            switch((int)Image->ImageType)
            {
            case 1:depthBW->setName("TimOfFlight distance value");break;
            case 3:depthBW->setName("TimOfFlight Reflection intensity value");break;
            case 5:
                depthBW->setName("TimOfFlight Normal X value");
                depthBW->getIplImage()->depth=IPL_DEPTH_16S;// allows for negative numbers
                break;
            case 6:
                depthBW->setName("TimOfFlight Normal Y value");
                depthBW->getIplImage()->depth=IPL_DEPTH_16S;// allows for negative numbers
                break;
            case 7:
                depthBW->setName("TimOfFlight Normal Z value");
                depthBW->getIplImage()->depth=IPL_DEPTH_16S;// allows for negative numbers
                break;
            case 8:
                depthBW->setName("TimOfFlight Kartesian X value");
                depthBW->getIplImage()->depth=IPL_DEPTH_16S;// allows for negative numbers
                break;
            case 9:
                depthBW->setName("TimOfFlight Kartesian Y value");
                depthBW->getIplImage()->depth=IPL_DEPTH_16S;// allows for negative numbers
                break;
            case 10:depthBW->setName("TimOfFlight Kartesian Z value");
                depthBW->getIplImage()->depth=IPL_DEPTH_16S; // allows for negative numbers
                break;
            }
            depthBW->imgTime = CurrentImageTime;
            depthBW->imageNumber = varUpdateCnt->getInt();
            depthBW->camDevice = varCamDeviceNum->getInt();
            depthBW->used = false;
            DP = (int16_t*)depthBW->getData();
            for(y=0;y<(64);y++)
            {
                for(x=0;x<(50);x++)
                {
                    *DP = (int16_t)(Image->Data[x*64+(y)]*1000);
                    if(Image->ImageType == 1 || Image->ImageType == 3) // if negative numbers are
not allowed
                    {
                        if(*DP < 0) *DP = 0;
                    }
                    DP++;
                }
            }

            depthBW->updated();
            depthBW->unlock();
        }

        if(varExportData->getInt())
        {
            FILE * fh;
            char filenamr[100];
            snprintf(filenamr,100,"exportdata%d_%d.dat",(int)Image->ImageType,varUpdateCnt-
>getInt());
            fh = fopen(filenamr,"w");
            for(y=0;y<(64);y++)
            {
                for(x=0;x<(50);x++)
                {
                    fprintf(fh,"%f ",Image->Data[x*64+(y)]);

                }
                fprintf(fh,"\r\n");
            }
            fclose(fh);
        }
    }
    return false;
}

///////////////////////////////////////////////////
// C function used to start a thread (will not work with object function
void * startUFuncTOFThreadSetting(void * obj)
{ // call the hadling function in provided object
  UFuncTOF * ce = (UFuncTOF *)obj;
  ce->runSetting();
  pthread_exit((void*)NULL);
  return NULL;
}
void * startUFuncTOFThreadData(void * obj)
{ // call the hadling function in provided object
  UFuncTOF * ce = (UFuncTOF *)obj;
  ce->runData();
  pthread_exit((void*)NULL);
  return NULL;
}

///////////////////////////////////////////////////
// thread start
bool UFuncTOF::start()
{
  bool result = true;
  pthread_attr_t  thAttr;
  //
  if (not (threadRunningData|threadRunningSetting))
  {
    pthread_attr_init(&thAttr);
    //
    threadStop = false;
    // create socket server thread
    result = (pthread_create(&threadHandleSetting, &thAttr,
              &startUFuncTOFThreadSetting, (void *)this) == 0);
    result  &= (pthread_create(&threadHandleData, &thAttr,
              &startUFuncTOFThreadData, (void *)this) == 0);
  }
  return result;
}

///////////////////////////////////////////////////
//thread stop
void UFuncTOF::stop(bool andWait)
{
  if (threadRunningSetting and not threadStop)
  { // stop and join thread
    threadStop = true;
    pthread_join(threadHandleSetting, NULL);
    if (threadRunningData)
    { // stop and join thread
      pthread_join(threadHandleData, NULL);
    }
  }
  if(andWait) while(threadRunningSetting || threadRunningData);
}

/////////////////////////////////////////////////////
// thread run function
void UFuncTOF::runSetting()
{
    // XML data
    // heartbeat
    UTime t, t2;
    // image request
    char ImageString[10];
    int StringCount;

    if (threadRunningSetting) // prevent nested calls;
        return;
    threadRunningSetting = true;
    t.now();
    while (not threadStop)
    {
    // check heartbeat / watchdog
        t2.now();
        if((t2.GetSec()-t.GetSec()) > 5)//time for heartbeat
        {
            if(SettingPort.isConnected())      // if port is open
            {
                if(callXMLRPC(MDAXMLHeartBeat) == false) //if errors, close ports
                {
                    SettingPort.closeConnection();
                    DataPort.closeConnection();

                }
//                  else
//                      printf("{H}");
            }
            t.now(); // get new time
        }


    // check setting
        if(SettingPort.isConnected())      // if port is open
        {
            if(ModulationFrequencySetting != varModulationFrequency->getInt() ||
               DoubleSampling != varDoubleSampling->getInt() ||
               IntegrationTime1 != varIntegrationTime1->getInt() ||
               IntegrationTime2 != varIntegrationTime2->getInt() ||
               FrameMuteTime != varFrameMuteTime->getInt())
            {
               ModulationFrequencySetting = varModulationFrequency->getInt();
               DoubleSampling = varDoubleSampling->getInt();
               IntegrationTime1 = varIntegrationTime1->getInt();
               IntegrationTime2 = varIntegrationTime2->getInt();
               FrameMuteTime = varFrameMuteTime->getInt();
               callXMLRPC(MDAXMLSetFrontendData);
            }
            if(FreeRunning != varFreeRunning->getInt())
            {
               FreeRunning = varFreeRunning->getInt();
               callXMLRPC(MDAXMLSetTrigger);
               callXMLRPC(MDAXMLGetTrigger);
            }
            if(NumberOfImagesAverage != varNumberOfImagesAverage->getInt())
            {
               NumberOfImagesAverage = varNumberOfImagesAverage->getInt();
               callXMLRPC(MDAXMLSetAverageDetermination);
            }
            if(MeanFilter != varMeanFilter->getInt())
            {
               MeanFilter = varMeanFilter->getInt();
               callXMLRPC(MDAXMLSetMeanFilterStatus);
               callXMLRPC(MDAXMLGetMeanFilterStatus);
            }
            if(MedianFilter != varMedianFilter->getInt())
            {
               MedianFilter = varMedianFilter->getInt();
               callXMLRPC(MDAXMLSetMedianFilterStatus);
               callXMLRPC(MDAXMLGetMedianFilterStatus);
            }
        }
    //image request check
        if(SettingPort.isConnected())
        {
    // if no pictures waiting, and free running or trigger request, request images
            if((FreeRunning || TriggerImage) && ImagesMissing == 0)
            {
                ImageRequestTime.now();
    //check for image trigger
                if(TriggerImage && FreeRunning == 0)
                    callXMLRPC(MDAXMLTriggerImage);
                StringCount = 0;
                if(varGetDistance->getValued() != 0)
                {
                    ImagesMissing |=0x01;
                    ImageString[StringCount++] = 'd';
                }
                if(varGetIntensity->getValued() != 0)
                {
                    ImagesMissing |=0x02;
                    ImageString[StringCount++] = 'i';
                }
                if(varGetNormalX->getValued() != 0)
                {
                    ImagesMissing |=0x04;
                    ImageString[StringCount++] = 'e';
                }
                if(varGetNormalY->getValued() != 0)
                {
                    ImagesMissing |=0x08;
                    ImageString[StringCount++] = 'f';
                }
                if(varGetNormalZ->getValued() != 0)
                {
                    ImagesMissing |=0x10;
                    ImageString[StringCount++] = 'g';
                }
                if(varGetKartesianX->getValued() != 0)
                {
                    ImagesMissing |=0x20;
                    ImageString[StringCount++] = 'x';
                }
                if(varGetKartesianY->getValued() != 0)
                {
                    ImagesMissing |=0x40;
                    ImageString[StringCount++] = 'y';
                }
                if(varGetKartesianZ->getValued() != 0)
                {
                    ImagesMissing |=0x80;
                    ImageString[StringCount++] = 'z';
                }
                ImageString[0] += 'A'-'a'; // make the first char upper case; this is to ensure it
gets the next image data
                ImageString[StringCount] = 0;
                if(StringCount > 0)
                {
                    if(DataPort.isConnected())
                    {
                        if(DataPort.blockSend(ImageString, StringCount) == -1)
                            printf("Send error!(%s)\r\n", ImageString);
                    }
                }
                TriggerImage = false;

            }
        }
    }
    threadRunningSetting = false;
}
void UFuncTOF::runData()
{
    // image data
    char ImageBuffer[sizeof(ImageHeaderInformation)];
    ImageHeaderInformation *CurrentImage;
    int sp = 0,sp2;
    // image timeout
    UTime t;

    if (threadRunningData) // prevent nested calls;
        return;
    threadRunningData = true;

    CurrentImage = (ImageHeaderInformation*)ImageBuffer;
    while (not threadStop)
    {
    //check for new data, only get data enough for one image
        if(DataPort.isConnected())
        {
            sp2 = DataPort.getDataFromLine(ImageBuffer+sp,sizeof(ImageHeaderInformation)-sp,5);
            if(sp2 == -1)
                DataPort.closeConnection();
            else
            {
                sp += sp2;
    // if enough data process image
                if(sp>=(int)sizeof(ImageHeaderInformation))
                {
                    processImages(CurrentImage);
    // removed processed data
                    sp = 0;
                }
            }
        }
   //check for image request timeout
        t.now();
        if(ImagesMissing != 0 && (t.GetSec()-ImageRequestTime.GetSec()) > 4)
        {
            t.now();
            ImagesMissing = 0;
            printf("\"imageget\" timeout");
        }
    }
    threadRunningData = false;
}
```