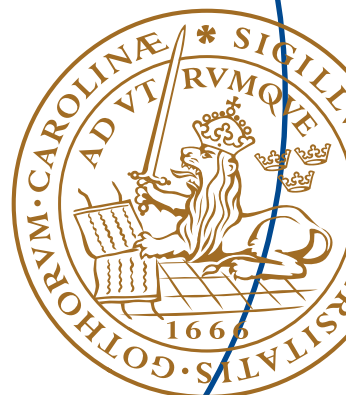


Master's Thesis

Virtual Image Sensor

Arvid Lindell
Johan Wennersten



Department of Electrical and Information Technology,
Faculty of Engineering, LTH, Lund University, March 2014.

Virtual Image Sensor

Arvid Lindell, Johan Wennersten
dt08a16, dt08jw2@student.lth.se

For Axis Communications AB

Advisors:
Mats Cedervall (LTH)
Carl-Axel Alm (Axis)
Axel Landgren (Axis)

March 5, 2014

Printed in Sweden
E-huset, Lund, 2014

Abstract

The primary goal of this thesis is to investigate the possibility of emulating an image sensor to provide alternative inputs to a network-connected surveillance camera. This is shown to be possible, by implementing such a system using an FPGA development platform. Using an unmodified camera main unit, alternative image data is accepted and forwarded through the network camera system, with the final result of it being viewable in the browser live-view.

A specific sensor example, a laser radar (LIDAR), is successfully implemented to demonstrate the concept, capable of providing a radar image of its surroundings.

Acknowledgements

We would like thank our university advisor Mats Cedervall, for all his uplifting encouragement and finding this thesis work for us. We would also like to thank Carl-Axel Alm for his unwavering enthusiasm throughout the course of the project, as well as all the encouragement and advice given. Furthermore, we would like to thank Axel Landgren for always being friendly and outwardly, and helping us with pretty much anything. We would also like to thank Igor Gurovski and Thomas Ekdahl, for explaining to us how electricity works.

Finally we would like to thank Glenn Svärd for helping us with the PCB design, Stefan Lundberg for technical advice and encouragement, and Per Kannermark and Andreas Nilsson for explaining vital parts of the camera imaging system.

Table of Contents

1	Introduction	1
1.1	Motivation	2
1.2	Original problem description	2
1.3	Thesis outline	3
2	Possible sensor alternatives	5
3	Technologies	7
3.1	AXIS P1214	7
3.2	Field-programmable gate array	7
3.3	VGA	8
3.4	Image sensor	8
3.5	Serializer-deserializer	9
3.6	Low-level serial communication	9
3.7	Stepper motor	10
3.8	Laser rangefinder	11
4	System overview	13
4.1	General method	13
4.2	Structure	13
5	Serial-deserializer and image sensor	19
5.1	Texas instruments DS92LX162x	19
5.2	I2C communication	19
5.3	Camera sensor	21
5.4	Image sensor emulation	21
6	Image sensor example	25
6.1	Pan-tilt mechanism	25
6.2	SICK DT50Hi	29
6.3	EMI reduction	32
6.4	Software	34
6.5	Other uses	35

7	Discussion	37
7.1	Alternative image output solutions	37
7.2	Acceleration curve considerations	37
8	Results	39
8.1	Sensor emulation	39
8.2	Sensor example	40
	References	41



Figure 1.1: AXIS P1214

The goal of this thesis is to consider the possibility and construction of an alternative image input system for an Axis network camera. A network camera contains an image sensor along with circuitry and software to transport captured image data over a network connection. The image data is encoded in a video stream, which can be viewed in a web browser.

An existing Axis camera solution (the P1214, pictured in Figure 1.1), has been used, which has its image sensor separated from the rest of the camera. A system has been designed which emulates the sensor subsystem using an FPGA. A complete example sensor has been designed and a few additional examples of possible sensor use cases will be described. An alternative sensor based on a laser distance sensor (LIDAR) has been designed, which is used to generate images similar to that of a typical radar device.

1.1 Motivation

A video management system (further referred to as VMS) is used to display video feeds from multiple cameras simultaneously. It allows an operator (or operators) to get an overview of the video feeds in an installation in an efficient manner.

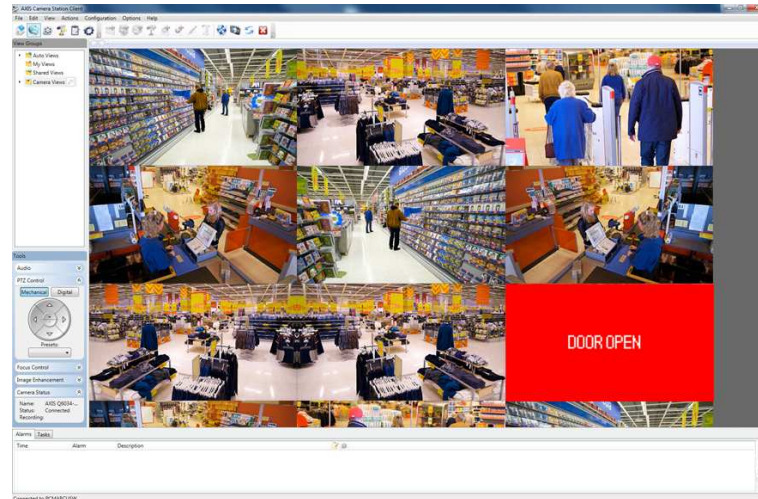


Figure 1.2: VMS

The primary motivation behind the virtual image sensor concept is the ease of integrating additional information sources into existing VMS systems. An example of a VMS client with a virtual camera sensor stream (door open/closed) is pictured in Figure 1.2. Another example could be a factory of some kind, with an already existing video surveillance system, where operators need additional information to complement to the video. Monitoring temperature in a cold storage room could be one such use case. Currently, a separate system would be needed to manage this extra category of data. With the suggested solution, the information from the thermometer could be displayed as just another video stream integrated into the already existing VMS.

1.2 Original problem description

Axis currently manufactures a camera system (called the P12-series) which offers connectivity to exchange the camera (image sensor) for something else. The sensor can be placed up to 8 meters (via wire) from the main unit, which in turn is connected to the network. This thesis will focus on the practical application of constructing a system which can be used as an alternative sensor for the existing P12-system. Our system will produce image information in a format identical to the existing image sensor, which will enable it to work on an unmodified P12-setup.

1.3 Thesis outline

Chapter 2 lists a number of sensor ideas. Some of the technologies used to complete the project are described in Chapter 3. Chapter 4 gives an overview of the system. In Chapter 5 the implementation of the image sensor emulation is presented in detail. Chapter 6 contains the description of the implemented alternate sensor system. Chapter 7 and 8 contains the compiled results, conclusions, some discussion, and possible future uses of the system that has been designed.

Possible sensor alternatives

This Chapter shows a few examples of possible alternative sensors which could be created together with the sensor emulation system, to give an idea of its possible uses.

Alternative sensor types could range from being an indicator of an open or closed door (as seen in Figure 1.2) to providing 3-dimensional image data, or the LIDAR system which is implemented in this thesis.

XY-graphs

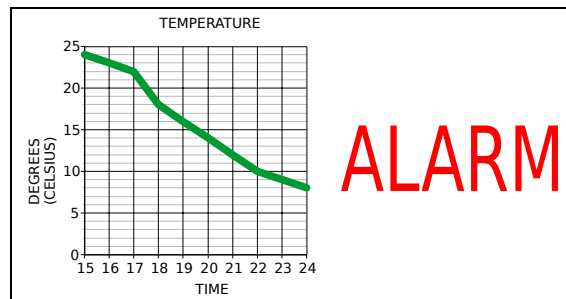


Figure 2.1: Temperature sensor

An example of a thermometer video stream is pictured in Figure 2.1. Any kind of input data representable in an XY-type graph could be generated by the same system configuration. Limits which should generate some type of action when breached, such as an alarm, could be added.

Environmental overviews

Figure 2.2 shows an overview of trains present on different tracks at a railway station, which could be achieved by connecting several simple motion detectors to the sensor emulation system. Similar systems for cargo ports or airports could be designed.

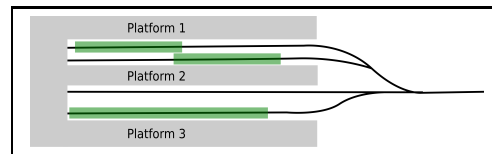


Figure 2.2: Railway station overview

Video retransmission

Another possible use case that could be considered is external video sources, that could be intercepted, reformatted and sent over the network in a seamless manner.

Numeric data

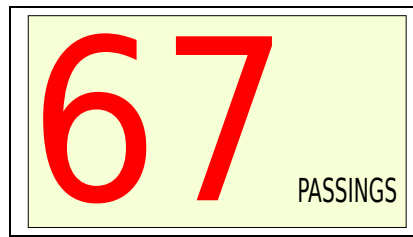


Figure 2.3: Displaying number of objects passed

Displaying one or several simple numeric values representing cars passed by a checkpoint, people through a gate, or similar could be useful in some cases. An example is pictured in Figure 2.3.

Other graphs

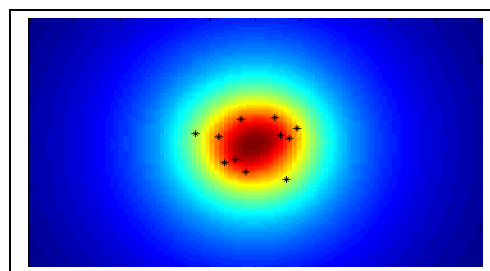


Figure 2.4: Heatmap sensor

Complex heat maps (see Figure 2.4), could be based on directional sound measurements, vibrations, or other data.

This Chapter serves as an introduction to the various different existing technologies that have been used in this project. These are general overviews, and more specific details of how they are used will be presented in Chapter 5 and 6.

3.1 AXIS P1214

The AXIS P12 series are network cameras designed for covert surveillance. They are designed such that the image sensor unit is separated from the rest of the camera. It is connected via a wired high speed serial link which can span up to 8 meters.

3.2 Field-programmable gate array

A field-programmable gate array (FPGA) is an integrated circuit with hardware that can be programmed to perform different operations. Basic logic functions (AND, OR, etc) are combined in large numbers in order to carry out complex computations. FPGAs also contain register elements to store values, and often blocks for specific computations or fast memory (called on-chip RAM). All of this is connected through a programmable interconnect system, which also provides access to components located outside the chip.

FPGAs are usually programmed in a hardware description language, of which the two most common are VHDL and Verilog. A small snippet of VHDL code can be seen in Figure 3.1 (a latch implementation), taking two inputs, D and Enable, and producing one output Q.

```
process(D, Enable)
begin
    if Enable = '1' then
        Q <= D;
    end if;
end process;
```

Figure 3.1: VHDL latch

3.3 VGA

VGA (Video Graphics Array) is a display standard introduced with the IBM Personal System/2 in 1987. [1]

The basic timing of VGA is provided by a pixel clock, at each rising edge a new pixel is sent to the display, in row order. When the end of a row is reached a horizontal sync signal is sent to the display for a specific duration to indicate that a new row is to be drawn. The original display type used with VGA, CRT displays, needed this time to move their electron beam back to its starting position at the next row. In a similar manner a vertical sync signal is sent once the bottom row is done to allow the beam to move back to the top.

In addition to this, there is an empty area around the entire image which only contains black pixels. This is usually divided into sections called the horizontal and vertical back and front porch. This can be used to center the image on the screen. [2]

3.4 Image sensor

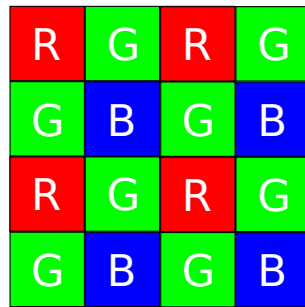
An image sensor is a device which converts an optical image into an electrical signal of some sort. The base unit of an image sensor is a pixel sensor, which reacts to incoming photons by the photoelectric effect, and produces a voltage proportional to the incoming light intensity. This voltage is then converted to a digital signal. The complete image sensor consists of a large matrix of single pixel sensors.

3.4.1 Bayer pattern

The individual pixel sensors are generally not capable of distinguishing between different colors within the span of wavelength which they react to, only the total light intensity. They are thus incapable of producing color images on their own. A typical way to solve this problem is to place a color filter array over the sensor. This filter lets through only a subset of the visible spectrum to each pixel sensor, typically each sensor is only reached by either red, green, or blue light. There are a variety of different filter layouts but the most common is the Bayer filter which can be seen in Figure 3.2. The resulting pixel data only contains partial color information in each pixel, and to recreate an approximation of the full color image (for display on a traditional computer monitor, for example) a process called demosaicing is performed, which is described briefly in the next section.

Demosaicing

A small section of an image in Bayer format can be seen in Figure 3.2. In this example a blue pixel would have its red and green color components approximated by averaging the values in the surrounding pixels of the corresponding colors. The same process is performed for each pixel in the entire image. [3]



R	G	R	G
G	B	G	B
R	G	R	G
G	B	G	B

Figure 3.2: 4x4 Bayer matrix

3.5 Serializer-deserializer

The original camera sensor is connected to the camera main unit through a serial link with a serializer on one end, and a deserializer in the other. A serializer takes several parallel inputs and combines them into a single faster stream of data. A deserializer performs the reverse process to split the stream into its original multiple connections. A serializer and deserializer pair is often referred to as a SerDes. [4]

3.6 Low-level serial communication

3.6.1 I2C

I2C is a bus protocol designed by Philips in the early 1980s. It uses two signals, SDA and SCL (data and clock). The basic configuration is a system consisting of a single master device and a number of slave devices. Each slave device is addressed using a unique identification number (usually 7 bits). The master generates the clock and initiates a data transmission by addressing the targeted slave device with its associated identification number, along with a single bit set for read or write operation (usually 8 bits total). The slave will ACK the transmission if received properly and act accordingly. A slave can never initiate a transmission.

The signals are in an open collector configuration, and can in a pull-up configuration be released by a bus peripheral to represent a logical one, or pulled low to represent a logical zero. [5]

3.6.2 Asynchronous RS232

Asynchronous RS232 is a communications protocol used in many digital systems. A bitrate is set initially by both receiver and sender (a typical transmission rate is 38400 bps). Separate lines are used for transmission and reception. A transmission typically consists of a start bit followed by 8 data bits, an optional parity bit, and a stop bit. [6]

3.7 Stepper motor

Stepper motors are a type of electrical motors which are rotated in discrete steps. The rotor is a permanent magnet, and it is typically surrounded by both ends of two electromagnets driven by two coils (see Figure 3.3).

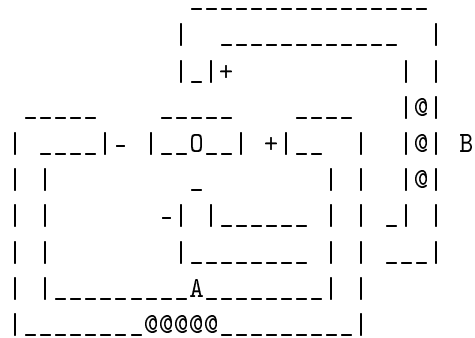


Figure 3.3: Stepper motor coils and rotor

By sending a current through the coils a magnetic field is created which affects the central magnet. By varying the direction and strength of the current in the coils, the direction of the resulting magnetic force can be controlled, which in turn rotates the central magnet in that direction.

The coils can be energized in sequence in several different ways, resulting in different types of steps. Using the designations A, !A, B and !B, where A and B are the different coils, and !A implies a current in the reverse direction, we can describe the different sequences.

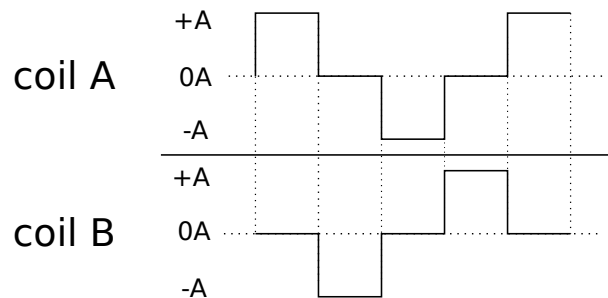


Figure 3.4: Stepper motor wave drive

The most basic form is the sequence A -> !B -> !A -> B -> A, which is called wave drive, is illustrated in Figure 3.4. By allowing both coils to be active at once, and with varying currents, more complex sequences can be used. Smaller steps can be achieved by doing more fine-grained changes to the currents.

In the case of this thesis, this process is controlled by an existing motor controller chip (called Big Easy Driver), which is sent control signals to perform steps. The resulting rotational speed is proportional to the frequency of this signal.

3.8 Laser rangefinder

A laser rangefinder measures distance by emitting a laserbeam towards a target and analyzing the reflected light. The most common technique of deciding the distance is to measure the time it takes from sending a laser pulse until it is recieved. Since the speed of light is known this information can be used to calculate the distance traveled by the laser beam, and thus also the distance to the object. The accuracy of the distance measurement depends on the accuracy of the time measurement.

The term laser radar, or LIDAR, refers to using a laser rangefinder to gather distance measurements of the surrounding world by aiming it in different directions.

System overview

As the main part of this thesis, a system has been designed, capable of emulating an image sensor (referred to as sensor emulation layer), and providing a stream of images which can be generated based on any input data. The system has also been extended to act as a specific sensor (called sensor example) providing distance data originating from a rotating laser distance sensor.

4.1 General method

An off-the-shelf FPGA development platform has been used, the Terasic DE2-115, which uses an Altera¹ Cyclone IV FPGA. Besides the FPGA itself, the board features a number of external components. This project uses the external 128MB SDRAM, and RS232 level conversion modules. [7]

Different vendors provide various pre-constructed HDL modules, such as CPUs, memory controllers, buses or other peripherals [8], which can be used together with user created designs. During this project, Altera's Qsys application has been used to configure the system with the above mentioned components.

The base system was configured with a Nios II/f soft-CPU (which is a 6-stage pipeline design with built-in instruction and data cache, [9]), 337.5 kB (out of a total 486 kB) of on-chip RAM used for the framebuffer, JTAG debug module, an SDRAM controller and RS232 module. The clock speed was set at 150 MHz. The JTAG uart is used to program the FPGA with a software executable and the RS232 UART is used to output debug messages to the connected computer. The base system was then extended with the custom built HDL modules pertaining to the sensor emulation and sensor example layer. When the system had been compiled and uploaded to the FPGA, software was written in C. The custom built modules were accessed from the software application using memory-mapped I/O.

4.2 Structure

Figures 4.1 and 4.2 contain block diagrams of the design. A photo of the actual system with the components marked can be seen in Figure 4.3.

¹FPGA manufacturer

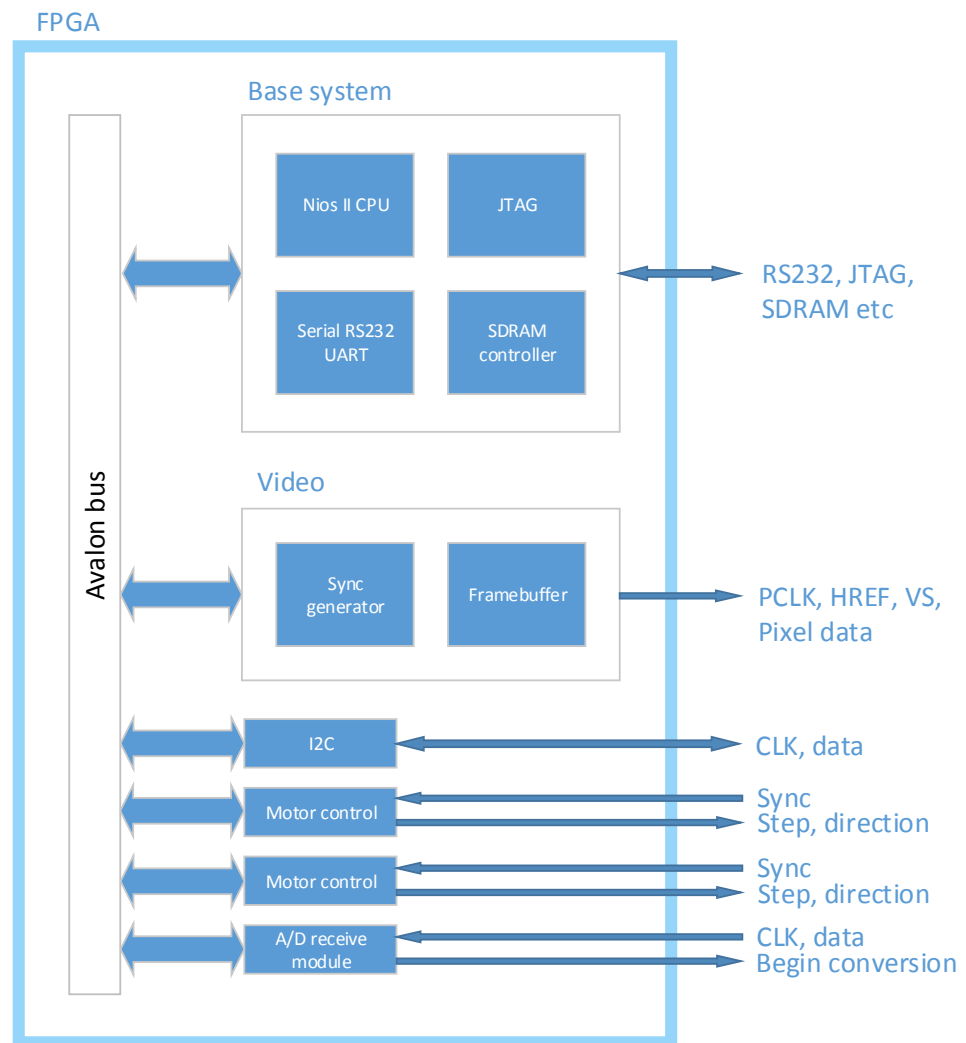


Figure 4.1: FPGA block diagram

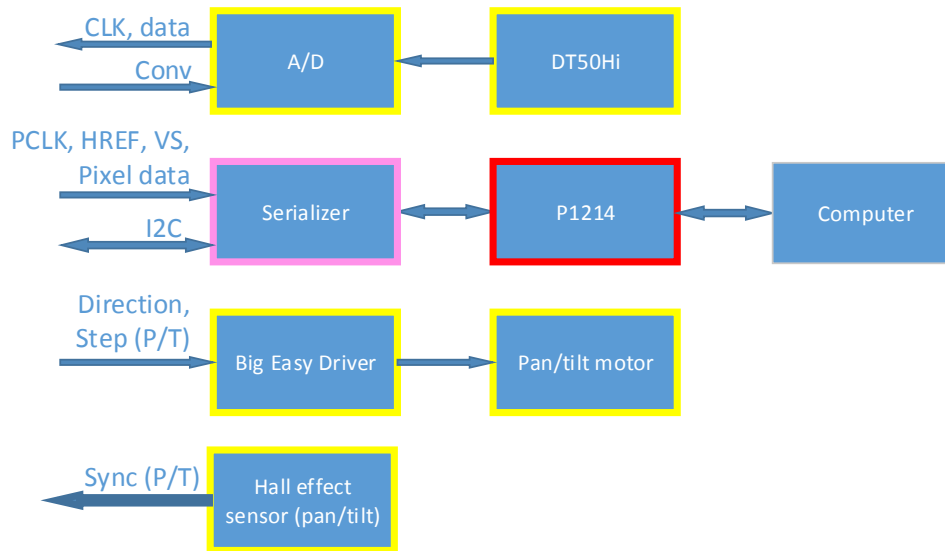


Figure 4.2: Components

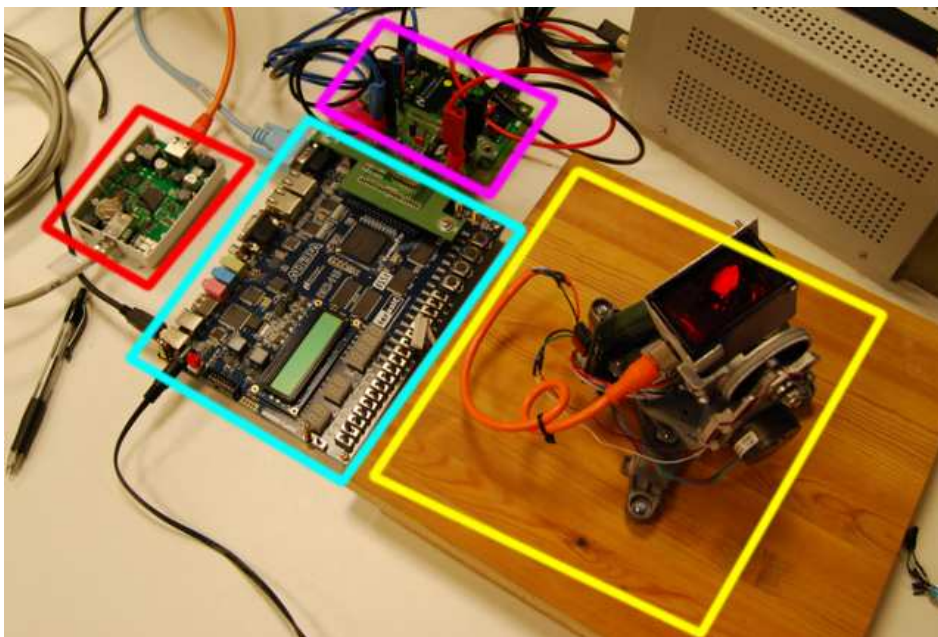


Figure 4.3: Actual system, P1214 is red, serializer is pink, DE2-115 is cyan and the pan-tilt platform is yellow

4.2.1 Sensor emulation

The lower layer is the basic sensor emulation, containing everything required to act as a replacement image sensor for the main unit. It provides the capability to correctly respond to I2C communication during the setup of the serial link, and to then stream arbitrary image data. It consists of a specialized hardware peripheral developed as a part of this thesis to generate correct video information, and a pre-existing I2C module (along with the base system). In Figures 4.1 and 4.2, the specific parts of the sensor emulation layer are the video, serializer, and P1214 block.

Details of the sensor emulation layer is described in Chapter 5.

4.2.2 Sensor example

The other part of the system consists of components required to implement our specific sensor example, the LIDAR sensor. These include FPGA components acting as motor controllers communicating with a motor driver chip and A/D conversion management communicating with an A/D converter chip. Additional external components used are the laser distance sensor itself, and a mechanical platform capable of pan- and tilt manouvers.



Figure 4.4: AXIS Q6032



Figure 4.5: SICK DT50Hi

The laser distance sensor used is a SICK DT50Hi (Figure 4.5), which provides an analog output signal in the form of a current ranging from 4mA to 20mA representing distance in the interval of 20 cm to 20 m, updated at 500 Hz with an accuracy of $\pm 7\text{mm}$.

The mechanical platform used for panning and tilting is taken from an AXIS Q60 camera (see Figure 4.4), and provides mounted stepper motors, two magnet sensors mounted to determine a known motor start position, and a slip ring interface to connect signals to the rotating top structure. The distance sensor has been mounted on this platform to enable it to acquire distance measurements in 360

degrees (pan) and -20 to 90 degrees (tilt, relation to the horizontal plane). The specific components in Figures 4.1 and 4.2 for the example sensor is the A/D, A/D receive module, motor controllers and Hall Effect sensors. The example sensor is described more thoroughly in Chapter 6.

Serial-deserializer and image sensor

This chapter details the process of emulating an image sensor and transferring the image data from the FPGA, via serial link, into the P1214 main unit.

Figure 5.1 shows the initialization procedure the camera goes through in order to set up the SerDes link and image sensor.

5.1 Texas instruments DS92LX162x

The SerDes which the camera uses is a Texas Instruments DS92LX162x, which is well suited for high-capacity video transfer. It is capable of transferring up to 900 Mbit/s, and supports a pixel clock between 10 and 50 MHz. It uses LVDS¹ for signalling. [10]

Before the sensor identification procedure takes place, the SerDes itself is initialized. This involves setting a couple of register parameters to correctly initialize I2C communication between the serial- and deserializer, and detecting a valid pixel clock signal. When a valid pixel clock has been detected, and the sensor has been identified, the P1214 will start to receive the transmitted image data. This setup procedure is pictured in Figure 5.1.

5.2 I2C communication

After initializing the SerDes itself, the P1214 requests an identification token from the sensor via I2C. If the sensor fails to correctly identify itself, the link will be reset, and the identification procedure will start over. If the sensor returns the correct token, the camera will start to accept the incoming video signals. In the initial setup, the camera firmware was modified to disable this check, which enabled testing of the generated video signals without having a functional I2C slave peripheral.

I2C was implemented after verifying that the sync signals were correctly received by the camera. Instead of designing an I2C-slave from scratch, Frank Buss's I2C-slave peripheral² which is written in VHDL and has been released into the

¹Low-voltage differential signalling, an EMI-resistant signalling scheme.

²Available at <http://www.velocityreviews.com/forums/t374657-p2-i2c-slave.html>.

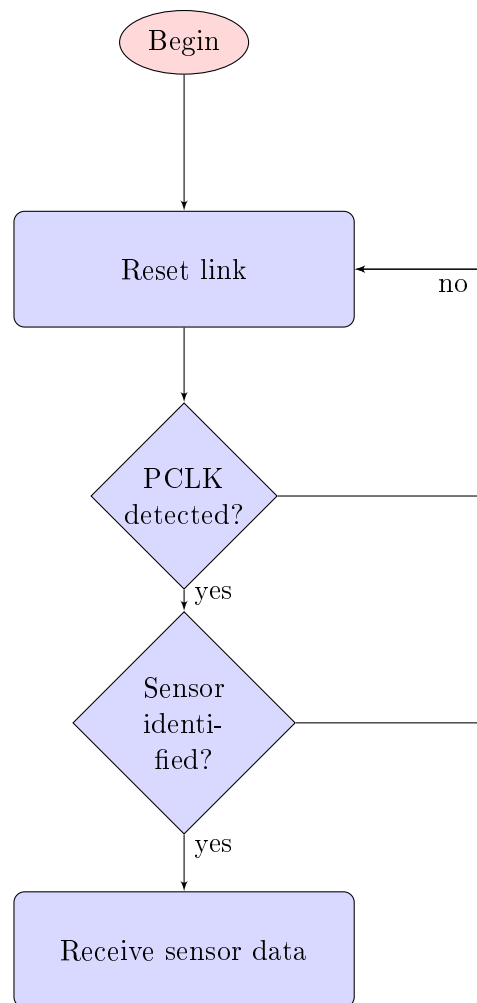


Figure 5.1: SerDes initialization flowchart

public domain was integrated into the system. Currently, the peripheral has been modified to automatically respond with the correct identification token required. Furthermore, the commands sent from the camera (exposure, gain, etc.) are received and can be replied to (the messages themselves are accessed from the software through an interrupt). Besides responding with the correct identification, this communications channel is currently not used for anything.

5.3 Camera sensor

The P1214 features a 1 megapixel CMOS image sensor. The external interface of the sensor is very similar to the VGA interface used on ordinary computer monitors (see section 3.3). Register-level communication with the P1214 is performed over I2C (see Section 5.2). Settings such as gain and white balance can be controlled.

The sensor supports WXGA resolution (1280x800 pixels) at 30 fps. Since the sides of the sensor are padded with a black material (to evaluate the black level of the sensor), this results in a final pixel output rate of about 42 MHz.

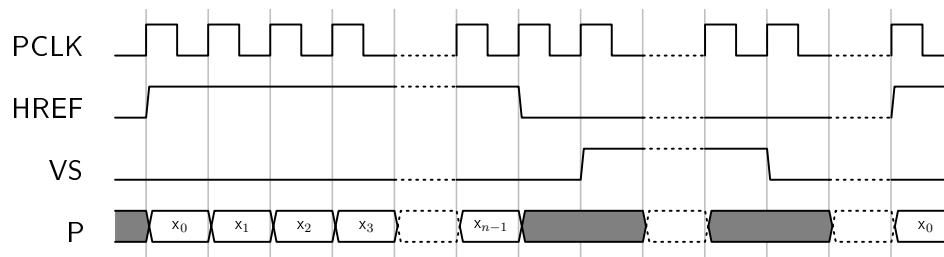


Figure 5.2: Sensor timing diagram

A typical data transmission from the sensor is pictured in Figure 5.2. When HREF is set, pixel data is regarded as valid and stored by the P1214. When a row has been transmitted, HREF is unset. The camera counts the number of rows received, since absolute location information is not supplied by the sensor. When a whole frame has been transmitted, the sensor sets VS for a number of clocks, and the process starts over.

5.4 Image sensor emulation

A custom hardware component was constructed in VHDL in order to emulate the sensor. The sensor interface is as mentioned in Section 5.3 similar to a normal VGA interface, so the initial implementation was a normal VGA controller, which was later modified to match the sensor sync signals. The emulation hardware consists of two modules. One module that generates the synchronization signals and one that stores the image data in a framebuffer.

5.4.1 Sync module

The sync module is driven by the pixel clock at 42 MHz, and in addition to generating HREF and VS, it also maintains the position of where we currently are in the image. These coordinates are then sent to the framebuffer which extracts the appropriate pixel and outputs it from the device.

5.4.2 Framebuffer

The image data is stored internally as 640x360 pixels, each pixel with R4G4B4 color (four bits per color component). The data is kept in fast on-chip RAM. Data is written to the framebuffer from the CPU via the Avalon bus system, and read from the pixel output process. Since pixels are output by the pixel clock at about 42 MHz, and the rest of the system runs at 150 MHz, the RAM module is dual ported and runs at two different clockspeeds. Due to the fact that data only flows in one direction, this does not cause noticeable problems. If the same pixel is written and read simultaneously, the output is undefined, and will result in a miscolored pixel in a single frame. It takes roughly 11 ms to set all pixels in the framebuffer from the software application.

5.4.3 Bayer inflation

The ordinary sensor is constructed with a Bayer-filter, so the main unit interprets the pixels received in accordance with this configuration. This means that to get the correct color the pixels need to be separated by color in the same way as a Bayer filter would.

As mentioned, image data is internally stored as 640x360 pixels, each with full R4G4B4 color. These full color pixels are separated into pixels containing one color component each. Each single full color pixel maps to 4 single color pixels, as seen in Figure 5.3.

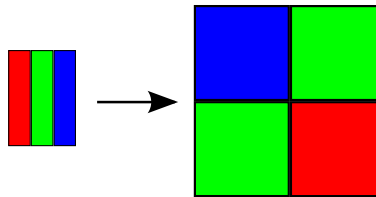


Figure 5.3: Single pixel to Bayer inflation

On the receiving end the missing colors for a given single color pixel are interpolated based on the surrounding pixels to recreate an approximation of the original image with full color pixels. This process is called demosaicing, and can be done in a variety of different ways. A general description of demosaicing can be found in Subsection 3.4.1, but the exact process done on the particular chip we are using is unknown to us, due to it being a trade secret of Axis.

5.4.4 Subpixel addressing

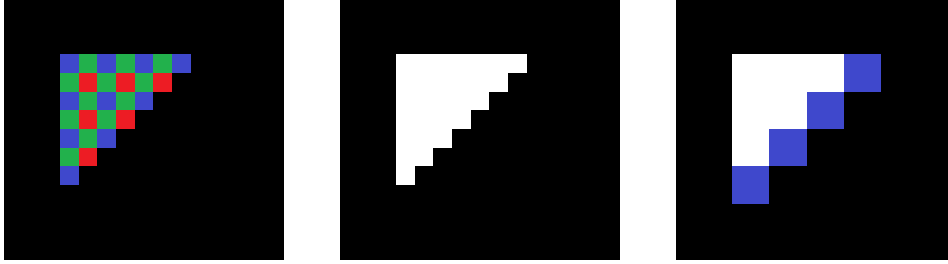


Figure 5.4: Camera demosaicing

Experimentation with addressing individual subpixels in the 1280x720 image was performed, with the expectation of smoother contours and finer detail. One test that was performed was an attempt to create a smoother diagonal transition between a white area and a black area in the image, with the end result of producing sharper lines. The result can be seen in 5.4, with the bayer output (left), the expected result (middle), and the actual result (right). As can be seen, the result does not correspond with the expectation.

5.4.5 Bits

The original sensor uses ten bits for each single color pixel, but to conserve on-chip RAM utilization our solution uses four. The images to be displayed are mainly simple figures and text, which in themselves do not warrant maximum color resolution. The four bits are placed as the four top bits out of the ten which are transmitted to the main unit, the others are tied low.

Alternative bit mappings were considered, such as having each individual bit map to two in the output, as seen in Figure 5.5, but was decided against. Using the top bits has the property of being linear in the interval, which the above mentioned alternative does not. To further conserve memory, a palette solution could be used.

$$\begin{array}{l|l|l} \text{4-bit data} & b_3b_2b_1b_0 & b_3b_2b_1b_0 \\ \text{10-bit data} & b_3b_2b_1b_0\{000000\} & b_3b_3b_2b_2b_1b_1b_0b_0\{00\} \end{array}$$

Figure 5.5: 4 to 10-bit conversion

5.4.6 Camera image enhancement



Figure 5.6: With correction to the left, without on the right.

Currently, the implementation does not correct for many of the raw sensor image enhancement routines the camera performs on the received image (see Figure 5.6). This incorporates, amongst other things, white balance correction.

Image sensor example

An example image sensor using LIDAR to create a radar image of its surroundings has been created, by controlling a mechanical pan-tilt platform upon which a laser distance sensor has been mounted. This Chapter describes the hardware and software needed to control the motors, interpret the incoming data, and present it as an image.

6.1 Pan-tilt mechanism

The process of controlling the pan-tilt platform in an efficient manner, while keeping track of its current position, is described in this section.

6.1.1 Stepper motor driver

The pan-tilt mechanism has two mounted stepper motors (Lin Engineering 3509v). A stepper motor requires a stepper motor driver, which provides varying currents to the coils as described in Section 3.7. An off-the-shelf driver chip, called Big Easy Driver (further referred to as BED) has been used, which provides a simple way to control the motor currents from the FPGA. Two BEDs are utilized, one for each motor. The BED takes 5 control inputs, step, direction and three inputs to select the step size. It is possible to choose from full steps, half steps, all the way down to sixteenth steps.

Initially, different combinations of frequency and step size were tested, for simplicity's sake the goal was to find a preferred step size. It was found that regardless of step size, the maximum rotational speed achievable stayed the same. The decision was made to go with the smallest steps, sixteenth steps, due to the fact that the motors made a much less noticeable noise when driven with this step size.

At first, for testing purposes, all of the motor signals were processed in software, which communicated with the BEDs through memory-mapped I/O-pins on the FPGA. This solution was not optimal, since it consumed a lot of CPU-time, and made the software unnecessarily complex. In an attempt to offload the CPU, a hardware-accelerated motor controller was implemented.

The hardware motor controller takes responsibility of sending a pulse-train of steps to the BEDs with a number of pulses and a frequency both specified

through software accessible registers. Instead of sending individual pulses, the CPU now only has to instruct the motor controller to send for example 100 pulses at a frequency of 10 kHz, and can then continue with other computations. The hardware also keeps track of the current motor position, based on how many steps it has been instructed to take. This information is available to the CPU through another register.

The motor controller can also be configured to interrupt to the CPU once a command has been completed. In some use-cases this functionality can be used to iterate over a list of motor commands in a interrupt-driven manner, starting the next once the previous is completed. More on this in Section 6.4.1.

Integrated into the motor controller is also the magnet synchronization system, which can be used to initialize the motor position information to a known state. It also serves to prevent any error in the rotational tracking to accumulate over multiple revolutions. This system is described more in the next section.

6.1.2 Magnet synchronization

Two Hall-effect sensors of type ALLEGRO A1104EUA-T, and their corresponding magnets which were already fitted on the Q60 construction, have been incorporated into the system. They are used for two main purposes, to find a known starting position, and to make sure that the estimated position does not drift when performing pan manoeuvres spanning multiple revolutions.

The sensor output is of open collector type, when a magnetic field is detected the output is connected internally to ground. Connecting the output via a pullup resistor to VCC results in a 0 if there is a magnet present, and a 1 if there is not.

Since, depending on configuration, the magnet detection both affects motor positioning information and can generate interrupts for the CPU, generating unnecessarily many magnet detection events needs to be avoided. The desirable outcome is a single transition from 1 to 0 and 0 to 1 in the output each time the sensors pass a magnet. The unmodified signal did not fulfill this requirement, and fluctuated between 1 and 0 near the edges of the magnet. The sensors themselves have a built in hysteresis threshold to help avoid this [11], but it turned out not to be sufficient.

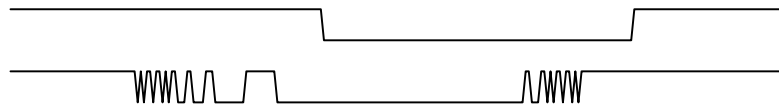


Figure 6.1: Filtered magnet sync signal (top)

In an effort to avoid having multiple magnet detection events for each magnet passing, a threshold system was added to the VHDL code handling the magnet signals coming in to the FPGA. It requires the magnet signal to be low for at least n clock cycles in a row before it is considered a true magnet detection, and then requires it to be high for another n cycles before it will consider the magnet as being passed. The number n is in this case configurable through a software accessible register. The result of this filtering is illustrated in Figure 6.1.

6.1.3 Acceleration curve construction

In order to make quick movements with the pan-tilt mechanism, acceleration curves must be used. If the motors are instructed to run at high speed without gradual acceleration, they will simply stop or jerk. Steps can also be lost intermittently when the acceleration and/or deceleration procedure is done too quickly [12]. If steps are lost, the accumulated stepper motor position information is no longer accurate.

An ‘S-curve’ based acceleration scheme has been used. The curve is based on a variant of the logistics function, which is given in equation 6.1. The parameter a controls the steepness of the curve, and b controls the midpoint (see Figures 6.2 and 6.3).

$$l(x) = \frac{1}{1 + 10^{-a(x-b)}} \quad (6.1)$$

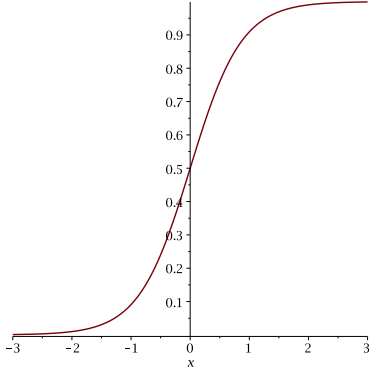


Figure 6.2: $l(x)$, with $a = 1, b = 0$.

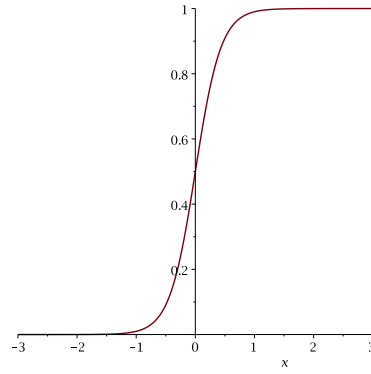


Figure 6.3: $l(x)$, with $a = 2, b = 0$.

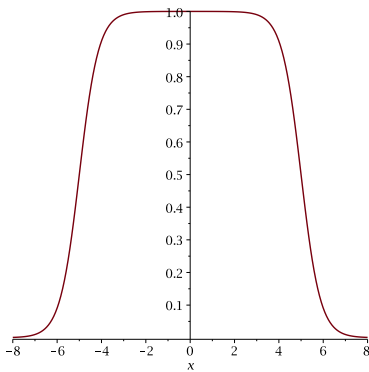


Figure 6.4: $c(x)$, with $a = 1, d = 10$.

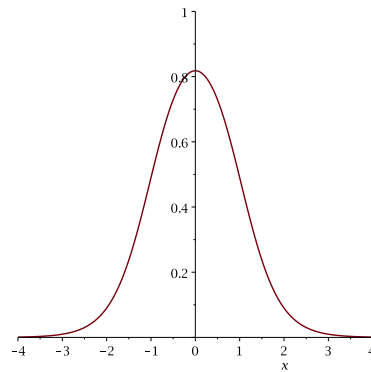


Figure 6.5: $c(x)$, with $a = 1, d = 2$.

The complete curve, $c(x)$, which represents velocity over time, is constructed according to equation 6.2, and two examples are pictured in Figures 6.4 and 6.5,

where d is the distance between the midpoints of the acceleration and deceleration part (see equation 6.2).

$$b_1 = -\frac{d}{2}, \quad b_2 = \frac{d}{2} \quad (6.2)$$

$$c(x) = \frac{1}{1 + 10^{-a(x-b_1)}} - \frac{1}{1 + 10^{-a(x-b_2)}}$$

The rationale for using the approach described above is as follows. Changing the acceleration rapidly produces ‘jerk’ in the motor motion, which is represented by the second derivative of the velocity curve. This is pictured in Figure 6.6. If the jerk produced by the change in acceleration is too profound, we might lose steps and invalidate our position information.

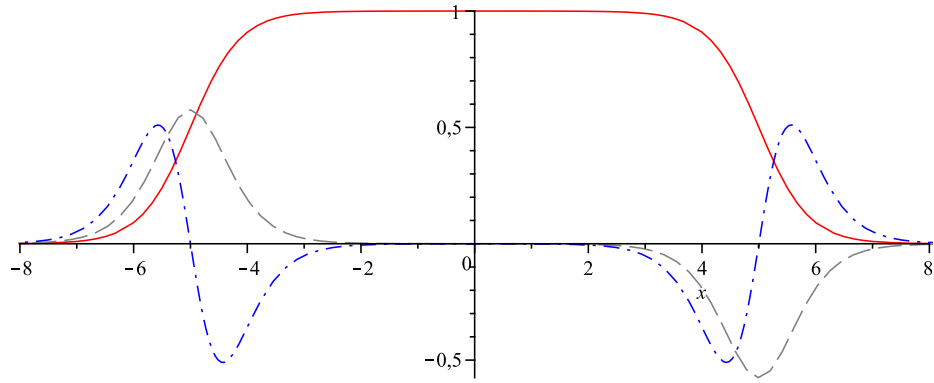


Figure 6.6: $c(x)$, with $a = 1, d = 10$, with its first (dashed) and second (dashed, dotted) derivative.

Implementation

When testing the viability of the scheme, an Octave-script was used to generate curves similar to those in Figures 6.4 and 6.5. The curve was sampled at a predetermined number of points, denoted below as c_i . Furthermore, f_{max} is the maximum frequency sent to the stepper motor drivers, and d is the number of steps to be taken divided by 32. For reference, a full revolution of the platform amounts to about 64000 steps. The frequency sent to the stepper motor drivers is further limited from f_{max} when performing small manoeuvres, according to equation 6.3 (which was determined through experimentation). Velocity in the given point is given by equation 6.4. The steps to be taken, s_{tot} , were distributed according to equation 6.5, which becomes a linear map of velocity versus step size.

$$f(d) = \frac{f_{cpu}/2}{\max(1700, 16000 - 200 \cdot d)} \quad (6.3)$$

$$v(i) = c_i \cdot f(d) \quad (6.4)$$

$$s(i) = c_i \frac{s_{tot}}{\sum_{n=0}^n c_n} \quad (6.5)$$

It was determined, through experimentation, that good results with the scheme described above were achieved by determining the parameter a according to equation 6.6. By further limiting the maximum frequency sent to stepper motor drivers (according to equation 6.3), and changing the a -parameter according to steps to be taken allows the scheme to work regardless of distance to be travelled. This allows a single software interface to control the motors.

$$a(d) = \max(0.05, \frac{8}{d}) \quad (6.6)$$

To generate acceleration curves on the fly in the software implementation, a fixed-point C-library, `libfixmath` is used, since the Nios II CPU does not have native floating point capability. By utilizing this approach, much of the flexibility of the Octave implementation is retained. An obvious alternative would have been to use a number of look-up tables for different speed profiles, which would have been cumbersome to regenerate if any of the parameters were to be changed.

Degrees	Time (s)	Degrees	Time (s)
3	0.125	27	0.397
6	0.221	30	0.431
9	0.271	45	0.543
12	0.275	60	0.636
15	0.232	75	0.717
18	0.270	90	0.795
21	0.313	180	1.201
24	0.355	300	1.684

Table 6.1: Motor manouvers of varying degree

To measure the performance of the implementation, elapsed time was measured by counting the number of times a 1 ms interrupt fired during the course of the manouver. Each manouver was repeated 8 times. The results can be seen in table 6.1. Furthermore, the top rotational speed is 250 degrees/s.

As can be seen in table 6.1, further parameter tweaking is warranted, since it takes longer to manouver 12 degrees than 15 degrees. Since the curve generation algorithm is implemented and run on the fly in software, it is easy to improve upon. It was, however, decided that the performance was adequate, and warranted no further study.

6.2 SICK DT50Hi

The SICK DT50Hi is a laser-based rangefinder with a 500 Hz output rate, 1 mm resolution, ± 7 mm accuracy, and up to 15 ms response time. It is capable of measuring from 20 cm to 20 m.

The output rate is the rate at which the output signal is updated, which in the DT50Hi's case is a current ranging from 4mA to 20mA (16bit resolution). The response time is defined as the maximum time until the output signal reports the 'correct' value. The rangefinder also uses a moving average over two samples when set to sample in 'fast' mode (which is the setting used).

6.2.1 Reading the DT50Hi output signal

The design uses a Texas Instruments ADS7813p successive approximation analog to digital converter [13], and the rangefinder distance signal is sampled at 10 kHz. Unfortunately, the signal from the rangefinder is affected by noise. This might be caused by the placement of the A/D on the constructed PCB, or other factors.

The following algorithm attempts to reconstruct the synchronous signal output by the rangefinder. The main goal of the reconstruction is to identify which of the oversampled values originate from the same original value created by the rangefinder, and to then calculate an average over these values and report it as the actual value. The averaging is used to reduce noise in the signal. This is done by collecting 20 values and trying to identify where the edge between different original values might be. The reasoning behind this is that a few different cases of 20 values can occur. Each case is displayed in Figure 6.7.

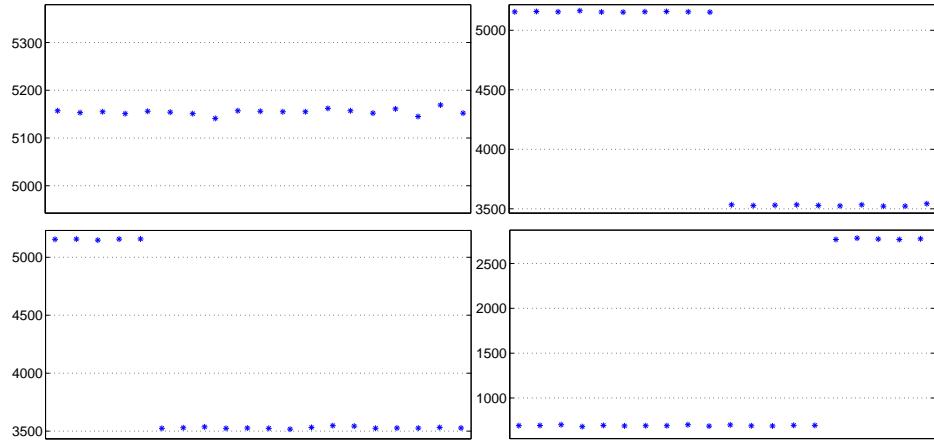


Figure 6.7: From top left: case 1, 2, 3, 4

After collecting 20 samples they are divided into four blocks of five, and their respective averages are calculated. The differences in average values between the blocks are also calculated. Using this information the algorithm attempts to identify the different cases according to the following scheme:

- If all of the block average differences are similar, select case 1.
- If the difference between blocks 2-3 is greater than that of 1-2 and 3-4, select case 2.

- If the difference between block 1-2 or 3-4 is greater than the others, select case 3 or 4 respectively.

When encountering case 2, the algorithm keeps the last block of five values for the next iteration, and effectively only consumes 15 values. The other cases result in collecting a full set of 20 new values. The reason for this is to avoid case 2 as much as possible, since it is not obvious if the values in block 1 and 2 or the ones in 3 and 4 should be considered the 'actual' value reported from the rangefinder there.

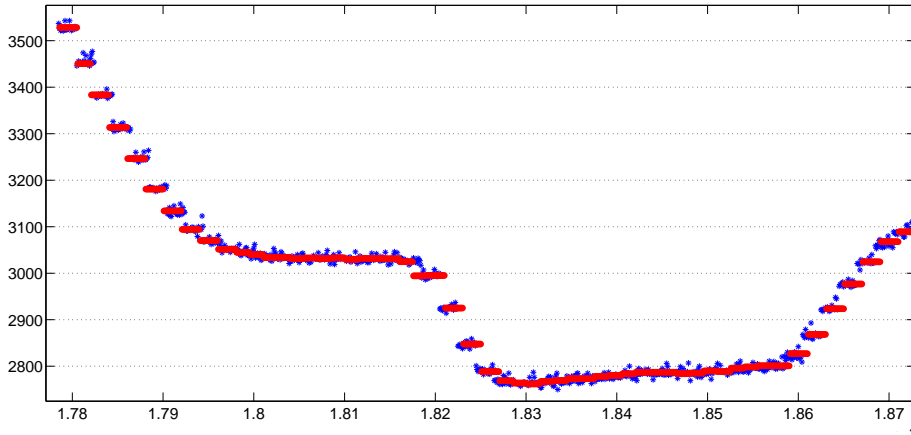


Figure 6.8: Plot of the reconstruction algorithm. Blue is the input signal, red is the synchronous estimation.

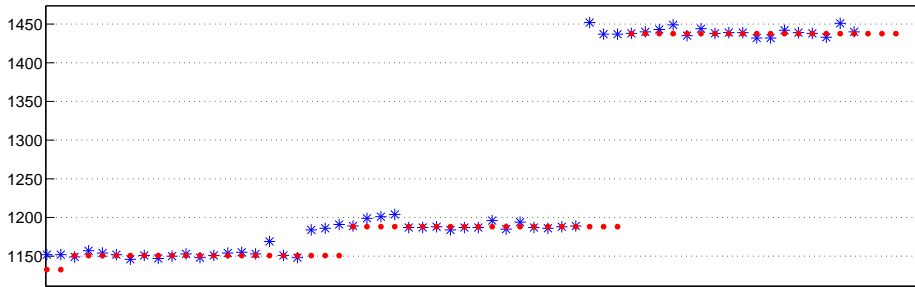


Figure 6.9: Algorithm closeup

To verify that the algorithm produces accurate values a baseline was first created by running the algorithm on actual data from the spinning LIDAR. Each data point of the baseline was duplicated 20 times, with individual noise added to each new value. The noise added was collected by targeting a stationary object with the LIDAR and collecting the fluctuation in the reported value, to make sure the added noise in the test was representative of the actual noise present. The algorithm was then run on this newly created data and the new result was

compared to the baseline. While the new result was not exactly the same as the baseline, due to the noise added, the result was very closely correlated to the actual 'real' values of the baseline data. A run of the algorithm is presented in Figure 6.8, and a closeup in Figure 6.9.

6.3 EMI reduction

During the course of the project, varying degrees of electromagnetic interference was present, which manifested itself in the system not functioning correctly.

6.3.1 First hardware iteration

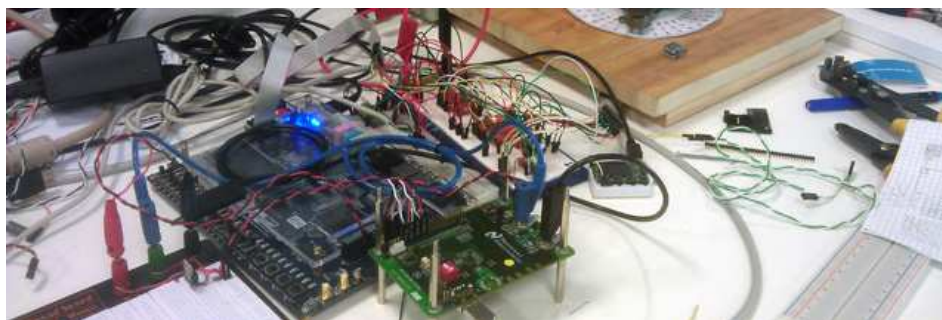


Figure 6.10: First iteration

During the first iteration (see Figure 6.10), a disturbance on the direction signal to the pan motor caused the motor to randomly change direction for a brief period of time ($< 1\text{s}$), which introduced significant jerk and invalidated the position information completely. This problem was mitigated by iteration two, in which all wires were shortened, and separate ground points were hooked up close to every digital control signal. This was done in an effort to try to achieve a common ground potential.

Another issue was that the sync signal from the magnets on the construction were bouncing profoundly when transitioning. An attempt to decouple the signal(s) with two 100 nF capacitors (one to ground and one to VCC), did not completely eradicate the problem. Debounce capability was then added to the motor controller, which completely solved the problem. This is described in Section 6.1.2.

6.3.2 Second iteration

The second iteration seemed to work well, without any noticeable disturbances to the digital control signals. The biggest problem was that a fair amount of noise was still present in the analog distance signal from the rangefinder to the A/D converter. Another problem was that the construction was very difficult to move around. These two issues warranted the construction of a printed circuit board.

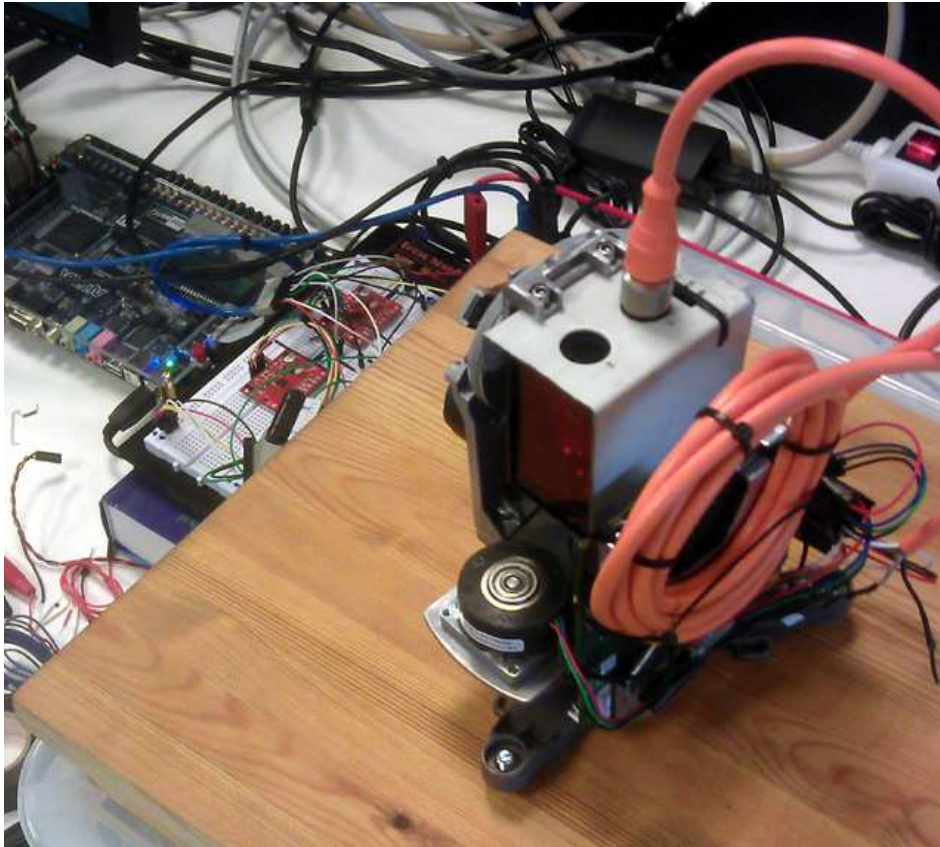


Figure 6.11: Second iteration, shorter wires and different breadboard

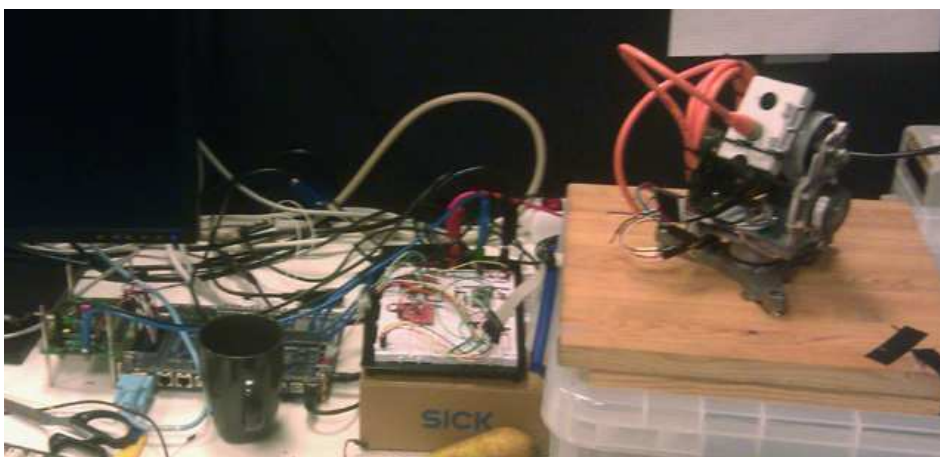


Figure 6.12: Second iteration

6.3.3 Third iteration

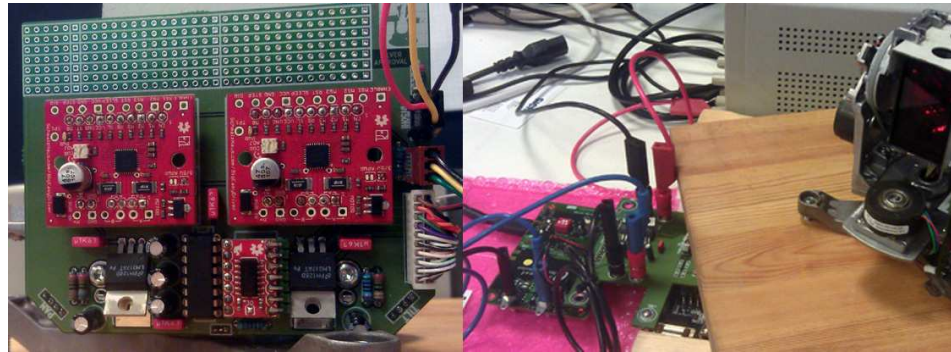


Figure 6.13: Third iteration, PCB constructed

The PCB was designed in KiCAD, which is an open-source PCB CAD suite. Two boards were designed, one to be mounted on the pan-tilt construction, and one to act as expander board for the FPGA (connecting the serializer, pan-tilt mechanism and the FPGA itself). Regarding the noise affecting the distance signal, data collected during the second iteration showed a deviation of ± 20 mm in the recorded value, and with the constructed PCB ± 10 mm. With the A/D mounted on the PCB, the algorithm described in Section 6.2.1 reduced the deviation to ± 5 mm. The measurements were done by sampling the rangefinder output signal, while being pointed on a stationary object, and taking note of the recorded distance value reported on the LCD display of the device itself.

6.4 Software

6.4.1 Motor command lists

Using the scheme described in 6.1, linked lists consisting of commands to be sent to the motors were constructed (a simple example could be; take 200 steps right at speed 10 and then take 220 steps right at speed 8 etc). The motor controllers were configured to interrupt the CPU when the command had been performed, the interrupt routine then sent the next command in the list to the relevant motor controller (pan or tilt).

6.4.2 P1214 radar

After implementation, the design was able to be used as a radar connected to the P1214 (see Figure 6.14). The software consists of a look-up table for sinus and cosinus, for drawing the recorded distance in polar coordinates, functionality to set individual pixels in our framebuffer and the algorithm documented in Section 6.2.1 for reading the signal from the A/D converter. The motor control scheme described in 6.1 was used to control the motors. The main software loop is described below.

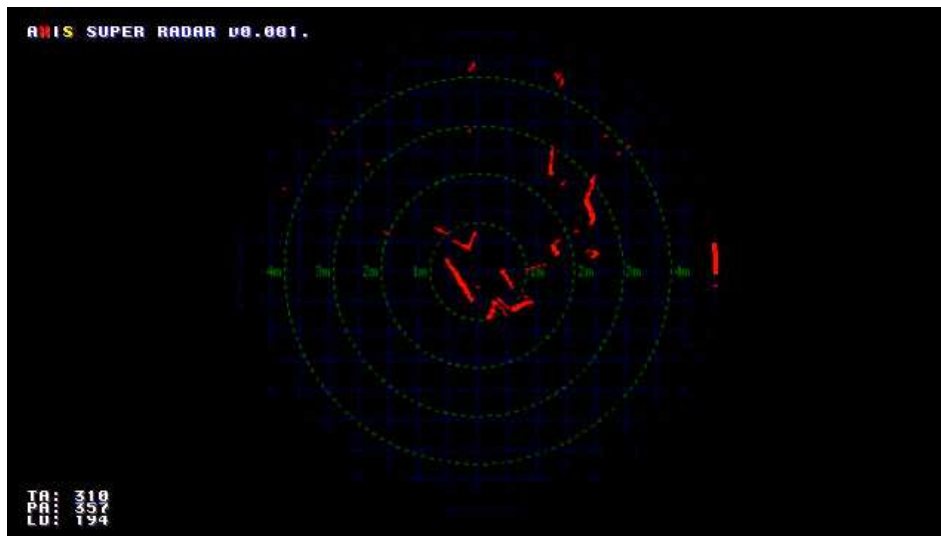


Figure 6.14: P1214 radar

```
initialize the motors (set to known starting position)
enable interrupts
while (forever)
    if (a new distance value has been reported)
        draw it on the circle
```

6.5 Other uses



Figure 6.15: 3D scan in camera live view

In addition to our main example described in this Chapter, we have considered what other possible uses our device could have. A somewhat related case we tried was using the LIDAR to create a small low resolution 3D scan of an item in front

of it. In Figure 6.15 such a scan of a person can be seen. Although a basic image, the contours of a human are clearly distinguishable.

6.5.1 3D scanner

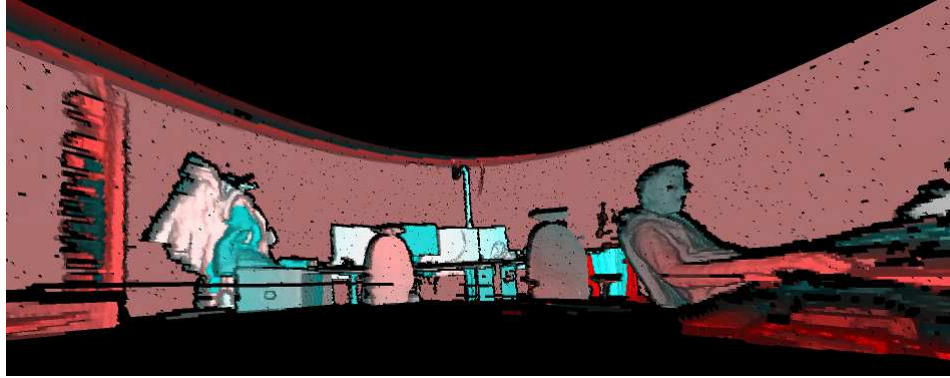


Figure 6.16: 3D scan

We investigated other uses for the platform created, and realised that it could be used as a 3D scanner of sorts. By sweeping the LIDAR in a spiral it covers a multitude of directions, while distance and motor positioning data is collected. This data is sent to a PC over RS232. The pan- and tilt-angles in conjunction with the distances are essentially spherical coordinates. They can be converted into Cartesian coordinates (x, y, z) by the formulas seen below, where r is the measured distance, θ is the tilt angle and ϕ is the pan angle.

$$x = r \cos \theta \cos \phi$$

$$y = r \sin \theta$$

$$z = r \cos \theta \sin \phi$$

These coordinates are then used to render surfaces using OpenGL in a small C-program. The surfaces are created by connecting measurements points which are adjacent in terms of pan and tilt angle to form rectangles, filtering and removing those that span a too large distance gap. The result can be seen in Figure 6.16.

A few noteworthy motivations regarding various parts of the project are mentioned in this Chapter.

7.1 Alternative image output solutions

Before deciding on the framebuffer-backed solution, alternatives were considered. One of these would be to continuously stream data to the pixel output module from main system memory. This would most likely be DMA-driven. Another alternative would have been to create specific HDL modules for shape generation (circles, text etc), these would have the advantage of being highly portable, but cumbersome to create and not very flexible. The framebuffer-backed solution was primarily chosen due to the fact that the image data is always available to be transmitted to the sensor interface, regardless of when the CPU chooses to update the image. This decouples the sensor interface timing requirements from the image updates generated by the CPU. The main drawback is that it uses a large amount of on-chip RAM, and can thus not easily be ported to less capable FPGAs (which usually have a more limited availability of on-chip RAM).

7.2 Acceleration curve considerations

At first, a uniform step distribution over the sample points of the generated curve was attempted. Thus, if the sampled curve had 10 different points, and 3000 steps were to be taken, 300 steps per point had to be taken. This proved lacking, since it is desirable to accelerate quickly, without losing steps. Running the motors at low speed without acceleration will not lead to step loss in itself, so it was decided that the linear map alternative would be more fitting.

The main reason for having quick stepper motor manoeuvrability is for scanning items of interest. One possible example is described in 6.5. Instead of scanning a square area and painting a picture in the live view, it could be interesting to see if the platform could be used to (for instance) detect people using intelligent scan patterns.

8.1 Sensor emulation

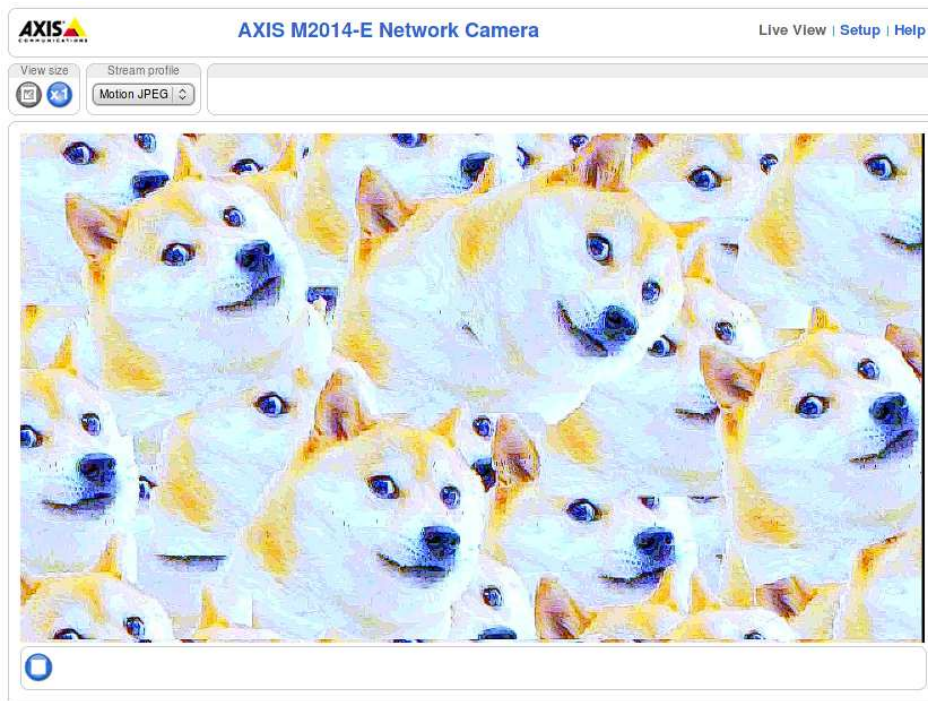


Figure 8.1: Camera live view showing test image of dogs

The primary goal of this thesis was to investigate the possibility of emulating an image sensor to provide alternative inputs. This has been shown to be possible, by implementing such a system. Using an unmodified main unit, alternative image data is accepted and forwarded through the network camera system, with the final result of it being viewable in the browser live-view. An example of this can be seen in Figure 8.1. The system is capable of exchanging the entire content of the frame

in 11 ms, with 12 bits of color depth, giving ample time for any other computations required by the specific sensor emulated.

Some attempts at addressing the individual pixels in the Bayer data were made, but this created significant demosaicing effects, and was thus avoided.

Future improvements include taking into consideration white balance and similar settings, as well as using the exposure time information sent by the main unit via I2C in some way. Some of these improvements require changing the software in the main unit, which was deemed to be outside of the scope of this thesis.

While the sensor example implemented is quite complex, a simpler sensor could be implemented using a less capable FPGA and other hardware.

8.2 Sensor example

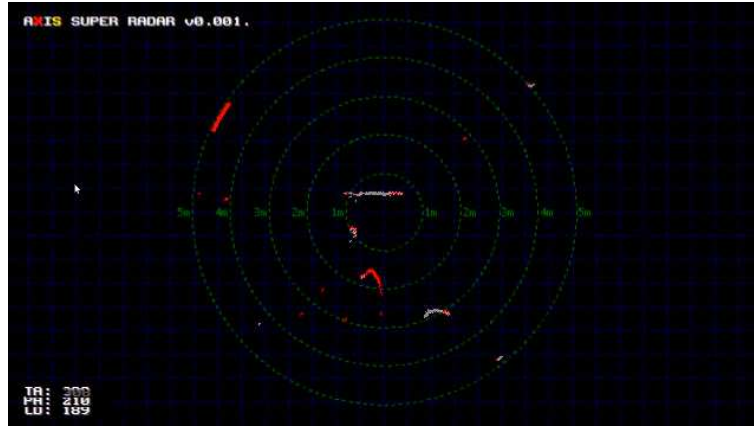


Figure 8.2: Camera live view showing radar image

The chosen sensor example, a LIDAR, has been successfully implemented. It fulfills its objective, providing a radar image of its surroundings as can be seen in Figure 8.2. It can achieve a rotational speed of 250 deg/s, which combined with the distance sensor output rate of 500Hz results in an angular resolution of 0.5 deg. At a distance of 5m this corresponds to 4.4 cm between measurement points.

By further making use of the pan tilt capacity, it can also be used for 3D-scanning objects. A simple 3D scan can be run on the sensor itself, with a result seen in Section 6.5.

Further development of this sensor include using the distance data to perform motion detection and perimeter defence.

This example shows that the current FPGA platform is powerful enough for more complex sensor types, requiring control of external components and non-trivial computations.

References

- [1] “Nexys 3 Board Reference Manual.” http://www.digilentinc.com/Data/Products/NEXYS3/Nexys3_rm.pdf.
- [2] “IBM VGA Technical Reference Manual.” http://www.mcarnafia.de/pdf/ibm_vgaxga_trm2.pdf.
- [3] R. Ramanath, W. E. Snyder, G. L. Bilbro, and W. A. S. Iii, “Demosaicking methods for bayer color arrays,” *Journal of Electronic Imaging*, vol. 11, pp. 306–315, 2002.
- [4] A. Athavale and C. Christensen, *High-Speed Serial I/O Made Simple, A Designer’s Guide, with FPGA Applications*. Xilinx, 2005.
- [5] J.-M. Irazabal and S. Blozis, *I2C Manual, AN10216-01*. Philips Semiconductors, 2003.
- [6] B. Cuzeau, *A.L.S.E. Application Note, RS232 Basics*. A.L.S.E., 2009.
- [7] “DE2-115 Manual.” Terasic.
- [8] “Embedded Peripherals IP User Guide.” Altera.
- [9] “Nios II/f Core: Fast for Performance-Critical Applications.” <http://www.altera.com/devices/processor/nios2/cores/fast/ni2-fast-core.html>. Retrieved 24/1-2014.
- [10] “DS92LX1621/DS92LX1622 10-50 MHz DC-Balanced Channel Link III Serializer and Deserializer with Bi-Directional Control Channel.” Texas Instruments.
- [11] “A1101, A1102, A1103, A1104, and A1106 Continuous-Time Switch Family.” Allegro.
- [12] “How to prevent step losses with Stepper Motors?.” Micromo Micro Motion Solutions.
- [13] “ADS7813p Low-Power, Serial 16-Bit Sampling Analog-To-Digital Converter.” Texas Instruments.



LUND
UNIVERSITY

<http://www.eit.lth.se>