




Embedded systems (2881) - Networks and RF (2885)

# Displacement tracking with ADNS3080 camera sensor on Raspberry Pi

*Students :*
Ling ZHANG
Thomas VANDENABEELE

*Teachers :*
Prof. Dr. Ir. Luc CLAESEN
Drs. Ing. Wout SWINKELS

December 20, 2016

**Acknowledgements**

We would like to thank Professor Luc Claesen for his expert advice and encouragement throughout this project. He has provided us so many necessary suggestions. We would like to thank teacher assistant Wout Swinkels as well for his brilliance in the lab. Thanks to his responsibility and inspiration, he has taught us attractive and useful lessons.

Also a noteworthy special thanks of gratitude to the ASEM-DUO Fellowship Programme, who provided us the golden opportunity to do this wonderful project and collaboration together. The authors have cooperated with each other very well and have therefore learned about so many new things and technologies.

# Contents

# Introduction

This project focuses on tracking the displacement of a target. It utilizes an ADNS-3080 optical mouse sensor to obtain a video stream. Then the x- and y-displacement between two frames are calculated on an embedded system, a Raspberry Pi. In order to improve the accuracy, sub-pixels are added into account to the frames. Further, the obtained camera frames and displacements are streamed through the Internet via a network. After that the video stream and calculated results can be seen on a website and an iOS application. So, clients can make a connection to the Raspberry Pi via a browser or our developed iOS application.

## Fields of application

As a basic study, this project can be used in many aspects, like Simultaneous Localization and Mapping, Pedestrian Dead Reckoning and other applications about localization and positioning based on a camera system.

Simultaneous Localization and Mapping (SLAM) is the computational problem of constructing or updating a map of an unknown environment while simultaneously keeping track of a target's location within it. SLAM approaches are employed in self-driving cars, unmanned aerial vehicles, autonomous underwater vehicles, planetary rovers, ... SLAM will always use several different types of sensors, as well as optical sensors. Therefore, this project can be applied in SLAM systems.

In navigation, dead reckoning is the process of calculating one's current position by using a previously determined position, or fix, and advancing that position based upon known or estimated speeds over elapsed time and course. Pedestrian dead reckoning (PDR) can be used to provide other navigation methods in a similar way to automotive navigation, or to extend navigation into areas where other navigation systems are unavailable. PDR is fairly complicated, as it is not only important to minimize basic drift, but also to handle different carrying scenarios and movements, as well as hardware differences across phone models. Instead of using built-in magnetometer and accelerometers, the on-board camera system could be used. Therefore, the algorithms and other topics from this project could be implemented in dead reckoning systems.

# Chapter 1

# ADNS-3080 Optical Mouse Sensor

The key device in this project is the ADNS-3080 optical mouse sensor. It is a high performance addition to Avago Technologies' popular ADNS family of optical mouse sensors.

The ADNS-3080 is based on a new, faster architecture with improved navigation en is capable of sensing high speed mouse motion - up to 40 inches per second and acceleration up to 15g.

The sensor forms a complete, compact optical mouse tracking system and is packaged in a 20-pin staggered dual inline package (DIP). There are no moving parts, which means high reliability. In addition, precision optical alignment is not required, facilitating high volume assembly. The sensor is programmed via registers through a four-wire serial port.

In this chapter the working method of this sensor will be discussed [1].

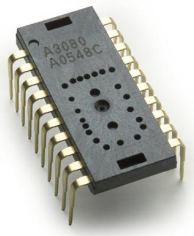



Figure 1.1: The ADNS-3080 Optical Sensor

## 1.1 Theory of Operation

The ADNS-3080 is based on Optical Navigation Technology, which measures changes in position by optically acquiring sequential surface images (frames) and mathematically determining the direction and magnitude of movement. It contains an Image Acquisition System (IAS), a Digital Signal Processor (DSP), and a four-wire serial port. The IAS acquires microscopic surface images via the lens and illumination system. These images are processed by the DSP to determine the direction and distance of motion. The DSP calculates the $\Delta x$ and $\Delta y$ relative displacement values.

Normally, an external microcontroller would read the $\Delta x$ and $\Delta y$ information from the sensor's serial port. For our project we did not take advantage of the on-board IAS and DSP. We readout the full frame via the four-wire serial port and calculates the $\Delta x$ and $\Delta y$ displacement values ourselves. This is neccesary because we do not have a fixed distance between the sensor and the filmed surface.

## 1.2 Synchronous Serial Port

The synchronous serial port can be used to set and read parameters in the ADNS-3080, and to read out the motion information.

The port is a four-wire, serial port, named the Serial Peripheral Interface (SPI) bus. SPI is used for short distance communication, primarily in embedded systems. The host microcontroller always initiates communication; the ADNS-3080 never initiates data transfers. The serial port cannot be activated while the chip is in power down mode (NPD low) or reset (RESET high). SCLK, MOSI, and NCS may be driven directly by a 3.3V output from a micro-controller. The port pins may be shared with other SPI slave devices. When the NCS pin is high, the inputs are ignored and the output is tri-stated. As shown in Figure 1.2, the lines which comprise the SPI port are:

- SCLK: Clock input. It is always generated by the master (the microcontroller).
- Input data (Master Out/Slave In).
- Output data (Master In/Slave Out).
- NCS/SS: Chip select input (active low).

The ADNS-3080's NCS port needs to be low to activate the serial port; otherwise, MISO will be high-Z, and MOSI  SCLK will be ignored. NCS can also be used to reset the serial port in case of an error.

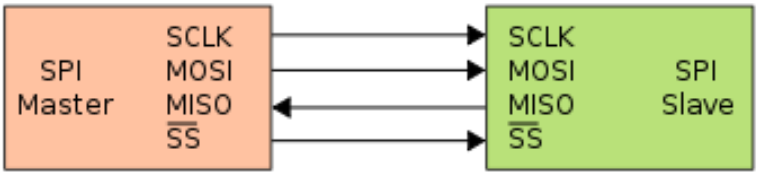



Figure 1.2: SPI connection diagram

Next subsections describes how we can interact with the ADNS-3080 sensor. Occasionally, we will also provide the C++ code we used to accomplish the kind of operation in the Raspberry Pi.

### 1.2.1 Write operation

Write operation, defined as data going from the micro-controller to the ADNS-3080, is always initiated by the micro-controller and consists of two bytes. The first byte contains the address (seven bits) and has a "1" as its MSB to indicate data direction. The second byte contains the data. The ADNS-3080 reads MOSI on rising edges of SCLK.

### 1.2.2 Read operation

A read operation, defined as data going from the ADNS-3080 to the micro-controller, is always initiated by the micro-controller and consists of two bytes. The first byte contains the address, is sent by the micro-controller over MOSI, and has a "0" as its MSB to indicate data direction. The second byte contains the data and is driven by the ADNS-3080 over MISO. The sensor outputs MISO bits on falling edges of SCLK and samples MOSI bits on every rising edge of SCLK.

The 250 ns minimum high state of SCLK is also the minimum MISO data hold time of the ADNS-3080. Since the falling edge of SCLK is actually the start of the next read or write command, the ADNS-3080 will hold the state of data on MISO until the falling edge of SCLK.

### 1.2.3 Motion Read

This feature has not been used in our project, but for the sake of completeness we mention this possibility. This mode is activated by reading the Motion_Burst register. The ADNS-3080 will respond with the contents of the Motion, Delta_X, Delta_Y, SQUAL,

Shutter_Upper, Shutter_Lower and Maximum_Pixel registers in that order. After sending the register address, the micro-controller must wait $t_{SRAD-MOT}$ and then begin reading data. All 56 data bits can be read with no delay between bytes by driving SCLK at the normal rate. The data are latched into the output buffer after the last address bit is received. After the burst transmission is complete, the micro-controller must raise the NCS line for at least $t_{BEXIT}$ to terminate burst mode. The serial port is not available for use until it is reset with NCS, even for a second burst transmission.

### 1.2.4 Frame Capture

This is the fastest way to obtain a full array of pixel values from a single frame. To trigger the frame capture, we need to write to the Frame_Capture register. The next available complete 1 2/3 frames (1536 values) will be stored to memory. The data is retrieved by reading the Pixel_Burst register once using the normal read method, after which the remaining bytes are clocked out by driving SCLK at the normal rate. The byte time must be at least $t_{LOAD}$. If the Pixel_Burst register is read before the data is ready, it will return all zeros. This operation can be seen in Figure 1.3.

To read a single frame, read a total of 900 bytes. The next 636 bytes will be approximately 2/3 of the next frame. The first pixel of the first frame (1st read) has bit 6 set to 1 as a start-of-frame marker. The first pixel of the second partial frame (901st read) will also have bit 6 set to 1. All other bytes have bit 6 set to zero. The MSB of all bytes is set to 1. If the Pixel_Burst register is read past the end of the data (1537 reads and on), the data returned will be zeros. After the download is complete, the micro-controller must raise the NCS line for at least $t_{BEXIT}$ to terminate burst mode. The read may be aborted at any time by raising NCS. Alternatively, the frame data can also be read one byte at a time from the Frame_Capture register.

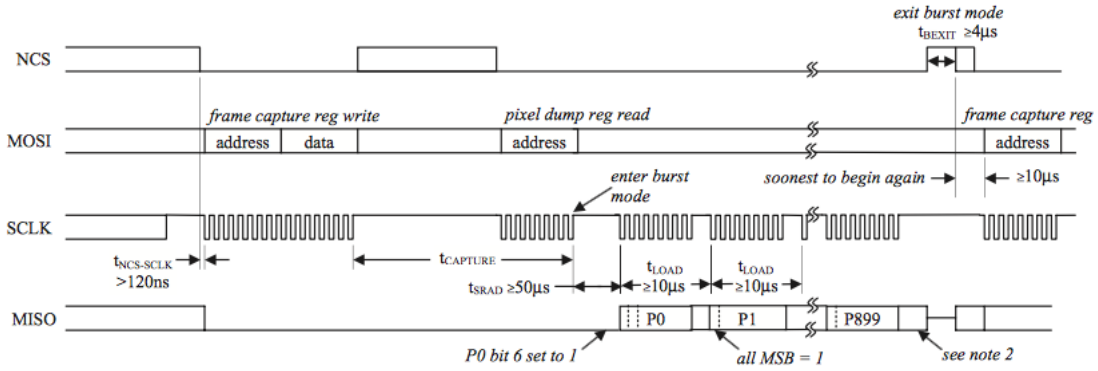



Figure 1.3: Frame capture burst mode timing

We implemented following code to use the frame capture function on the Raspberry Pi:

```
1  // pdata must point to an array of size ADNS3080_PIXELS_X x
       ADNS3080_PIXELS_Y
2  // you must call mousecam_reset() after this if you want to go back to
       normal operation
3  int mousecam_frame_capture(uint8_t *pdata)
4  {
5    mousecam_write_reg(ADNS3080_FRAME_CAPTURE,0x83);
6    bcm2835_gpio_write(PIN_MOUSECAM_CS, LOW);     // Select ADNS chip
7    bcm2835_spi_transfer(ADNS3080_PIXEL_BURST);
8    bcm2835_delayMicroseconds(50);
9
10   int pix;
11   uint8_t started = 0;
12   int count;
13   int timeout = 0;
14   int ret = 0;
15   for(count = 0; count < ADNS3080_PIXELS_X * ADNS3080_PIXELS_Y; )
16   {
17     pix = bcm2835_spi_transfer(0xff); // Get pixel value
18     bcm2835_delayMicroseconds(10);
19     if(started==0)
20     {
21       if(pix & 0x40)    // Check if it is first pixel of frame
22         started = 1;
23       else
24       {
25         timeout++;
26         if(timeout==100)
27         {
28           printf("A timeout occurred reading the frame.\n");
29           ret = -1;
30           break;
31         }
32       }
33     }
34     if(started==1)
35     {
36       pdata[count++] = (pix & 0x3f)<<2; // scale to normal grayscale
             uint8_t range
37     }
38   }
39
40   bcm2835_gpio_write(PIN_MOUSECAM_CS,HIGH);     // Deselect ADNS chip
41   bcm2835_delayMicroseconds(14);
42
43   return ret;
44 }
```

## 1.3 Optical Flow v1.0 Package

For our project we used the Optical Flow V1.0 from 3DRobotics.com, see figure 1.4. There is a 4.2mm 2.0 Mega Pixel lens mounted on top of the sensor. This makes it is also a lot easier to connect the device to the micro-controller.

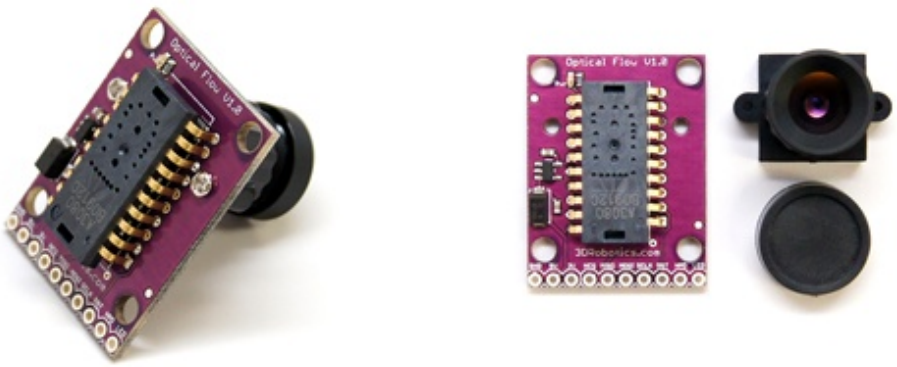


Figure 1.4: Optical Flow Sensor

# Chapter 2

# Embedded Systems

## 2.1 Arduino Processing

To understand the basic principles of the optical sensor, we first tried to connect the sensor to an Arduino before using the Raspberry Pi. The Arduino code reads from the ADNS3080 sensor and outputs the data to the terminal. It captures the motion vectors or dumps out the whole frames and sends it over a serial port to the connected computer.

In the arduino code we used the SPI library. It allows us to communicate with SPI devices, with the Arduino as the master device. Reading values from the ADNS-3080 sensor SPI-transfer, based on a simultaneous send and receive, is used.

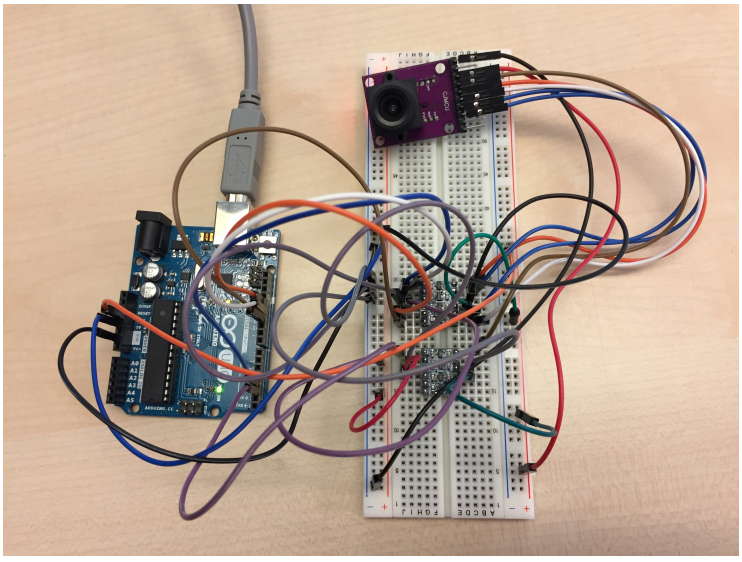


Figure 2.1: Arduino &ADNS-3080 Setup

### 2.1.1 Setup

The connection between the ADNS3080 and Arduino ports can be seen from table 2.1. Logic level shifters are used for providing the right voltage levels for the communication. A connection diagram is shown in figure 2.2.

| ADNS 3080 Port | Arduino Port | Color |
|---|---|---|
| CLK | 13 | yellow |
| MISO | 12 | green |
| MOSI | 11 | orange |
| NCS | 2 | brown |
| RST | 3 | purple |

Table 2.1: Connections between Arduino and Sensor

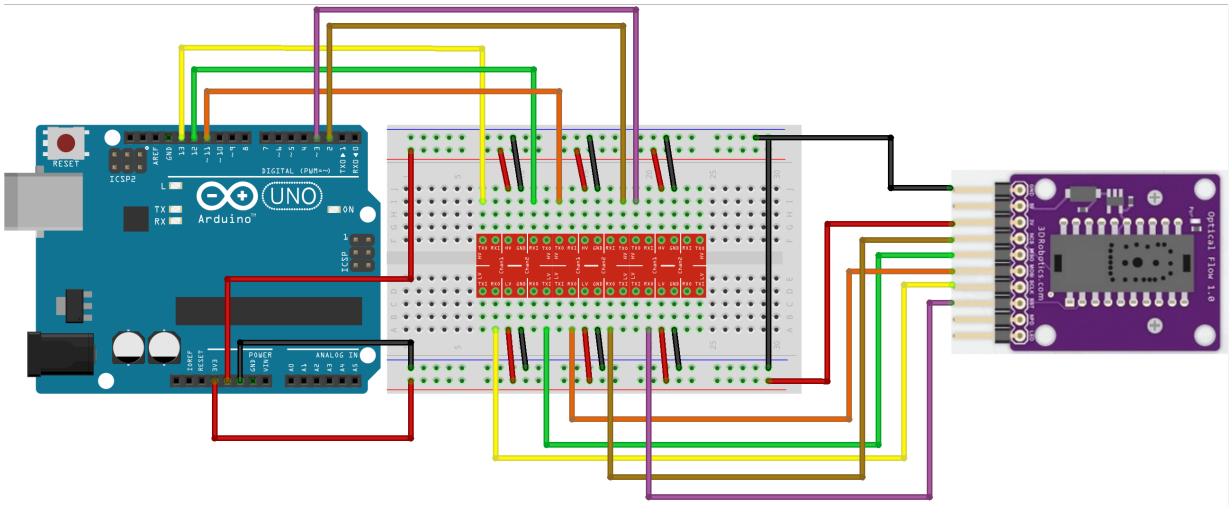



Figure 2.2: Connection scheme: Arduino Uno, Logic Level Converters ADNS 3080

### 2.1.2 Arduino code & theory

The camera motion is obtained by SPI.transfer(). As is shown in figure 2.3, the code-flow used to readout the frame capture is straightforward.

## 2.2 Matlab Integration

The frames are dumped out on the serial port by the Arduino as illustrated in the previous section and in Figure 2.4. In this section, the vertical and horizontal displacement is

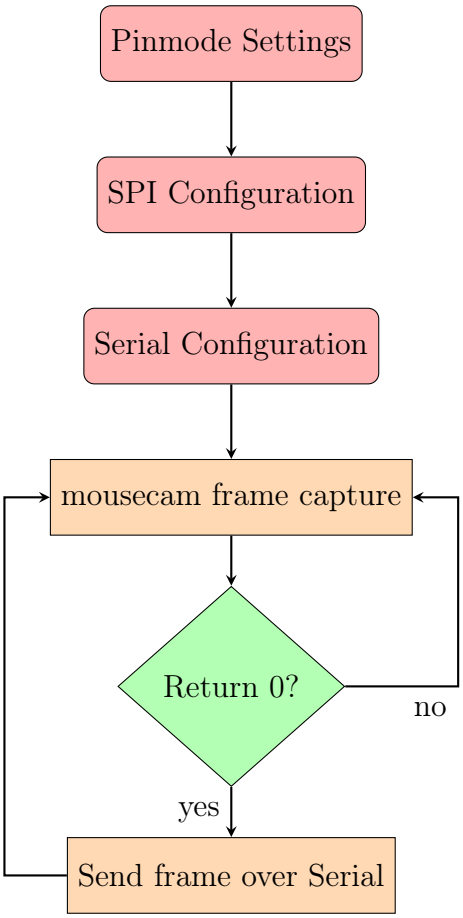



Figure 2.3: ADNS-3080 Arduino code - Flow Diagram

proceeded by MATLAB after reading out the frame capture from the serial port.

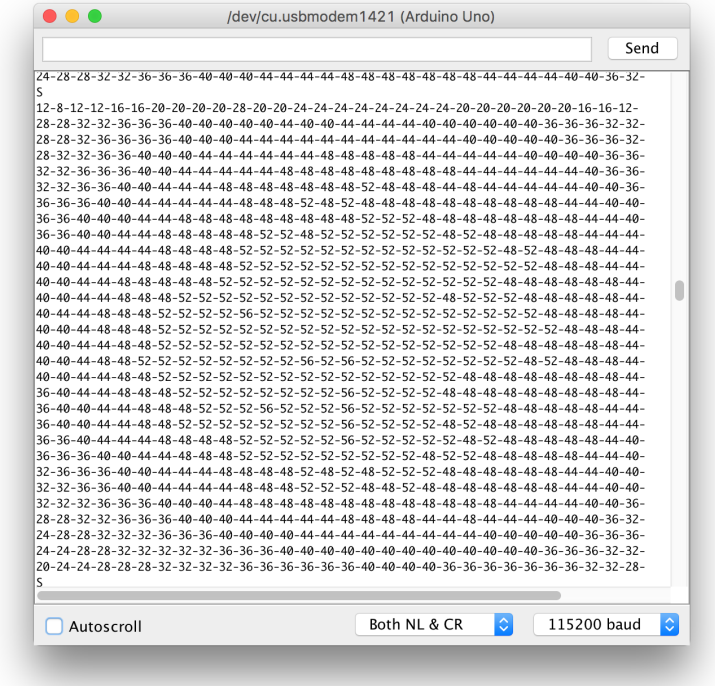



Figure 2.4: Arduino Serial Communication

### 2.2.1 Displacement

The offset of X and Y is calculated by cross-correlation. Every time when a new frame occurs, the correlation between new frame and the template window is calculated. A template window is chosen from the previous frame. The peak of correlation stands for the position of the template window in the new frame. Then the X-Y offset is obtained. The results can be seen in Fig. 2.5. A representation of the used algorithm is shown in the next section of this chapter.

### 2.2.2 Sub-pixel interpolation

The frame captured by the mouse camera is 30x30. To improve the accuracy, sub-pixel method is utilized.

The interpolated values on a refined grid are formed by repeatedly halving the intervals k times in each dimension. This results in $2^{k-1}$ interpolated points between sample values. For example $k = 2$, the following illustration Fig. 2.6 shows the placement of interpolated values (in red) among nine sample values (in black).

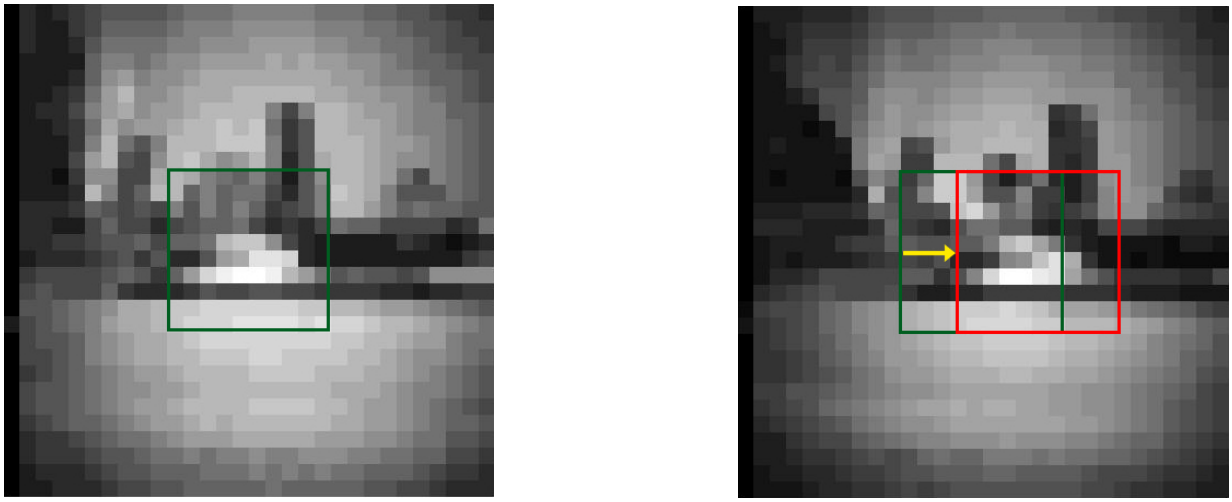


Figure 2.5: $\Delta x = +3$ & $\Delta y = 0$ offset calculated with cross-correlation

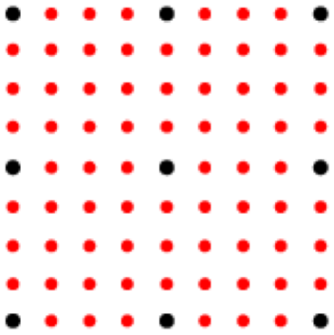


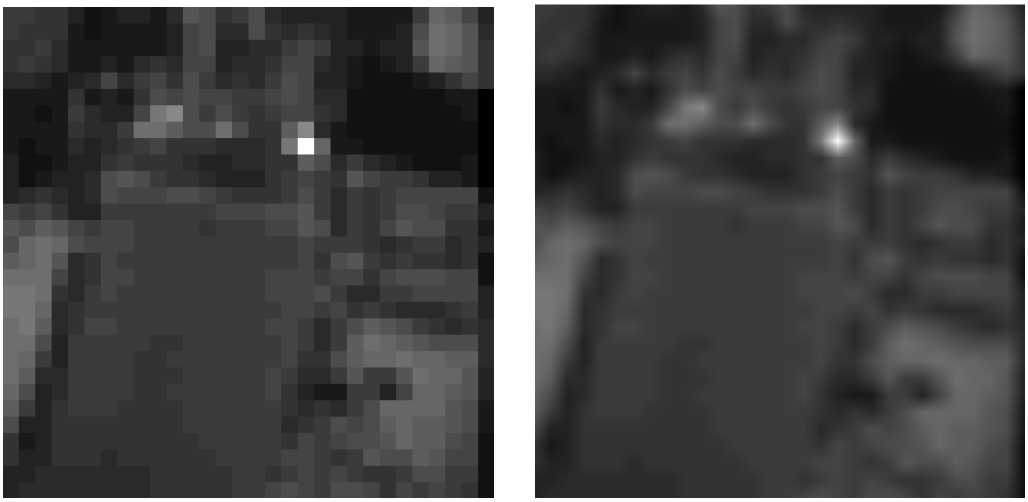
Figure 2.6: Correlation

The interpolated value at a query point is the value at the nearest sample grid point. Two grid points in each dimension are required. The advantage is that fastest computation with modest memory requirements.

In order to compare the images with and without sub-pixel, an image is taken from the mouse camera and is proceeded by MATLAB. The images with sub-pixel are shown in Fig. 2.7, where (a) represents the original image and (b) represents the target image when $k = 2$. So the size of (b) image is $117 * 117$. Apparently, more sub-pixels mean more details.

### 2.2.3 Cross-correlation

The $normxcorr2(template, A)$ function in MATLAB computes the normalized cross-correlation of the matrices template and A. It uses the following general procedure [2]:



(a) Original Frame (b) Interpolated frame ($k = 2$)

Figure 2.7: Sub-pixel Interpolation

1. Calculate cross-correlation in the spatial or the frequency domain, depending on size of images.
2. Calculate local sums by precomputing running sums.
3. Use local sums to normalize the cross-correlation to get correlation coefficients. The implementation closely follows following formula from:

$$\gamma(u,v) = \frac{\sum_{x,y}[f(x,y) - \bar{f}_{u,v}][t(x-u,y-v) - \bar{t}]}{\left(\sum_{x,y}[f(x,y) - \bar{f}_{u,v}]^2 \sum_{x,y}[t(x-u,y-v) - \bar{t}]^2\right)^{0.5}}$$

where

- $f$ is the image.
- $\bar{t}$ is the mean of the template.
- $\bar{f}_{u,v}$ is the mean of $f(x,y)$ in the region under the template.

### 2.2.4 Displacement with subpixel

The used algorithm for calculating the X and Y displacement of the frame stream from the camera sensor is shown in Algorithm 1.

Furthermore, the displacement is calculated again for sub-pixel images. To keep the run speed not that slow, we set k equals to 2 and choose template window as 70*70.

**Algorithm 1** XY displacement calculation from ADNS 3080 with matlab

```
1: procedure XYDisplacementCalculation
2:     loop:
3:     NewFrame ← Read frame from serial port
4:     NewFrame ← Rotate & flip frame
5:     NewFrameSubPix ← Use interp2 for calculate subpixels from NewFrame
6:     Template ← Get template-window from OldFrameSub
7:     Correlation ← Cross-correlation (normxcorr2) with NewFrameSubPix and Template
8:     XPeak, YPeak ← Find peak values in Correlation
9:     XOffset, YOffset ← Calculate offset values from XPeak  YPeak
10:    Draw frame on figure from NewFrameSubPix
11:    OldFrameSub ← Save NewFrameSubPix
12:    goto loop.
```

## 2.3 Raspberry Pi

Because we want our project to become an embedded system, we implemented it on a Raspberry Pi 2. The speed of the system gets much higher now because we do not need to send the frames over the serial port. The raspberry pi has ofcourse also a higher performance then an Arduino. We only need to readout the frame from the ADNS-3080, then we can calculate the displacements on the raspberry pi with a C++ program. Therefore, we converted our *.ino* arduino code into a native C++ program. On the Raspberry Pi we made use of the bcm2835 library for controlling the SPI and other GPIO pins. For implementing the image processing from Matlab we also installed OpenCV on the device. For the Matlab's $normxcorr2(template, A)$ template matching we used OpenCV's $minMaxLoc$ function. We implemented a variant on this function from Bill Arden to provide SubPixel Accuracy with $minMacLocSubPix$ [3]. Then we can calculate the x and y displacements the same way as in Matlab.

Instead of printing our results in the terminal, we provided a clean way of showing them trough the Internet. This is explained in the next chapter.

# Chapter 3

# Networks & RF

We embedded our ADNS 3080 project also in the 'Networks  RF' course. In this part we examined the best way to make a connection between our camera node and clients that want to see the image stream of the camera and the calculated displacements.

## 3.1 Bluetooth connection

Firstly, the possibility to make a bluetooth connection between a raspberry pi and any iOS device was investigated. Bluetooth is a standard wire-replacement communications protocol primarily designed for low-power consumption, with a short range based on low-cost transceiver microchips in each device. Because the devices use a radio (broadcast) communications system, they do not have to be in visual line of sight of each other.

Because we found a faster and cleaner way for wireless communication, we did not further implement the use of bluetooth.

## 3.2 WebSocket protocol

Another examined communication protocol is the use of the WebSocket protocol. WebSocket is a computer communications protocol, providing full-duplex communication channels over a single TCP connection. The technology is designed to be implemented in web browsers and web servers, but it can be used by any client or server application [4].

The WebSocket Protocol is an independent TCP-based protocol. Its only relationship to HTTP is that it's handshake is interpreted by HTTP servers as an Upgrade request. The WebSocket protocol makes more interaction between a browser and a web server possible, facilitating the real-time data transfer from and to the server. This is made possible by

providing a standardized way for the server to send content to the browser without being solicited by the client, and allowing for messages to be passed back and forth while keeping the connection open. In this way, a two-way (bi-directional) ongoing conversation can take place between a browser and the server.

## 3.3 Node.js & Socket.IO

To implement a webserver on the raspberry pi the Node.js package was used. Node.js is an open-source, cross-platform JavaScript runtime environment for developing a diverse variety of tools and applications. Node.js allows the creation of Web servers and networking tools using JavaScript and a collection of "modules" that handle various core functionality. Modules are provided for file system I/O, networking (DNS, HTTP, TCP, TLS/SSL, or UDP), binary data (buffers), cryptography functions, data streams and other core functions. Node.js's modules use an API designed to reduce the complexity of writing server applications.

Node.js is primarily similar to PHP. The biggest difference between Node.js and PHP is that most functions in PHP block until completion (commands execute only after previous commands have completed), while functions in Node.js are designed to be non-blocking (commands execute in parallel, and use callbacks to signal completion or failure) [5].

Socket.IO is a JavaScript library for realtime web applications. It enables realtime, bi-directional communication between web clients and servers. It has two parts: a client-side library that runs in the browser, and a server-side library for node.js. Both components have a nearly identical API. Like node.js, it is event-driven. Socket.IO primarily uses the WebSocket protocol with polling as a fallback option, while providing the same interface. Although it can be used as simply a wrapper for WebSocket, it provides many more features, including broadcasting to multiple sockets, storing data associated with each client, and asynchronous I/O [6].

Node.js and Socket.IO are both implemented in our project. The power of Node.js is that we can implement a webserver on the raspberry pi with just a few lines of javascript code:

```
1  var app = require('http').createServer(handler)
2  var io = require('socket.io')(app);
3  var fs = require('fs');
4  app.listen(5000);
5
6  function handler (req, res) {
7    fs.readFile(__dirname + '/public/index.html',
8    function (err, data) {
9      if (err) {
10       res.writeHead(500);
```

```
11          return res.end('Error loading index.html');
12        }
13        res.writeHead(200);
14        res.end(data);
15      });
16    }
17
18    io.on('connection', function (socket) {
19      console.log("client connected");
20
21      socket.on('frame', function (_xoffset, _yoffset, _pixels) {
22       socket.broadcast.emit('frameData', { xoffset: _xoffset, yoffset:
            _yoffset, pixels: _pixels });
23      });
24    });
```

Listing 3.1: Node.js webserver on port 5000

Socket.IO will wait until a frame event occurs. This frame event is alway initiated by the running C++ that is connected as a client to the webserver, as can be seen in Listing 3.2.

```
1  // Send image over websocket to server
2  message::list arguments(to_string(xOffset));
3  arguments.push(to_string(yOffset));
4  arguments.push(std::make_shared<std::string>(buffer,900));
5  h.socket()->emit("frame", arguments);
```

Listing 3.2: Send frame data from C++ program trough WebSocket

We developed a small HTML page where we implemented Socket.IO to receive the frame broadcast from the server. When the frame data is received we update a HTML canvas to show the video stream:

```
1  var socket = io();
2  var frameRate = 0;
3  socket.on('frameData', function (data) {
4    frameRate = frameRate + 1;
5    var frame = data.pixels;
6    var view = new Uint8Array(data.pixels);
7    document.getElementById('videostream').innerHTML = '';
8    for (var i = 0; i<900; i++) {
9      pixDraw(view[i]);
10   }
11   $("#xoffset").html(data.xoffset);
12   $("#yoffset").html(data.yoffset);
13 });
```

Listing 3.3: JavaScript from index.html

## 3.4 iOS Application

In order to show the result of the Raspberry Pi, an application is designed by Apple's Swift language to run on an iPhone 6. The flow diagram is illustrated in Fig. 3.1 and the main screenshots are shown in Fig.3.2.

The application connects to the server by IP address. As can be seen in Fig.3.2a, the IP address is typed in a text field and the default value is 192.168.0.112 (for easier development). With the correct IP address, the socket connects with the server.

Then the horizontal and vertical displacements and frame-pixels are transferred through the web socket. Fig.3.2b shows the video taken by the ADNS3080 camera.

Below the video, the movement path of the camera is drawn. Further, a clear button is utilized to reset the path-drawing.

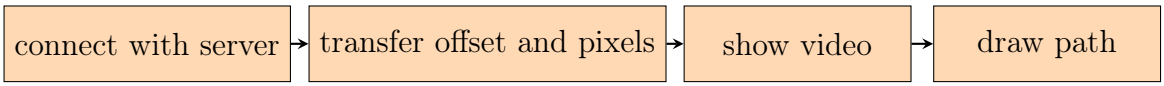
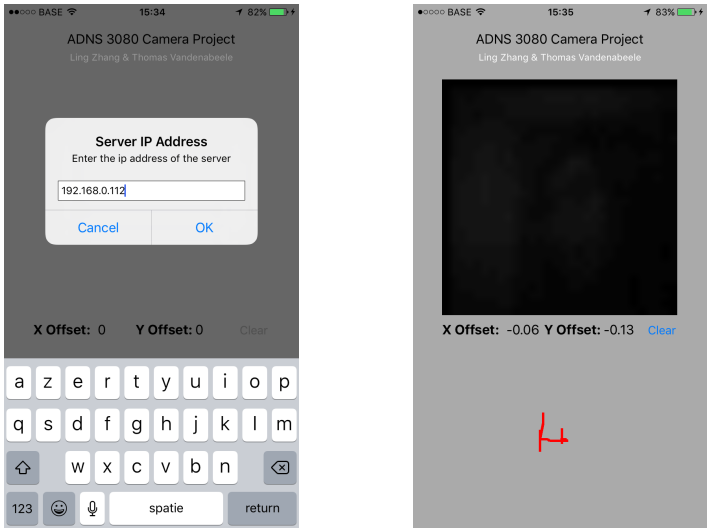



Figure 3.1: iOS Application flow diagram




(a) Connection Screen (b) Main Screen

Figure 3.2: Screenshots from iOS Application

# Conclusion

In this report a method for calculating the x- and y-displacement from a moving camera system has been investigated. The camera system makes use of the ADNS-3080 optical mouse sensor. The WebSocket protocol is a good way to send a large amount of data to connected clients. It is convenient to utilize a webserver on the Rasberry Pi. By using the OpenCV library we were able to calculate the x- and y-displacements from the frames. In order to improve the accuracy, we improved the calculation to take subpixel accuracy into account.

Thanks to this project the authors have learned many new important skills and have become acquainted with new technologies.

## Future Work

There are still some improvements that could be implemented in this project. We could implement more than one camera to calculate the displacement values in more directions. An proximity sensor can be added to the setup, to take the depth from the sensor to the captured objects into account.

The project could be used in visual SLAM (VSLAM) systems which uses primarily visual (camera) sensors to track localization. Due to the increasing ubiquity of smartphone cameras, there is a high probability that VSLAM could be native implemented on mobile devices in the future. Another hot topic, named PDR systems, can also use camera sensors to track pedestrian positioning. Therefore, the algorithms and other topics discussed in this project could be implemented in VLSAM and PDR systems.

# Bibliography

[1] https://people.ece.cornell.edu/land/courses/ece4760/FinalProjects/s2009/ncr6_wjw27/ncr6_wjw27/docs/adns_3080.pdf

[2] https://nl.mathworks.com/help/images/ref/normxcorr2.html

[3] http://answers.opencv.org/question/29665/getting-subpixel-with-matchtemplate/

[4] https://en.wikipedia.org/wiki/WebSocket

[5] https://nodejs.org/en/

[6] http://socket.io