ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

Titulación:

INGENIERÍA DE TELECOMUNICACIÓN.

Título del proyecto:

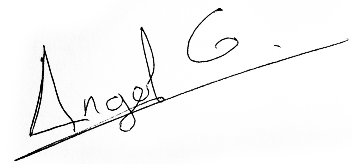
CHARACTERIZATION AND USE OF THE PAC7001 CAMERA WITH THE POSIT ALGORITHM ON AN AVR MICROCONTROLLER.

Ángel González García
Luis Javier Serrano Arriezu
Pamplona, septiembre 2013

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Declaration

I confirm that this paper is entirely my own work. All sources and quotations have been fully acknowledged in the appropriate places with adequate footnotes and citations. Quotations have been properly acknowledged and marked with appropriate punctuation. The works consulted are listed in the bibliography. This paper has not been submitted to another examination panel in the same or a similar form, and has not been published. I declare that the present paper is identical to the version uploaded.



Vienna, 29th of September 2013

Place, Date

Signature

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Resumen:

Este proyecto trata sobre la caracterización y uso de la camara PAC7001 para la implementación de un sistema de estimación de pose. Este sistema estará orientado al uso en tecnologías asistivas como un sistema de posicionamiento de la cabeza. La pose se calcula mediante el algoritmo POSIT, ejecutado en un microcontrolador AVR. Este algoritmo a su vez será optimizado para su uso en disposivos AVR de 8 bits.

En este proyecto las capacidades de la camara son analizadas, comprobando las diferentes configuraciones tanto de hardware como de software. Para probar las diferentes configuraciones se montará un prototipo del sistema, y se analizará la respuesta del mismo para las diferentes configuraciones.

Además del software del microcontrolador, se implementan diferentes programas auxiliares para analizar el sistema desde el PC , de forma que se facilite el desarrollo del mismo en un futuro.

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Kurzfassung

Diese These beschreibt die Charakterisierung und Verwendung des Kamerachips PAC7001 zur Positionsbestimmung im Raum mittels Infrarot Referenzsignalen.

Die Position wird per POSIT Algorithmus bestimmt, welcher auf einem AVR Microcontroller ausgeführt wird und für diesen optimiert wurde.

Es wird sowohl die Eignung der Kamera an sich analysiert, als auch ein einfaches Head-Tracking system aufgebaut welches als Maus-Ersatz benützt werden kann.

Schlagwörter: POSIT, Positionsbestimmung, PAC7001, AVR

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Abstract

This thesis focus in the characterization and use of the PAC7001 camera in order to implement pose estimation system. This system would be used in assistive technologies as a head tracking system. The pose is calculated with a POSIT algorithm running in an AVR microcontroller, the algorithm is improved to work on AVR 8-bits devices.

In this thesis the possibilities of this camera are analysed, by both hardware and software configuration. Therefore a initial head tracking system is implemented. Along with the microcontroller software, debug programs for the PC are implemented in order to help in future development.

Keywords: POSIT, Pose estimation, PAC7001, AVR

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Contents

1. Introduction **1**
1.1. Objetives . . . 1
1.2. State of the Art . . . 1
1.3. Methods, concepts and devices . . . 12

2. Development **20**
2.1. Hardware research and design . . . 20
2.2. Software design . . . 22
2.2.1. Software overview . . . 24
2.2.2. Data format and structures . . . 25
2.2.3. Actual block diagram . . . 28
2.3. System operating range . . . 36
2.4. Tests . . . 38
2.5. Support programs . . . 51
2.6. Future improvements . . . 55
2.7. Summary and conclusions . . . 56

Bibliography **58**

List of Figures **62**

List of Tables **63**

List of Abbreviations **64**

A. Provided Files **65**

B. Hardware schematic **66**

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

1. Introduction

1.1. Objetives

The focus of this project is the characterization and analysis of the PAC7001 camera, in order to implement a head tracking system, using the POSIT algorithm in an AVR microcontroller.

The first goal is to reverse engineer the PAC7001 camera. This camera came, at first, with a very simple and incomplete datasheet, so the understanding of the camera, and its registers, is the first step in the project.

The second step is to implement the POSIT algorithm in an 8-bits AVR microcontroller. One of the challenges when implementing the algorithm in this microcontroller, is the lack of floating point operations, which are necessary in the POSIT algorithm. This floating point operations are programmed in the AVR, which makes them slow.

Finally with the camera and the POSIT algorithm implemented, the goal is to check if it is possible, with these devices , the development of a head tracking system.

1.2. State of the Art

Nowadays, there are several devices, adapted to be used by disable people, which make them less dependant. Some of the devices improves mobility (i.e: wheel chairs or walkers), others replace parts of the body, the prosthesis, or help with different disabilities. Due to the wide range of existing systems, it is impossible to explain all of them, so we are focusing on the interface human-computer, where a computer can be a PC, a laptop or a smarthphone.

The interfaces human-computer can be unidirectional or bidirectional. The interest in this Master Thesis is about unidirectional systems, more specifically the input devices adapted to disabled people, so a enumeration and description of them is made in the next paragraphs. The systems that we are talking about can be grouped in:

- Physical Switches
- Mouse interfaces
- Image recognition
- Voice Input

Physical Switches

The simplest of this kind consists in a single push button, which has been enlarged so it is easier to hit. These devices need less force to be activated and deal better with bouncing. These push buttons are limited by the fact that they can only have two states, activated or deactivated, so the usefulness is limited. That is why these kind of inputs are more used by people with severe disabilities, as they can not use a more complex system. To increase the utility, arrays of buttons

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

are used. They are usually arrays of special push buttons (the ones just commented) with bold colors to be easily distinguishable. We can find a lot of examples of this kind of buttons:

- Gumball Switches[1]
- Big Buddy[2]
- Switch it up [3]
- It switch [4]

Figure 1.1.: Example of big push buttons.[1]

These arrays of buttons can be designed for a specialized purpose or to work as a keyboard. To keep all the keys in the keyboard the quantity of needed buttons make impossible to make them big, so other solutions are followed. The keyboard layout is changed to be used by only one hand or with low mobility in the fingers. The main problem is that the keys keep the same size or even smaller. Some examples can be found:

- Keyboard BigKeys ABC layout[5]: In this product, the size of the keys has been increased, improving visibility and accuracy while typing.
- Contoured Advantage[6]: This is an example of how the position of the keys are changed to adapt to low mobility and to be more ergonomic.
- BAT Keyboard[7]: The last one is a single hand controlled keyboard. The different key combinations give us all the possible characters of a keyboard.

[1] http://enablingdevices.com/catalog/capability_switches/best-sellers/gumball-switches-accessories
[2] http://www.spectronicsinoz.com/product/big-buddy
[3] http://www.adaptivetechsolutions.com/pd-switch-it-up-switch.cfm
[4] http://www.infogrip.com/products/it-switch.html
[5] http://www.iltsource.com/BigKeys_Plus_Color_ABC_p/bigkeysmp6.htm
[6] http://www.infogrip.com/products/keyboards/contoured-advantage.html
[7] http://www.infogrip.com/products/keyboards/bat-keyboard.html

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

Figure 1.2.: Keyboards adapted to different diseases[2][3]

In some cases more sophisticated systems are needed. The sip-and-puff systems, are switches in which each action, sip and puff, has a different function. The function depends on the application and it is specifically design for each case. However, the research in this devices makes it possible to increase the number of functions, with the distinction between soft and hard sip and puff. These switches are used when severe mobility diseases are present, since no movement is necessary to activate it. They are activated with the mouth, so they can be mounted on the head, on a table or on a wheel chair. From this devices we can find:

- SB-4H Sip and Puff switch with hardware [8]
- Sip and Puff with headset[9]

Figure 1.3.: Two implementations of the sip and puff systems.[4][5]

Mouse interfaces

Simple machines can be used with a few buttons, however most of the devices we use, such as smartphones or PCs, require a pointer controller. The usual interfaces are mouse and digitizer, however these interfaces are not suitable for some disabilities. In this scenario different approaches have been followed.

In hardware the simplest way to implement a mouse interface is by a joystick, the move of the joystick in each axis moves the pointer in that axis. The adapted joystick usually consists in one or two switches and a bigger than usual handle. Depending on the disability the joystick is going

[8] http://www.zygo-usa.com/usa/index.php?page=shop.product_details&flypage=vmj_naru.tpl&product_id=314&category_id=139&option=com_virtuemart&Itemid=11

[9] http://www.spectronicsinoz.com/catalogue/sip-and-puff-switch-solutions-by-origin-instruments

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

to cover, it may change its behaviour: resistance to movement, total displacement of the handle or available axes. We can see some examples of this kind of mice in figure 1.4, and a list of commercialized adapted mice:

- Funkey Joystick[10]
- OPTIMAX Wireless Joystick[11]

Figure 1.4.: Different implementations of joysticks, working as mice.[6][3]

When the disability prevents moving the joystick, the pointer can be moved using buttons (figure 1.6) or trackballs. The layout of the buttons in the first solution varies among products. In some of them there are only four buttons, two for each axis, but more complex systems can include combinations of the basic axis, implementing i.e.: diagonal movement in some keys. The trackball alternative consists of a ball in which the pointer moves according to the amount and direction of the rotation. The advantages of the trackball over the buttons and joysticks is the freedom of movement of the pointer. However, it is more difficult to use than the others.

- Mouse button box[12]
- Footime Foot Mouse with Programmable Pedal[13]

Figure 1.5.: Trackball systems with different buttons layout.[7]

[10]http://www.keytools.co.uk/funkey-joystick.html
[11]http://www.infogrip.com/products/mice/optimax-wireless-joystick.html
[12]http://www.enablemart.com/mouse-button-box
[13]http://www.enablemart.com/footime-foot-mouse

upna
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

Figure 1.6.: Mouse buttons system, activated with the feet.[7]

In the case of not so severe disabilities special mice can be used. These mice may be bigger, have less buttons or different sensibility than ordinary mice. Sometimes the mouse has the buttons moved apart and they have to be activated by the other hand, this is used when the disability prevents movement of fingers.

In some cases touch screens can also be used. These interfaces do not require force to be applied but work only with skin contact. The size, resolution and behaviour to touches vary among different models and uses, increasing the utility of these interfaces. They can be used as switch buttons, mouse or even both at the same time. The implementation of movement patterns increases the versatility of this input systems. They can be controlled by the fingers, the whole hand or with a stick. And they can be prepared to avoid unintentional pushes. Some examples of track pads are:

- Orbitrack[14]
- Track-IT[15]

Figure 1.7.: Two trackpad system.[8][6]

[14]http://www.pretorianuk.com/orbitrack
[15]http://www.keytools.co.uk/trackit.html

upna
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Image Recognition

When the previous systems are not enough, image recognition can be used, which consists in the recognition of patterns or shapes (i.e.: Head, eyes and body) in images recorded by a camera. The method used by the cameras varies among systems: some cameras work with visible light (regular cameras), on the other hand the IR cameras, which usually have IR light sources, see the rebound of the IR light. The complexity of this method lies in the recognition algorithm. In all three cases (Head, eyes and body), the shape, size and layout vary among individuals, which makes the recognition of all possible shapes, sizes and layouts very complex. Since few years ago this kind of input can be seen in commercial products, the most important is the *Kinect for Xbox*[16] but due to its complexity it will be reviewed later.

One of the first image recognition system researched tracked the gaze. It was one of the first implemented methods because of the low variance in the algorithm for different eyes, that is, the algorithm has to distinguish a circle (the pupil) and the cornea, which are almost the same for every eye. To do this, the camera has to record images from the eye. In these images the software recognizes the pupil and the rest of the eye. With the position of the pupil relative to the rest of the eye, the software is capable of calculating the gaze orientation and translate it in a movement of the pointer on the computer. To make it easier, the camera which detects the eye, is usually head mounted and positioned just over an eye, so even when the head is moved, the eye keeps in the same position relative to the camera. The eye tracking can be used to move the pointer in a screen or to use simple machines. One of its disadvantages is the losing of the view, of what one is doing, because the eyes are moved. The utility of these systems are scenarios in which the important event takes place where you are looking at (like moving the pointer, the important thing is where the pointer is). In addition, implementation of patterns is hard because it is not easy to do movements with the eyes for a long time.

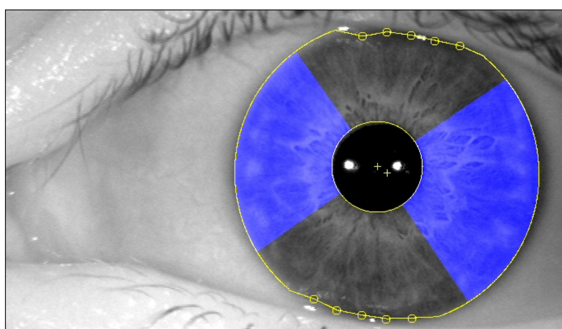


Figure 1.8.: Eye tracking, pupil's features[9]

As the eye tracking was the first developed system, there are eye tracking systems for different purposes, from psychology studies to marketing and human-computer interaction research. In figure 1.9 the *Tobii X2-69 Eye Tracker* [17] can be seen, which is a small implementation of an eye tracking system, this device can record the position of the eyes so they can be later analyzed, for example, to study how people look at the websites or pictures. This device has the added complexity that the camera is not near the eyes so the algorithm has to recognize the face and the eyes, and then calculate the position and orientation of them. Another eye tracking system is

[16] http://www.xbox.com/es-ES/Kinect/GetStarted

[17] http://www.tobii.com/en/eye-tracking-research/global/products/hardware/tobii-x2-60-eye-tracker/

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

the *Eye tracker type L*[18]. This one is mounted near the eye so the implementation is easier than in the first one. Although two examples are given, many others exist, but there are not so many differences among them. Other examples that can be found are:

- S2 Eye Tracker[19]
- EMR-9[20]
- RED250[21]

Figure 1.9.: Examples of eye tracking system already developed.[10][11]
[11]

Due to the loss of vision while working and the impossibility to do complex moves with the eyes, head tracking was researched. Head tracking works in the same way as eye tracking, but in this case the position and orientation of the head are tracked. The procedure is also the same: the camera records the images and then the software has to recognize the head to calculate its position and orientation. With the eye tracking, only the orientation of the eyes can be used as an input parameter. However, with the head tracking the position and the orientation can be used, which gives more possible inputs. This system has its advantages, but this requires a more complex algorithm for detection. The face of different people varies more than their eyes, and the head has to be distinguishable from the environment which also varies, from dark rooms to the most illuminated office. In some cases, the background of the image may be the same tone as the face which also complicates the head recognition. In most of the cases the recognition is not for the head but for the face. Once the position and orientation of the face is calculated, the same values are for the head. We can see in figure 1.10 how the algorithm detects the face and calculate the position and orientation of it.

The head/face tracking products that are commercialized are harder to find, this is because they are more complex than eye tracking. The *FaceLAB^{TM}5*[22] is a device capable of doing eye and head tracking. It can track the head movement in all the axis and rotation in the x and y axes. This device not only can acts as an input for the pointer, but also as a study device. It can create heat maps (example of heat map of a website figure 1.12) so one can study where people spend more time starring.

[18] http://en.pertech.fr/eye-tracker-l.html
[19] http://mirametrix.com/products/eye-tracker/
[20] http://www.nacinc.com/products/Eye-Tracking-Products/EMR-9/
[21] http://www.smivision.com/en/gaze-and-eye-tracking-systems/products/red-red250-red-500.html
[22] http://www.seeingmachines.com/product/facelab/specifications/

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

Figure 1.10.: Example of head tracking methods and results.

Figure 1.11.: *FaceLAB*^{TM}*5*[12]

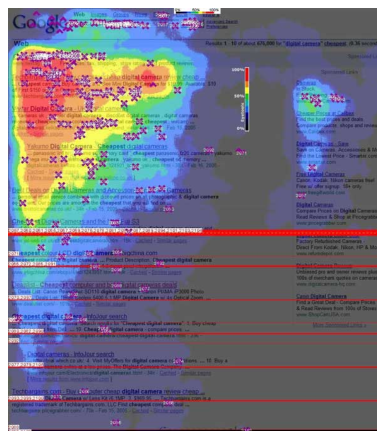


Figure 1.12.: Heat map generated by a eye tracking system, which colour the webpage indicating where the user stay more time looking at.[13]

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

The last step in image recognition is the body recognition. Although, this kind of recognition is not very useful in people with disabilities, it is worth mentioning. This is the most complex system of image recognition since it has to distinguish between the body and the environment. The main problem is that the body may not be looking forward to the camera, but turned so the camera can not see an arm or leg. The different types of bodies and clothes makes the task more difficult. The most successful device capable of body tracking is the already mentioned *Kinect for Xbox*. In figure 1.13 can be seen that the device has[14]: color and depth-sense lenses ,voice microphone array and a motor for tilt adjustment of the sensor. According to [14] it can track up to 6 people and 20 joints per active user. The device is also capable of speech, face and gesture recognition.

Figure 1.13.: Kinect system for Xbox.[15]

Speech recognition

The speech recognition (SR) consists in the translation of spoken words into text or orders that can be understood by a computer or a device. Some SR systems are trained, requiring the user to read a text. Then, the SR learns to recognize the voice. Other systems are speaker independent, meaning training is not necessary. If the SR needs to be trained it should be used only by one person, which makes the implementation easier and the recognition more accurate. The speaker independent, on the other hand, is harder to implement and the recognition less accurate. For disabled people the speaker dependent SR is more suitable, as it is used only by one person.

The SR does not require physical movement of the user, so can be used with severe movement impairment. This fact makes it useful in various situations: it can be used when severe mobility disabilities are present, or to increase the comfort when using devices. The main disadvantage of this kind of input systems appears when moving the pointer. When specific orders are necessary (such as open, close, move,:.) the SR systems are useful. However, moving the pointer can be hard.

There are different options in SR: there are SR engines, so developers can use them to do their own programs, fully functional programs, which the user only have to install, and integrated solutions, in which the SR is integrated in the operating system. The first group, the engines, is suitable for specific situations in which is hard to find software that fulfill the requirements. These scenarios, which require special conditions, need to be implemented by SR engines. We can find some example of these engines:

- CMU Sphinx[23]: This is an open source toolkit for speech recognition.

[23] http://cmusphinx.sourceforge.net/

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

- Julius[24]: It can be used under BSD license but need of citation.
- RWTH ASR[25]: Proprietary speech recognition toolkit.
- Zanzibar OpenIVR[26]: It uses the CMU Sphinx ASR engine and can also work with TTS(Text to speech)

The general purpose programs are already programmed and usually cover the basics functionalities. Due to the existence of different operating systems, there are programs for the most popular ones. Some examples are:

- LumenVox[27]: for Linux and Windows, it can understand 6 languages.
- Dragon Naturally Speaking[28]: Made by Nuance, is the Windows version.
- Dragon Dictate[29]: The same company of Dragon Naturally Speaking made this one for MAC users
- Dragon Dictation[30]: This is the iPhone and iPad version from Nuance software.
- Vocapia[31]: Is a completely functional software for Unix systems.

Even these fully operational programs are used, most of the main operating systems have their own SR system integrated in them. Usually the integrated SR only covers the basics requirements for elementary use. Nowadays Windows, MAC OS and android implement a speech recognition software, so the use of more complex SR system is only in cases with special requirements.

IR Tracking

The IR tracking systems usually consist of an IR light source, an IR camera and in some cases reflective stickers. The camera registers the IR light from the light sources, or the stickers, and calculates the position and the orientation of the object to be tracked. The layout of the devices varies between implementations, in some cases the lights or the stickers are in the tracked object. In other cases, the camera is mounted on the tracked object.

The implementation of the lights on the tracked objects is used in many scenarios. In the audiovisual industry, for complex special effects, the body of the actor is covered by IR light sources and, with several cameras, the body is tracked and can be easily modified in post-production, i.e.: modifying the shape or the texture of the skin. In high-level sport training, it is usually used to track the movement of the sportsman, so it can be analyzed in slow motion and therefore improved. For 3D face model animation these kinds of systems are used, a human model has reflective stickers on his face. These stickers are tracked and then passed to the 3D face model so it moves as the human model. The scenarios use a complex layout of lights and cameras so they are not suitable for home implementation. There are simpler layouts suitable for use at homes, which consist in a camera and a few small light sources. Some implementations are:

[24] http://julius.sourceforge.jp/en_index.php?q=en/index.html
[25] http://www-i6.informatik.rwth-aachen.de/rwth-asr/
[26] http://www.spokentech.org/index.html
[27] http://www.lumenvox.com/products/speech_engine/
[28] http://www.nuance.com/dragon/index.htm
[29] http://www.nuance.com/dragon/whats-new-dragon-dictate/index.htm
[30] http://www.nuance.com/for-business/by-product/dragon-dictation-iphone/
[31] http://www.vocapia.com/

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

- TrackIR 5[32]: this is a commercial head tracking which works with IR lights. It consists of an IR camera, and a set of lights which have to be mounted on the head (they can be attached to a hat, or to a headset).
- Johnny Chung Lee[33] developed a head tracking system with the Wiimote camera and the LED bar mounted on the head, can track the position and modify the images on the screen according to the head position.

On the other hand, the other implementation mounts the camera on the head and the lights are fixed. The system knows the layout of the lights, and has to calculate the position of the camera based on what the camera detects. This system is used by the Wiimote; the element that is moved is the camera and the lights are always fixed in top of the screen. This project follows this implementation, the camera will be mounted on the head, while the lights will be on the screen.

Pose estimation method

With the image of the lights taken by the camera, it is necessary to estimate the pose, translation and rotation of the camera. In order to do that several algorithms exist, however only the most important ones are commented.

- CamPoseCalib (CPC)
- POSIT algorithm.
- Direct Linear Transform (DLT)

The CPC is the algorithm implemented in the BIAS library[34]. This algorithm is based in the non-linear least squares problem, in which it is necessary to minimize a function error:

$$\hat{\phi} = \arg\min_{\phi} \sum_{i=1}^{m} (r_i(\phi))^2$$

In Gauss-Newton method, the term $r_i(\phi)$ represents the first order derivative of the residual. The algorithm is used to estimate the rotation and translation of an object. It needs the initial pose and then it is able to calculate the new pose. The goal of this method is to find the new pose by looking for the minimum of the error function. This minimum is found by calculating the derivative of the residuals. Once the minimum is found, the new pose is taken as correct.

To use this method, the functions for the derivatives are directly set, that is, the steps that the algorithm takes are predefined and may not be correct for some situations. In some cases the Gauss-Newton method is interpolated with the gradient descend. This makes the algorithm more robust, since it can start far off the correct minimum and still find the pose.

The POSIT algorithm uses a scaled orthographic projection (SOP), which gives a wrong perspective projection. However, when the algorithm converges, the true perspective is calculated. The SOP approximation leads to a linear equation system for the rotation and translation. This method uses a scale value for the points, and it is updated in every iteration.

[32] http://www.naturalpoint.com/trackir/products/trackir5/

[33] http://johnnylee.net/projects/wii/

[34] http://www.mip.informatik.uni-kiel.de/tiki-index.php?page=BIAS

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

This algorithm iterates by calculating the value of the scale value for the different points, until the change of the values between two consecutive iterations converges to 0. When that occurs the algorithm converges, and the solution is the translation and rotation vectors obtained by the algorithm. The advantage of this algorithm is that it does not require a starting pose, as in CPC.

However, when the object is planar, the POSIT algorithm fails because different poses have the same orthographic projection. In that case the POSIT for coplanar points is used. This algorithm creates two poses for the object, two **I** and two **J**. At the end of the algorithm, it selects one of them according to the distance between the points and a threshold level.

The DLT estimates the parameters of the projection matrix. To estimate all 12 parameters in the matrix, at least 6 points are needed. This method is the simplest one, however it is much slower than the CPC or the POSIT.

In our project the POSIT algorithm will be used, since it is the fastest one and it does not need the first pose to calculate the next one. That means that the POSIT can calculate the pose from one image.

1.3. Methods, concepts and devices

In this section the devices and algorithms used in the project will be explained. For the hardware part the next devices can be found:

- PAC7001CS: This is an IR camera which also implements object tracking algorithm. This implies that the camera does the job of analysing the image and get the coordinates of the objects in the image.
- AT90USB1286: This is the microcontroller with AVR architecture, it works at 16 MHz at 5V.
- LED structure: A structure holding the regular IR LEDs.

In the software part, the algorithm used by our application to calculate the position of the camera relative to the lights, is called POSIT. The communication between the camera and the microcontroller is over UART and the microcontroller uses USB to send the pointer data to the computer or device.

PAC7001CS

The PAC7001CS is the IR camera used for the project. This camera has the IR sensor with the DSP processor implemented. Therefore, the coordinates and size, of the object that are tracked, are sent instead of the IR image that the camera receives. Some features of the camera are:

- **Electronic features:**
  - Tracks up to 4 object
  - The supply voltage range is from 3V to 5V, but the data must me 3.3V
  - The communication interface is UART at maximum of 1687500 baud
  - System clock of 27MHz (oscillator included in the package)
  - Maximum current consumption 60 mA
  - Frame rate from 10 to 200 fps.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

- **Camera features**
  - Focal length: 1.3mm
  - Pixel size: $11\mu m \times 11\mu m$
  - Sensor size: 0.3 cm = 1/10"

The camera has two operational mode: Initial mode and Operation mode. In initial mode the camera does not track objects or send data trough the UART, but wait for the master to send the registers configuration. In this mode the camera accepts four orders:

- **Check device** (0xEA): This command is used to check the connectivity and functionality of the camera. Once the camera receive 0xEA it answers with *"Tracking V01"* string. The first time the camera is powered on, this command must be sent in order to activate it. If the string is received, the connection is correct and the camera can start sending and receiving data.

- **Set register** (0x10): The set register command should be followed by two or three another bytes. The first byte after the 0x10 is the register number one wants to change and the following one or two are the new value for that register. The number of bytes depends on the register to be changed, some registers need only one byte while other ones require two. Once the register is changed the camera will answer with a 0x10 byte.

- **Read register** (0x11): This byte must be followed by the number of the register which on wants to be read.

- **Switch to Operation Mode** (0x8D): With this instruction the camera switches to Operation Mode.

While in Initial mode the register can be set, these registers change some parameters of the camera. There are several register which can be configured, however the most important ones for our project are explained.

Name of byte	Name	Function
0x00	Gain 1	Sensor analog gain
0x01	Gain 2	Sensor analog gain
0x02	Sensor update flag	It has to be changed to 1 to update the registers once are changed.
0x05	IW (2 bytes)	Image Width
0x06	IH (2 bytes)	Image Heigh
0x0F	Feature option enable flag	Which features of the tracked object wanted to be included in the data while in Operation mode.
0x11	Baud rate	Change the baud rate of the UART communication

Table 1.1.: Some register of the camera.

Once the camera has been configured by the master, it can be turned to Operation mode. In this mode the camera does not accept more configuration orders, it just send the object's features

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

set in the *Feature option enable flag* register. The structure of the data received during Operation mode can be seen in table 1.2. The data received is structured in Frames, every frame has the data of four objects. The frame starts with the frame header, then the features of the objects come four times.

The communication between the camera and the microcontroller is through the UART port. The baud rate of the communication is set in the register 0x11, as can be seen in table 1.1, the possible values for the baud rate go from 19200 Bd to around 17 MBd. The UART transmission must be configured to start with a start bit, followed by 8 bits of data, and the stop bit. The number of objects that the camera tracks, depends on which is the UART baud rate. With the default baud rate the camera only sends data of one object, while with 115200 Bd, the camera can send the data of up to four objects.

Byte name	Bit	Function
Header	0xFF 0x00 0xFF 0xFF	Frame header to distinguish between frames
Flag Byte	Bit 7: Always 1 Bits [5:6]: 01 when trace is OK Bits[0:4]: Frame number	Used to check if the object has been tracked or not.
Object Flag Byte	Bit 7: Always 0 Bits[4:6] : Reserved Bits[2:3]: 00 Circle 01 Bar 10 Circle-hole 11 Bar-hole Bit 1: Reserved Bit 0: 0 if finished	It is used to check which kind of object is the camera tracking.
X coordinate	First goes the High byte, then the Low byte	Gives the X coordinate of the center of the tracked object.
Y coordinate	First goes the High byte, then the Low byte	Gives de Y coordinate of the center of the tracked object.
Size	First goes the High byte, then the Low byte	Gives the size of the tracked object.
EOB	Bit 7: Always 0 Bits[4:6]: Object number Bits[0:3]: Checksum	Distinguish between objects

Table 1.2.: Structure of the data received from the camera.

AT90USB1286

The AT90USB1286 is a AVR 8-Bit Microcontroller, which is mounted in a USB based microcontroller development board called *Teensy 2++*[35]. The features of the AT90USB1286 that are more important for this project can be found in table 1.3. The maximum frequency depends on the supply voltage, for 5.5V is 16Mhz and for 2.7V is 8Mhz.

[35] `http://www.pjrc.com/store/teensypp.html`

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

The rest of the characteristics can be found in the datasheet, and they will be not commented due to its extension.

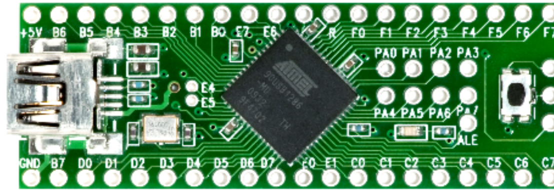


Figure 1.14.: Image of the Teensy2++[16]

Feature	Value
Supply voltage	2.7 - 5.5V
Maximum frequency	8 - 16Mhz
In-System Reprogrammable Flash program memory	128 KB
EEPROM Data Memory	4KB
Internal SRAM	8KB
USB	Full Speed 2.0 (Device)
UART Channels	1

Table 1.3.: Some register of the camera.

POSIT Algorithm

The POSIT algorithm(Pose from Orthography and Scaling with Iterations) is an algorithm used to estimate the 3D pose from a 2D camera. The POSIT requires to detect feature points in the image, and to know the real shape of these points in 3D. With this data the algorithm can calculate the position of the camera/object in 3D space.

It is needed to select one of the feature points of our model as reference point. From there, the algorithm with the vectors from this reference point to the other feature points, will do the calculation of the rotation and translation vector of the object.

In figure 1.15 we can see how the 3D coordinate of an object ($\mathbf{M_0}$-$\mathbf{M_i}$) project in the camera plane (m$_0$-m$_i$). The P$_i$ and p$_i$ points are the scaled orthographic projection of the point $\mathbf{M_i}$. f is the focal length of the camera and Z_0 is the distance between the camera and the reference point in our 3D model.

From [17] can be seen that the equations that are need to be solved are:

$$\mathbf{M_0M_i} \cdot \mathbf{I} = x_i(1+\epsilon_i) - x_0 \quad (1.1)$$

$$\mathbf{M_0M_i} \cdot \mathbf{J} = y_i(1+\epsilon_i) - y_0 \quad (1.2)$$

$$\mathbf{I} = \frac{f}{Z_0}\mathbf{i} \quad (1.3)$$

$$\mathbf{J} = \frac{f}{Z_0}\mathbf{j} \quad (1.4)$$

$$\epsilon_i = \frac{1}{Z_0}\mathbf{M_0M_i} \cdot \mathbf{k} \quad (1.5)$$

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erreserbatuta dira

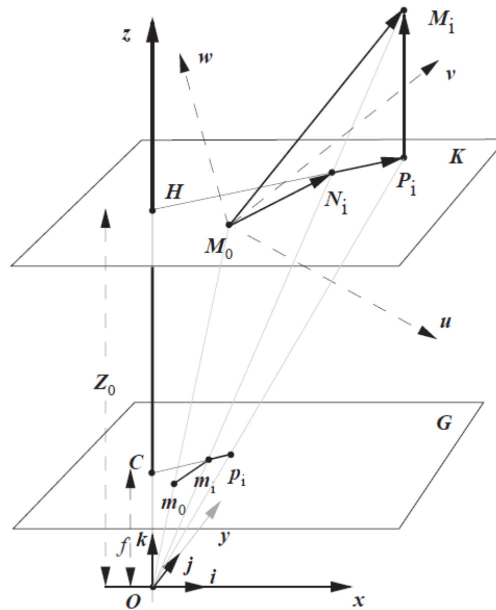


Figure 1.15.: Points in real world and their projection in the image plane[17]

When values are given to ϵ_i the equation system become a linear system in which only $\mathbf{I}$ and $\mathbf{J}$ are unknowns. Once $\mathbf{I}$ and $\mathbf{J}$ are found, we can calculate $\mathbf{i}$ and $\mathbf{j}$ by normalization. This part of the algorithm is the called POS(Pose from Orthography and Scaling). The orthography projection point p_i depends on the value of ε_i, so given an ε which is not exact to the actual value gives us only approximations to the pose. But once the $\mathbf{I}$ and $\mathbf{J}$ vectors have been calculated, improved values of ϵ_i can be computed and used in the next iteration to get better approximations to the pose. This last part is called POSIT(POS with iterations).

For the implementation two matrices are necessary: The A and B matrix. The A matrix is a matrix with the vectors of the points relative to the reference point in 3D space, and the B matrix is the pseudoinverse matrix of the A matrix. The shape of our real 3D shape will be fixed so the B matrix can be computed before so the microcontroller does not have to calculate it every time the program is executed. With this two matrices the algorithm is:

- **Step 1:** Set $\varepsilon_i = 0$ for every feature point in the object, so the POS algorithm can be used.
- **Step 2:** compute $\mathbf{i}, \mathbf{j}$ and Z_0
  - Compute the image vectors $\mathbf{x}'$ and $\mathbf{y}'$ coordinates. Each coordinate is calculated as:

$$\mathbf{x_i'} = x_i(1+\varepsilon_i) - x_0 \tag{1.6}$$
$$\mathbf{y_i'} = y_i(1+\varepsilon_i) - y_0 \tag{1.7}$$

  - With the image vector the $\mathbf{I}$ and $\mathbf{J}$ can be obtained by multiplying them, $\mathbf{I} = \mathbf{B}\mathbf{x}'$ and $\mathbf{J} = \mathbf{B}\mathbf{y}'$.
  - Compute the scale of the projection for each coordinate and calculate the average between them as the scale of the projection. $s_1 = \mathbf{I} \cdot \mathbf{I}^{1/2}$, $s_2 = \mathbf{J} \cdot \mathbf{J}^{1/2}$, $s = (s_1 + s_2)/2$.

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

  - Compute the non normalized vectors: $\mathbf{i} = \mathbf{I}/s_1$, $\mathbf{j} = \mathbf{J}/s_2$

- **Step 3:** Calculate the new ε_i values:
  - Compute the $\mathbf{k}$ vector as $\mathbf{k} = \mathbf{i} \times \mathbf{j}$
  - Compute the Z_0 of the translation vector as $Z_0 = \frac{f}{s}$
  - Compute the new ε_i for each feature point. $\varepsilon_i = \frac{1}{Z_0}\mathbf{M_0M_i} \cdot \mathbf{k}$

- **Step 4:** Check if the error is acceptable, difference between the new ϵ and the old one is less than a threshold. If the difference is greater than the threshold go to step 2.

- **Step 5:** Compute the translation vector and rotation matrix. The translation vector is $\mathbf{Om_0}/s$, and the rotation matrix is formed with the $\mathbf{i}, \mathbf{j}$ and $\mathbf{k}$ vector previously calculated.

By the end of the algorithm the translation vector and rotation matrix are obtained. The threshold level for the comparison between ϵ of different loops has to to be set manually, the smallest it is, the more iterations the algorithm will need.

LED structure

For the POSIT algorithm a fixed position of the lights should be used, so a structure to support the LEDs is necessary. Due to the limitations of the POSIT algorithm, the points can not be coplanar, so a cube with 4 light in alternative vertexes is used (figure 1.16). It is necessary to select one reference point to the POSIT algorithm, in this case the lower left light is our reference point, and the rest of the points are in clockwise order.

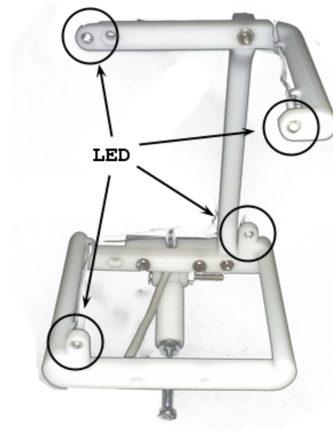


Figure 1.16.: Image of the LED structure used in this project.

USB

The USB protocol is a well known wired communication protocol. In our case the USB protocol is used to establish the communication between the microcontroller and a device capable of act as USB host. The configuration of the USB protocol is a very extensive topic so only the mouse descriptor will be commented. The base is a common mouse descriptor available as an example in the *HID Descriptor Tool* for the USB Webpage[36]. In this example the descriptor is for a mouse

[36] http://www.usb.org/developers/hidpage/

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

with 3 buttons, 2 axis of movement, and the wheel, but for our project, a customized version is used.

```
1  static uint8_t PROGMEM mouse_hid_report_desc[] = {
2   0x05, 0x01,    // Usage Page (Generic Desktop)
3   0x09, 0x02,    // Usage (Mouse)
4   0xA1, 0x01,    // Collection (Application)
5   0x05, 0x01,    //   Usage Page (Generic Desktop)
6   0x09, 0x30,    //   Usage (X)
7   0x09, 0x31,    //   Usage (Y)
8   0x15, 0x00,    //   Logical Minimum (0)
9   0x26, 0x00, 0x08, //   Logical Maximum (2048)
10  0x35, 0x00,    //   Physical Minimum (0)
11  0x46, 0x00, 0x08,  //   Physical Maximum (2048)
12  0x75, 0x10,    //   Report Size (16),
13  0x95, 0x02,    //   Report Count (2),
14  0x81, 0x02,    //   Input (Data, Variable, Absolute)
15  0xC0     // End Collection
16 };
```

With this descriptor the data that the USB host waits for is two blocks of 16 bits. Both blocks have to send the values between 0 and 2048, and the host will translate them as absolute movement in the X and Y axis.

libusbx

The *libusbx*[37] is a library that provides methods and functions to easily access to USB devices. It is a fork of *libusb*[38] which is less updated, so the *libusbx* is used.

The only configuration needed is to include the library directory in the linker. In the GCC compiler, with the Eclipse IDE, it is done by adding in the project properties the path and the library, or by adding the following line in the linker.

```
-L"<...>/LibUSB-Win32/lib/gcc" -lusb-1.0
```

Qt

Qt is a development framework to build applications and user interfaces for different platforms. The framework contains tools designed to speed up the development of the GUIs. The *Qt* code is reusable and the same code can be compiled to run on Windows, Linux or MAC, among others. This framework was selected for all the mentioned reasons, so it is used to make the PC program.

For this project the version 4.8.4 is used. It can be downloaded from *Qt download web page*[39]. Once downloaded the installation use 1.8 GB of hard drive.

In order to compile Qt code, an IDE is provided in the framework. However, the compilation of code, in which the *lib-usb* library was necessary, had some issues with MinGW (Windows GNU compiler). Therefore to write the code, the Eclipse IDE is used since it provides autocompletion when working with custom classes. To get the autocompletion with the *Qt* classes, the includes files have to been included in the properties of the project, as can be seen in figure 1.17.

With this configuration the Eclipse helps when writing the code providing autocompletion, however in order to compile the *Qt* code the command line has to be used.

[37] http://libusbx.org/
[38] http://www.libusb.org/
[39] http://qt-project.org/downloads

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

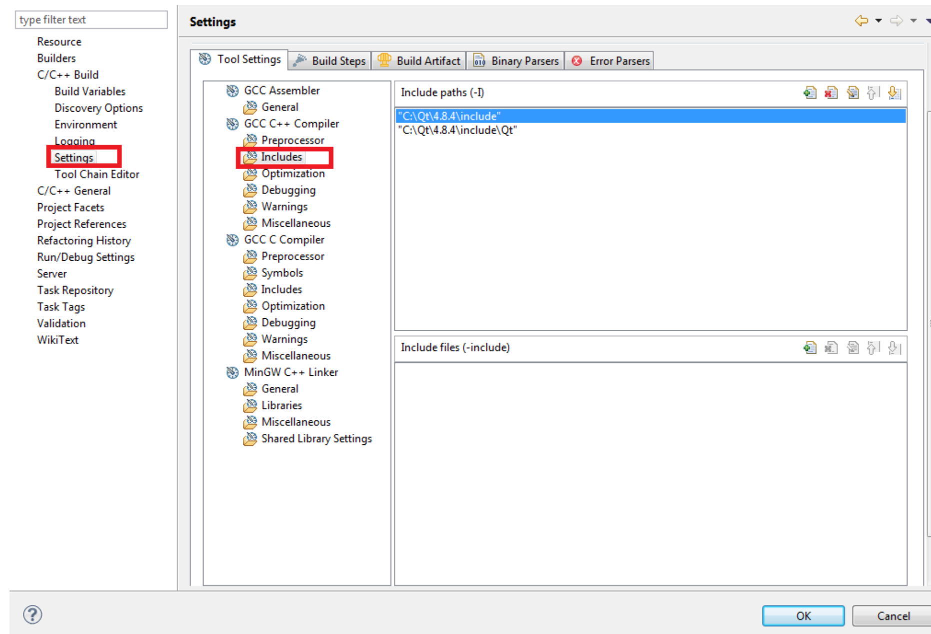


Figure 1.17.: Includes configuration in Eclipse for autocompletion with *Qt* libraries.

According to *Qt* tutorials around the web the following steps have to been followed to compile:

1. `qmake -project`
2. `qmake`
3. `make`

The first command generate the *.pro* file. This file contains some directives so the `qmake` can create a correct Makefile to the make. However, in our project the *lib-usb* has to be included in order to enable USB communication with the Teensy. In order to do that the -L<lib_usb_dir> -lusb1.0 parameters should be added to the linker. The `qmake -project` does not add the necessaries lines in the *.pro* file so the makefile fails when try to link the *lib-usb* library.

To fix this, the next line, in the case of using Windows, should be added in the *.pro* file to include the library:

```
LIBS += -L"<...>/LibUSB-Win32/lib/gcc" -lusb-1.0
```

The `qmake -project` generates a different *.pro* file if the project has new files or libraries. Then, as long as no new files are added to the project, the `qmake -project` command and manual editing of the *.pro* file can be avoided.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

2. Development

2.1. Hardware research and design

The first job in the hardware part was to discover how the camera is connected and which are the function of all the six pins it has. The camera were joined with a PL2303HX[1] which convert the data from the camera from the UART protocol to USB so it can be connected to the PC. The first task was to check the pins while the system was working.

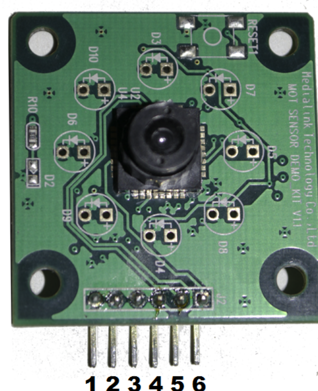


Figure 2.1.: Frontal image of the camera with the PINs numbered as in the schematic.

First, with the multimeter the DC pins were discovered, these pins are 1, 3 and 6, which are the ground, the reset and the supply respectively. Following the traces from the UART-USB bridge to the camera, the pins 4 and 5 turn out to be for transmission and reception. The last pin, the number 2 was unused, so it was the most difficult to understand. But after connecting it to an oscilloscope the clock signal was found out. This pin is and extra pin, that can be used to get the clock signal from the oscillator in the camera.

Once the camera was understood, the hardware design consists in the physical connection of the Teensy2++ module with the camera. The camera works with 3.3V in the data lines while the Teensy is configured to work with 5V, so the main task in the hardware design was to connect them with the right voltages. In the Teensy webpage[2] explains how can it be configured to work with 3.3V. The connection between them can be done:

- Directly if the modification to 3.3V is used in the Teensy.
- With some interface between the Teensy and the camera that adapts the voltages.

The first option, using 3.3V modification, implies that the microntroller has to work at 8Mhz since 16Mhz can only be achieved with 5V. This project is oriented to real time processing, so the

[1] http://www.electronicaestudio.com/docs/PL2303.pdf
[2] http://www.pjrc.com/teensy/3volt.html

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

frequency of the microcontroller has to be as high as possible, so it can do more operations per second. This option is therefore rejected.

The second option consists in adding an interface between them. As the Teensy is working with 5V we have to adapt the data and the supply lines. For the constant supply the easiest way is to use a voltage regulator LF33CV which converts the voltage from 5V to 3.3V. The configuration is as shown in figure 2.2. This circuit supply constant 3.3V in the supply pin of the camera.

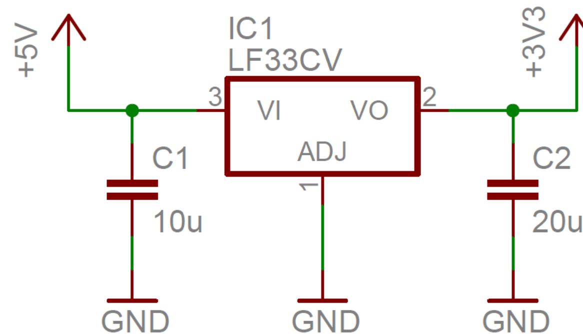


Figure 2.2.: Schematic configuration of the voltage regulator with the stability capacitors.

With the supply voltage for the camera, the necessary voltages in the data lines can be calculated so the microcontroller and the camera detects zero as zero and one as one. To achieve that, it is necessary that the voltages in the devices fulfil the requirements shown in table 2.1. According to the table, the transmission is possible in both ways. However, the maximum input voltage for the IO ports of the PAC7001CS is 3.6V, so an interface in need to down the voltage output from the AT90USB1286 to 3.3V.

	PAC7001CS	**AT90USB1286**
Input high voltage	≥ 2.31	≥ 3
Input low voltage	≤ 0.99	≤ 1
Output high voltage	≥ 2.97	≥ 4.2
Output low voltage	≤ 0.33	≤ 0.7

Table 2.1.: Electrical requierements for the data communcation

For the data transmission, the bandwidth of the elements is important, so the supply regulator cannot be used. Two options were considered: using a Zener diode of 3.3V or a voltage divider build from resistors. The Zener system would consist in a resistor in series with a Zener diode, the diode would let the necessary current flow so the voltage drop at the diode is 3.3V while the rest would drop in the resistor. The two resistors in parallel will do the same work but with a higher bandwidth. In that system the voltage drop at the first resistor, when the input is 5V, has to be 1.7V and 3.3V in the second resistor. This second approach is simpler and cheaper so it is the implemented one.

For the calculations of the resistor values, we have to take into account which is the maximum current in the pins of the microcontroller and camera. The AT90USB1286 can output up to 40mA of current. However, there is no data about the maximum input current.

upna
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

Figure 2.3.: Implementations of the voltage divider with the resistors and the Zener diode.

$$\frac{V_i - V_o}{R_1} = I_o + \frac{V_o}{R_2} \quad (2.1)$$

$$\frac{V_i - V_o}{R_1} < 40mA \quad (2.2)$$

From the second equation we can get the lowest possible value for R_1 is 42.5Ω. The output current will depend on the R_1 and since the limit value is 42.5Ω we select a higher value so the necessary current from the microcontroller pin is lower than the limit. The selected value for R_1 is 2.2$K\Omega$. With this value of R_1, and if V_o is 3.3V, then the current from the I/O pin of the microcontroller will be 0.77mA. From the equation 2.1 and the values obtained, we can derive:

$$I_o = \frac{5 - 3.3}{2.2K\Omega} - \frac{3.3}{R_2} = 0.77mA - \frac{3.3}{R_2} \quad (2.3)$$

With $R_2 = 2.2M\Omega$ the current that will be derived to the camera will be 0.769 mA which is not a high value, therefore the design of the interface can be seen in the next table:

Name	Value
R_1	2.2$K\Omega$
R_2	2.2$M\Omega$
I_i	0.77 mA
I_o	0.769 mA

Table 2.2.: Design parameters for the interface microcontroller-camera

With the design of the interface, and once the function of the pins of the camera is known, the final step is to connect everything together with the microcontroller. From the microcontroller only the UART TX and RX pins are used (pins 5 and 6 from the diagram, D2 and D3 from the Teensy2++ reference). The connection can be seen in figure 2.4. The 3.3V source comes from a part of the circuit shown in figure 2.2.

2.2. Software design

The implementation of the software part is done in C language. The IDE used is the Eclipse with the AVR plug-in and compiled with avr-gcc. The software can be structured in modules in order

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

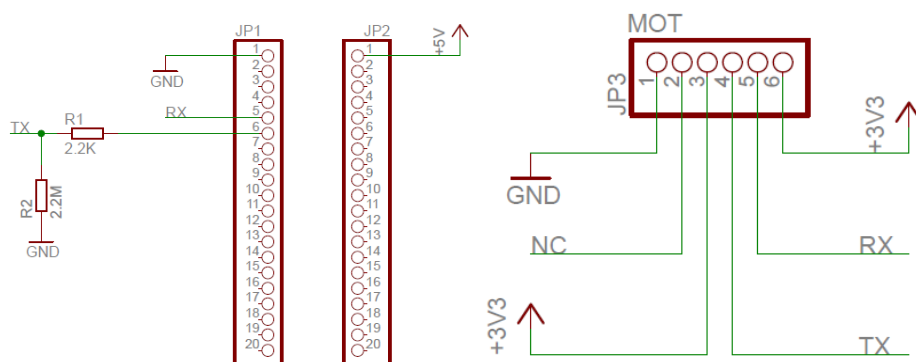


Figure 2.4.: Connection diagram of the Teensy2++ and the camera

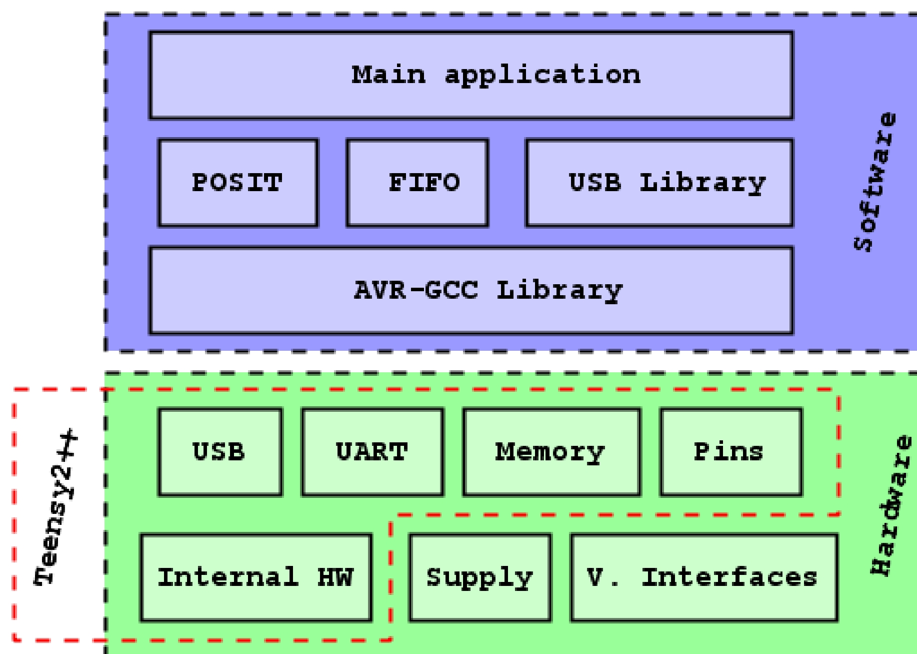


Figure 2.5.: Software structure divided in modules.

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

to abstract the hardware and the basic operations of the microcontroller. In figure 2.5 it is shown how the software is divided in modules that only depend on the layer beneath them.

In order to abstract all the microcontroller registers and variables, the functions to operate the UART, the FIFO and the USB are implemented. This method allows to program the software in a more readable way. The main software uses these libraries to access the more basic functions of the Teensy. These libraries also use the set of registers and instructions provided by the AVR-GCC library, which connects the software layer with the hardware layer. The physical elements of the microcontroller are found in the hardware layer.

2.2.1. Software overview

The main task of the software is to turn on the camera, get the object's coordinates and send the orders to the PC or device through the USB. For this purpose the software can be divided into simpler tasks:

- Configure the microcontroller.
- Turn on and configure the camera.
- Get the data from the camera.
- Apply the POSIT algorithm to the data.
- Send the data through the USB.

The first task is to configure the microcontroller. In this step the software turns on the services that are necessary in the next steps, also the FIFO queue is initialized, the clocks, USB and UART are turned on and configured. The interruptions of the USB and UART are enabled, so after this point the software does not need to configure or initialize protocols or pins, it just uses them.

Once all the systems inside the microcontroller are turned on and configured, the next step in to configure the devices outside of the AT90USB1286. In this project the only device that needs configuration outside the microcontroller is the camera. In this part of the process the camera is turned on and checked for possible errors. Once the correct message is received from the camera, the next step is to configure the registers of the camera for our purpose. The registers are set and then the camera is turned to Operation mode. In this mode the camera starts sending data of the objects it tracks. The data is stored in a FIFO queue which has been initialized in the first step.

With the data of the camera in the FIFO queue, the next step is to get the data and store it in an easier way to handle. The data of the camera comes in frames of data which are separated into structures that store the data in more human-readable format. Once the structures are filled with the necessary data, de POSIT algorithm is applied. The POSIT algorithm works with matrices of data, so the required values of the structures are therefore stored in matrix form before applying the algorithm.

From the FIFO algorithm we get the translation and rotation vectors for the camera. With this data the software has to calculate the coordinates for the mouse pointer to send them through the USB. For testing purposes there are two USB endpoints, the mouse endpoint and the debug endpoint. In the last one some debug information is sent and can be visualized in the PC.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

2.2.2. Data format and structures

For the different parts of the project some structures and data formats are used. The first step in our software is to get the data from the camera. The data is sent at 115200 baud per second, which is the baud rate that allows to send a full frame (four object in a frame,table 1.2). Depending on the processing speed of the data, a real time processing system can be achieved. For that reason, it is necessary to save the data obtained by the camera during the processing, in some buffer, so no data is lost. In this case, a circular FIFO queue was selected to act as the buffer for the UART input data. Due to simplicity purposes only the basic operations are implemented. The security checks are also kept at minimum. The structure that allocates the main variables of the FIFO queue is shown below.

```
typedef struct{
  unsigned char *pwrite;
  unsigned char *pread;
  unsigned char *end;
  unsigned char *buffer;
  volatile unsigned int count;
}fifo_t;
```

The FIFO queue consist on 4 pointers, `pwrite`, `pread`, `end` and `buffer`. The first two ones are used when read or write operations are implemented. `pwrite` points to the first available position to write, so when a write operation is necessary, the library writes where `pwrite` points and increments its position. If the end of the queue is reached (`pwrite` = `end`) then the new data is written in `pwrite` and after that the pointer is set to `buffer`. For the read operation the operations are the same but the read operation will fail if `pread` is equal to `pwrite`. The `pread` pointer points to the data to be read.

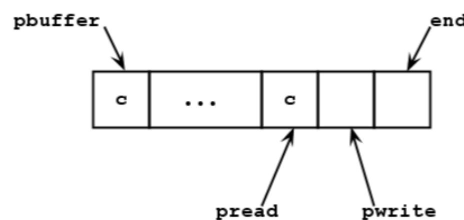


Figure 2.6.: Representation of the FIFO queue with its pointers.

The operations that are implemented for this queue are the followings ones:

```
void fifo_init(volatile fifo_t *fifo, unsigned char *buf);
char fifo_push(volatile fifo_t *fifo, unsigned char byte);
unsigned char fifo_pop(volatile fifo_t *fifo);
unsigned char fifo_empty(volatile fifo_t *fifo);
unsigned char fifo_full(volatile fifo_t *fifo);
void fifo_reset(volatile fifo_t *fifo);
```

Only the most important functions of the queue are implemented, the `fifo_push` and `fifo_pop` functions are to write and read respectively. In case of `fifo_pop` the returned value is the read data, while in `fifo_push`, the return value is a value representing the correctness of the operation.

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

The `fifo_empty` and `fifo_full` are auxiliary functions which return if the FIFO is empty or full respectively. The function `fifo_reset` is used to empty the FIFO queue. In pop and reset cases even if the data is read or "deleted", it is still stored only the pointers of the FIFO change. Finally the `fifo_init` initialize the FIFO queue, the input parameter is the FIFO to be initialized and a buffer (already initialized) which will store the data of the queue. With this functions the FIFO is fully operational for our purpose. In this project the FIFO is fulfilled with the data, but once the data necessary for the processing has been acquired the UART stop receiving data. Then FIFO is reset at the end of the processing so the available data is the newest one. The software does not need too much memory, therefore the length of the FIFO is set to 1024, that is bigger than we need. However, the available memory is not a problem.

Once the data is stored in the FIFO queue it is necessary to store it in more human readable format, for this purpose, in files *PAC7001.h* and *PAC7001.c*, the next structures and functions can be found.

```
typedef struct{
  uint8_t FlagByte;
  uint8_t ObjFlagByte;
  uint16_t  CenterX;
  uint16_t CenterY;
  uint16_t Size;
  uint8_t EOB;
}Objects;

typedef struct{
  unsigned char Header[4];
  Objects obj[4];
}CamFrame;
```

The function of these structures is to store the data from the camera, which come in a burst of bytes, in a more ordered manner. The structure of the `struct Objects` depends on how the camera sends the data, especially in how the register *Feature option enable flag* is configured, in our case this value will be 0x93 which correspond with the structure shown above. There are two structures, one for each object and the other for the full frame. The `Header` of the `CamFrame` is necessary to identify when a frame starts and to check that everything is working as expected, since if a header is different from 0xFF00FFFF something has gone wrong. With these structures the following function can be used.

```
int FrameCheckHeader(CamFrame *Frame);
int FrameFlagByte(int Getinfo, Objects *obj);
int FrameObjectCheck(int Check, Objects *obj);
unsigned char ProcessData(unsigned char* buffer, CamFrame *Dest);
void PACSetRegister(unsigned char RegNum, unsigned char RegValueHigh,unsigned char RegValu
void PACReadRegister(unsigned char RegNum);
void SwitchMode(unsigned char Mode);
```

The names of the functions are in most of the cases self-explanatory but a small overview will be done. The `FrameCheckHeader` check if the header of a `CamFrame` structure is correct. The next function,`FrameFlagByte`, is used to check if the object has been tracked or not, or to get

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

the frame number. The FrameObjectCheck is an auxiliary function to check the first bit of the Flag Byte and Object Flag Byte, which have to be one in both cases, if not an error has occurred. In ProcessData is where most of the work is done. This function takes as input the burst of bytes from the camera and store them in a structure. The first step is to recover the header and check it, if there is an error with the header, then the rest is rejected. If the header is correct the rest of the data is stored in the structure for future processing. The PACSetRegister and PACReadRegister work with the camera through the UART. These functions set and read de register, for the set operation RegValueHigh is only used in two register(the two registers with 2 bytes, IW and IH). The last function is used when the camera has to be changed from Initial Mode to Operation Mode and vice versa. These are the necessary functions to move the data from the buffer to structures, and then do the checks. With the data in the structure the final step is to apply the POSIT algorithm to the data.

In the POSIT algorithm some matrix and vectors are used. For the vectors the structure is a normal array, in which every position of the array corresponds to one element in the vector. For the matrix there are different possibilities:

- **Arrays of arrays:** In this case the pointer to the matrix will point to an array of pointers in which each pointer point to another array representing the row or column of the matrix. This format forces the processor to access one pointer, then move through the array, read the pointer and move to another array doing therefore lots of work to access one element of the matrix.

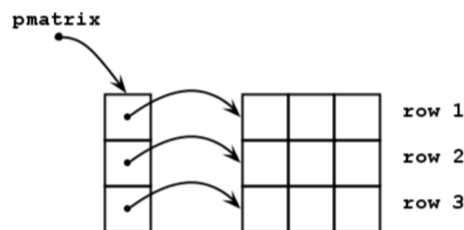


Figure 2.7.: Representation of a matrix build as an array of arrays.

- **All in a row:** this kind of matrix representation store all the matrix in one array in which the rows or columns are stored one row after another. This scheme is faster since only one address has to be accessed, the pointer to the matrix. Supposing pmatrix is a pointer to the matrix array storing a mxn matrix, to access one element the code will be pmatrix[COL*n+ROW] with the COL and ROW starting at 0.

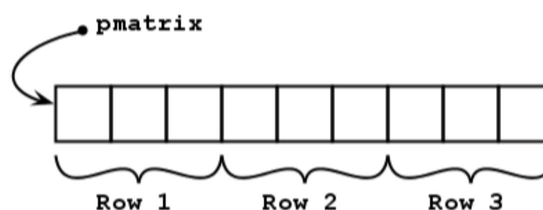


Figure 2.8.: Representation of a matrix build in only one array.

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

For the implementation of the POSIT algorithm the second option is used, storing one row after another. This method improves the speed and memory performance.

2.2.3. Actual block diagram

With the data structures and format the main program can be explained. The full block diagram can be seen in figure 2.9. The first step in the software is to configure the AT90USB1286 microcontroller so it can handle all the system that will be used.

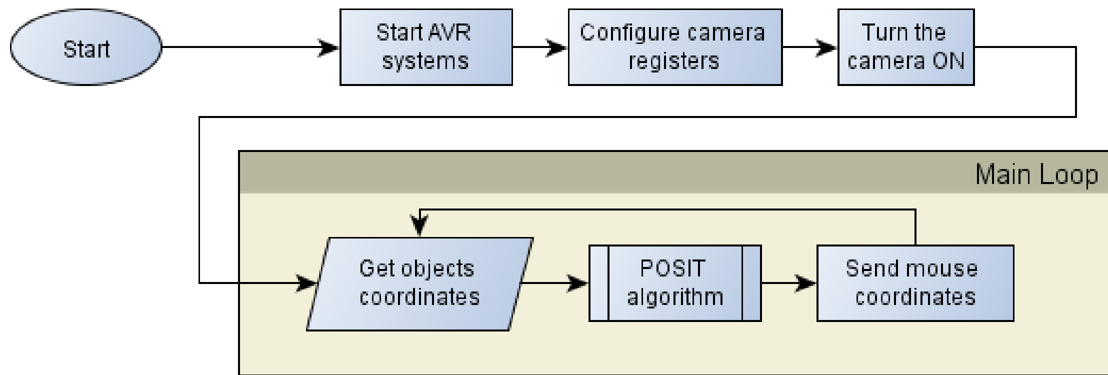


Figure 2.9.: Full software block diagram.

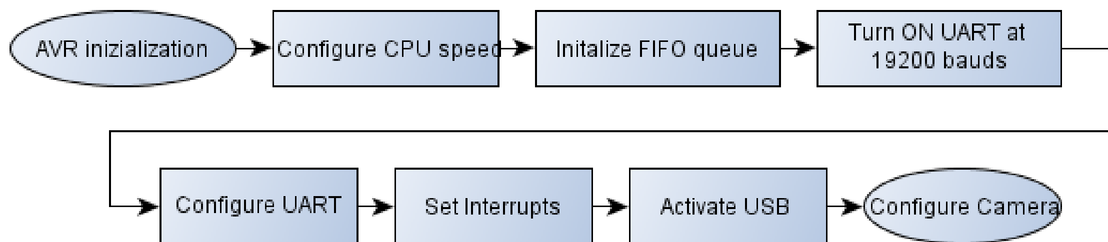


Figure 2.10.: Initialization procedure.

In the AVR initialization (figure 2.10) block the UART is turned on with a baud rate of 19200 baud ,which is the default baud rate that the camera has, when it turns on. Then the UART is configured to work with the camera protocol, that means with no parity bit nor stop bit and words of 8 bits. The UART of our microcontroller has the double speed option to achieve higher speeds of transmission. For the first baud rate high speed is not necessary, however when the UART has to work at 115200 Bd, the high speed register is set. In file *uart.h* the UART initialization function can be found as:

```
void uart_init(unsigned long baud_rate, unsigned char double_speed)
```

This function has to calculate the baud rate that the registers of the UART need. This value,when double speed is off, is calculated:

$$\text{Baud rate to registers} = \frac{\text{CPU freq.}}{baudrate \cdot 16} - 1$$

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

When double speed is on, the factor 16 changes to 8. The value obtained from the last equation is used in the registers UBRR1H and UBRR1L. The UBRR1H register stores the most significant bits of that value, while the rest is stored in UBRR1L. When the baud rate is set in the registers, the transmission and reception are enabled with no parity, 1 bit stop and character size of 8 bits with:

```
UCSR1B = _BV(TXEN1) | _BV(RXEN1); //Enable Tx and Rx
UCSR1C =  ~_BV(UPM11) | ~_BV(UPM10); // No parity
UCSR1C = ~_BV(USBS1);  //1 bit stop
UCSR1C = ~_BV(UCSZ12) | _BV(UCSZ11) | _BV(UCSZ10); //character size 8-bits
```

To send data through the UART the function uart_send_byte is used:

```
void uart_send_byte(unsigned char byte)
{
while ( !( UCSR1A & (1<<UDRE1)) );
UDR1=byte;
}
```

This function waits until the bit UDRE1 (USART Data Register Empty) in the register UCSR1A is set to zero. That means that the transmit buffer of the UART is empty and ready to receive more data. In that moment the data to be sent is stored in register UDR1, which will send the data as soon as possible. When the software needs to send a byte through the UART, this function will be used in order to abstract from how the microcontroller is doing it.

After turning on and configuring the UART, interrupts from UART receives are activated. Following the UART initialization comes the USB, with the activation of the USB pins and internal registers, the internal PLL for clocking the USB pins and the interruptions are enabled. The USB code comes in an example in the webpage of the Teensy2++, the USB mouse example[3], but after some changes it is suitable for our proyect. In order to identify the USB in the PC or USB host, the USB device has to be configured with some product ID and vendor ID. In our case the values for this parameters are defined as:

```
#define VENDOR_ID    0x16C0
#define PRODUCT_ID    0x047F
```

The communication between the device and the host is done through two pipes. The configuration of these pipes can be seen in table table 2.3.

Interface	Endpoint	Type	Use
0	3	IN	Mouse data
1	4	IN	Debug data

Table 2.3.: Pipes configuration for USB communication.

For the data format in the USB frames, the example comes with a simple mouse implementation, 2 axis, a wheel and 3 buttons, and a debug descriptor. The debug descriptor can be used without any change, however the descriptor of the mouse has to be adapted to our project. The example descriptor of the mouse is:

[3]https://www.pjrc.com/teensy/usb_mouse.html

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

```
1  static uint8_t PROGMEM mouse_hid_report_desc[] = {
2   0x05, 0x01,   // Usage Page (Generic Desktop)
3   0x09, 0x02,   // Usage (Mouse)
4   0xA1, 0x01,   // Collection (Application)
5   0x05, 0x09,   //   Usage Page (Button)
6   0x19, 0x01,   //   Usage Minimum (Button #1)
7   0x29, 0x03,   //   Usage Maximum (Button #3)
8   0x15, 0x00,   //   Logical Minimum (0)
9   0x25, 0x01,   //   Logical Maximum (1)
10  0x95, 0x03,   //   Report Count (3)
11  0x75, 0x01,   //   Report Size (1)
12  0x81, 0x02,   //   Input (Data, Variable, Absolute)
13  0x95, 0x01,   //   Report Count (1)
14  0x75, 0x05,   //   Report Size (5)
15  0x81, 0x03,   //   Input (Constant)
16  0x05, 0x01,   //   Usage Page (Generic Desktop)
17  0x09, 0x30,   //   Usage (X)
18  0x09, 0x31,   //   Usage (Y)
19  0x15, 0x81,   //   Logical Minimum (-127)
20  0x25, 0x7F,   //   Logical Maximum (127)
21  0x75, 0x08,   //   Report Size (8),
22  0x95, 0x02,   //   Report Count (2),
23  0x81, 0x06,   //   Input (Data, Variable, Relative)
24  0x09, 0x38,   //   Usage (Wheel)
25  0x95, 0x01,   //   Report Count (1),
26  0x81, 0x06,   //   Input (Data, Variable, Relative)
27  0xC0      // End Collection
28 };
```

The first problem with this implementation is that in our case we are not moving the mouse relatively but absolute to where the head is pointing, so the first step is to change the move from relative to absolute, changing the line 23 from:

```
0x81, 0x06,//   Input (Data, Variable, Relative)
to
0x81, 0x02,//   Input (Data, Variable, Absolute)
```

In our project the wheel and buttons are not used. Therefore, the following lines are deleted:

```
0x05, 0x09,//   Usage Page (Button)
0x19, 0x01,//   Usage Minimum (Button #1)
0x29, 0x03,//   Usage Maximum (Button #3)
0x15, 0x00,//   Logical Minimum (0)
0x25, 0x01,//   Logical Maximum (1)
0x95, 0x03,//   Report Count (3)
0x75, 0x01,//   Report Size (1)
0x81, 0x02,//   Input (Data, Variable, Absolute)
0x95, 0x01,//   Report Count (1)
0x75, 0x05,//   Report Size (5)
0x81, 0x03,//   Input (Constant)
...
0x09, 0x38,//   Usage (Wheel)
0x95, 0x01,//   Report Count (1),
0x81, 0x06,//   Input (Data, Variable, Relative)
```


upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

When trying to run the software with this USB descriptor the range of movement of the pointer was so limited because for relative movement, with a range between 0 and 127, is enough resolution. However for absolute movement 127 different points are not enough, so the resolution is changed in lines 19 and 20. Since the resolution necessary is higher than 255 points, the size of the report will also be increased from 8 bits to 16 bits.

```
0x15, 0x81,     //   Logical Minimum (-127)
0x25, 0x7F,     //   Logical Maximum (127)
0x75, 0x08,      //   Report Size (8),
0x95, 0x02,      //   Report Count (2),
to
0x15, 0x00,        //   Logical Minimum (0)
0x26, 0x00, 0x08, //   Logical Maximum (2048)
0x35, 0x00,       //   Physical Minimum (0)
0x46, 0x00, 0x08, //   Physical Maximum (2048)
0x75, 0x10,        //   Report Size (16),
0x95, 0x02,        //   Report Count (2),
```

With all this changes the USB descriptor fit our purpose and therefore is used. The final descriptor is as shown:

```
1  static uint8_t PROGMEM mouse_hid_report_desc[] = {
2   0x05, 0x01,   // Usage Page (Generic Desktop)
3   0x09, 0x02,   // Usage (Mouse)
4   0xA1, 0x01,   // Collection (Application)
5   0x05, 0x01,   //   Usage Page (Generic Desktop)
6   0x09, 0x30,   //   Usage (X)
7   0x09, 0x31,   //   Usage (Y)
8   0x15, 0x00,   //   Logical Minimum (0)
9   0x26, 0x00, 0x08, //   Logical Maximum (2048)
10  0x35, 0x00,   //   Physical Minimum (0)
11  0x46, 0x00, 0x08,   //   Physical Maximum (2048)
12  0x75, 0x10,   //   Report Size (16),
13  0x95, 0x02,   //   Report Count (2),
14  0x81, 0x02,   //   Input (Data, Variable, Absolute)
15  0xC0    // End Collection
16 };
```

The way the data is send to the computer has changed from 2 buttons, wheel and axis to only the axis, with 2 bytes for each axes. Therefore, the function that send the data to the computer need to be changed in order to send the correct data.

```
1  int8_t usb_mouse_move(int16_t x, int16_t y, int8_t wheel)
2  {
3   ...
4   /* keeps the same */
5   ...
6
7   UEDATX = LSB(x);
8   UEDATX = MSB(x);
9   UEDATX = LSB(y);
10  UEDATX = MSB(y);
11
12
13  UEINTX = 0x3A;
```


upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

```
14    SREG = intr_state;
15    return 0;
16  }
```

With the mouse data, the USB has to send the debug messages. The debug endpoint of USB and the function to operate it are included in the example program and need no change. For the use of the debug endpoint there are different functions to send data as numbers or strings. The function `print_P` is used to send complete strings through the USB debug endpoint. This function can be accessed through the `print` function. For numbers there are two options: `phex` and `phex16`, they should be used with 1 byte or 2 bytes numbers respectively. Their prototypes are:

```
void print_P(const char *s);
void phex(unsigned char c);
void phex16(unsigned int i);
```

All of this functions use the `usb_debug_putchar`. This function sends the characters through the USB. It waits until the USB lines are ready to transmit. Then, it selects the debug endpoint to transmit the data, and store it in the FIFO queue for the USB transmission. Once the USB has a complete packet of data, it sends it. The `print_P` function reads all the characters of a string and use the `usb_debug_putchar` to send them, one by one.

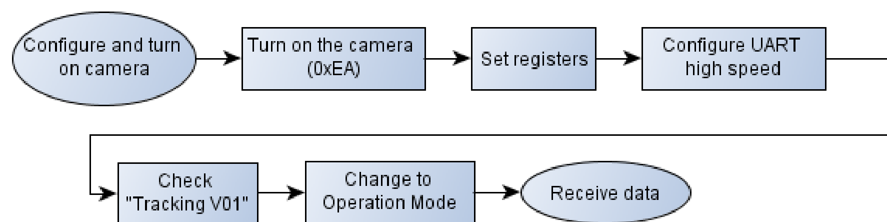


Figure 2.11.: Flowchart to set camera's registers.

The USB is the last system that needs to be initialized in the microcontroller, so once the USB has been configured, the next step is to turn on the camera, configure it and start receiving data (figure 2.11). The camera needs the `0xEA` command to be turned up so it is sent, followed by a `0x8E` just to be sure that the camera is in Initial Mode. In that mode the camera's registers can be set up, the values of them vary between applications and the actual values for each test will be commented on the tests part. After setting all the values of the registers the baud rate of the communication is set to 115200 baud. Then, in the debug window it is checked that the "Tracking V01" message and the ACKs from the camera are received. If everything were as expected the camera is changed to Operation Mode and the main loop of the program starts.

The main loop (figure 2.12) starts with a delay of 100 ms waiting for the data to arrive, until there are not enough bytes of data in the buffer the algorithm keeps waiting. When there is enough data the FIFO is read until a header (`0xFF00FFFF`) is found which means that from that point the buffer can be passed to the `ProcessData` function and be stored in a `CamFrame` structure. If the data is processed and there are not errors, then each of the objects in the frame is checked with `FrameFlagByte` function which checks the bit of the byte that represent if the object has been tracked or not. If is the case then the necessary operations are applied, in some cases is to

upna
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

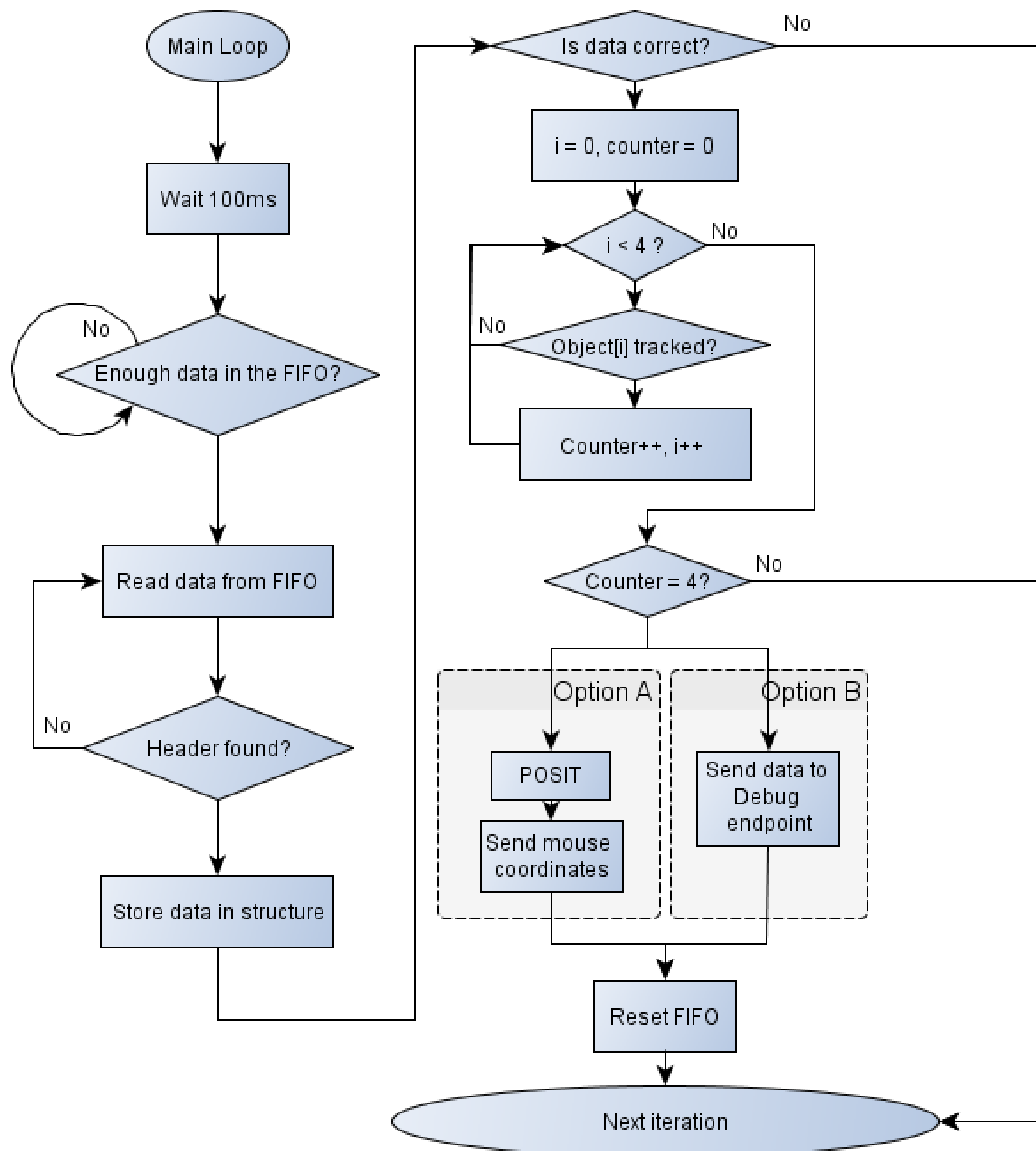


Figure 2.12.: Flowchart from the main loop of the software. The options A and B are for using the data with the POSIT algorithm or debugging it at PC.

upna
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

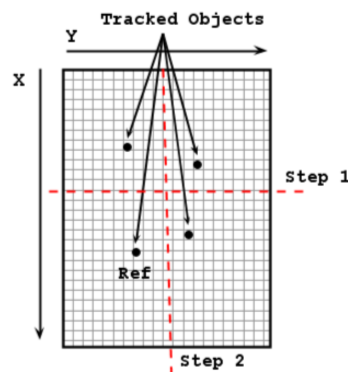
send the data through the debug channel to be analyzed with some PC software, in other cases the data of the four objects are stored and passed to the POSIT algorithm.

For the POSIT algorithm the matrices A and B are necessary, as shown in the methods sections. The matrix A is constructed with the vector connecting the reference point with each of the others points. The order in which the objects are included in the matrix has to be the same as the order in which the objects are sent to the POSIT algorithm. That is, if the first element in the matrix is the vector connecting the reference point to object a", then the first element in the array of data, for the POSIT, has to be the image coordinates of that object "a".

The tracked objects are, therefore, arranged in the same way that in the matrix for the POSIT. In the devices section we can see how is the structure for the LEDs (figure 1.16). The reference point is the lower left one, and then the objects are arranged in the matrix in clockwise order. In order to arrange them like that, the first step is to look for the reference point, which will be the one which has less Y value, from the two objects with higher X value (supposing that the camera orientation is as in the images, X axis vertical, Y axis horizontal).

The algorithm to order the objects will be:

1. Order all the elements in increasing X order.
2. From the two with higher X value, select the one with lower Y value, as reference point, and the other as last point.
3. From the two with lower X value, the one with lower Y will be the first point, and the other one the second point.



This method, of detecting which object is the first one and so on, includes a restriction in the pose detection. In some angles (extreme angles), the objects can be wrong assigned since the top objects are treated like bottom ones. The position, of the camera and the LED structure, necessary to achieve these angles are not common in normal use.

The data from the camera is referred to one corner of the image, however for the POSIT algorithm it is necessary that the coordinates are referred to the center of the image, so the first step in the POSIT algorithm is to refer the coordinate to the center. Then the needed variables are created and initiated, for the ε values there are 2 arrays, one for the current iteration and the other one for the last one, so at the end of the iteration they can be compared. The values of $\mathbf{x}'$ and $\mathbf{y}'$ are calculated:

```
x_[0] = x[1]*(1 + eps0[0]) - x[0];
x_[1] = x[2]*(1 + eps0[1]) - x[0];
```


upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

```
x_[2] = x[3]*(1 + eps0[2]) - x[0];

y_[0] = y[1]*(1 + eps0[0]) - y[0];
y_[1] = y[2]*(1 + eps0[1]) - y[0];
y_[2] = y[3]*(1 + eps0[2]) - y[0];
```

Then the calculation of vectors $\mathbf{I}$ and $\mathbf{J}$, and the values of s_1, s_2 and s are done. With these values the vectors will be normalized. Then the new values of ε are calculated. After that the error is calculated and if the error is below a threshold or the number of iterations exceed the limit, the translation and rotation vectors are calculated and stored in the respective matrices, else the new values of ϵ are used to do the same calculations and the process is done again. The auxiliary functions for the POSIT algorithm are related with vector operations. The functions are for the inner product, cross product and the difference between vectors. The prototypes are:

```
float InnerProduct(float *v,float *u, int len)
void CrossProduct3(float *v,float *u, float *res)
void difference3(float *v1, float *v2, float *v3)
```

The first implementation of the POSIT algorithm gives positive results, however some improvements can be made. To make the changes there are some facts that have to be taken into account. The used microcontroller only implement integer operation in the ALU, so floating point operations have to be implemented by software. That means that operations with floating points are expensive in microcontroller cycles. So, the operations with floating points should be avoided.

The first change made to decrease the number of operations with floating points numbers is to remake the calculation of $\mathbf{x}'$ with in the first implementation is:

```
x_[0] = x[1]*(1 + eps0[0]) - x[0];
```

Which is the direct implementation of:

$$x'_i = x_i(1 + \varepsilon_i) - x_0 \tag{2.4}$$

However this equation can be rewritten to:

$$x'_i = \underbrace{(x_i - x_0)}_{\text{Integers}} + \underbrace{x_i \varepsilon_i}_{\text{Floats}} \tag{2.5}$$

Since the main goal is to reduce operations of floating points, the algorithm goes from 3 floating points operations to 2 floating point operations and one integer operation. Since the difference between x_i and x_0 is always the same, for one set of feature points, it can be calculated before the loop and stored in a variable. As result, the calculation of one element of the $\mathbf{x}'$ is:

```
x_[0] = Difs[0] + x[1]*eps0[0];
```

The second step consists in reducing the operations while calculating the **I** and **J** vectors and length. The first implementation calculates the values of the elements, and then calculates the length of the vector. For each step a loop it is necessary, however if the length of the vector is calculated as the elements are, one loop can be avoided. The length of the vector is calculated by the square root of the sum of the squares of its elements. If the sum is done while getting the values, the second loop is avoided.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

The third improvement comes in the calculation of the Z_0. The first option was to store the value of Z_0 calculated as the focal distance of the camera divided by the parameter s. And when calculating the new values for ε, the inverse of the Z_0 is calculated. Since the value of Z_0 it is not used until the algorithm converges, the inverse of Z_0 is calculated so it is not necessary to invert it in every loop.

The last step is to include, the values that makes the algorithm converge in the last image, in the new calculation. According to [17], if the system is tracking the position of an object, once the algorithm has converged for one set of feature points, the values for ε should be used by the POSIT algorithm for the next image. Therefore in the main program the values of the ε are stored between images, so they can be used as starting point for the algorithm. With all this changes the prototype of the function is as follows

```
void POSIT(   unsigned short *xx,
              unsigned short *yy,
              float *A,
              float *B,
              float focal,
              short ImageCenterX,
              short ImageCenterY,
              float *translation,
              float *rotation,
              unsigned char *eps_
          )
```

Where the `*xx` and `*yy` are the arrays with the coordinates of the objects, already ordered. `*A` and `*B` are the matrices for the POSIT algorithm. Then comes the effective focal length of the camera. The `*translation` and `*rotation` are the matrices where the output (translation and rotation) will be stored once the algorithm has finished. Finally the `*eps_` is the array in which the value of ε is stored between images.

2.3. System operating range

Before the camera is used a more theoretical analysis is made in order to get the best spatial resolution that can be achieved. To do that we need the physical data of the camera which can be seen in section 1.3. The values that are interesting to do the analysis are grouped in table 2.4, while the diagram that represent the points, the camera and the image plane is in figure 2.13.

Parameter	Value
Focal length	1.3 mm
Pixel size	11μm$\times 11\mu$m
Sensor size (in pixels)	128x96

Table 2.4.: Camera necessary characteristics for the analysis.

In figure 2.13 de distance between the objects is $\mathbf{M_1M_2} = s$ and the distance between the two points in the image plane is $\mathbf{m_1m_2} = p$. With all this values we can now calculate which is the maximum and minimum distance s so the objects can be differentiated. The best conditions

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

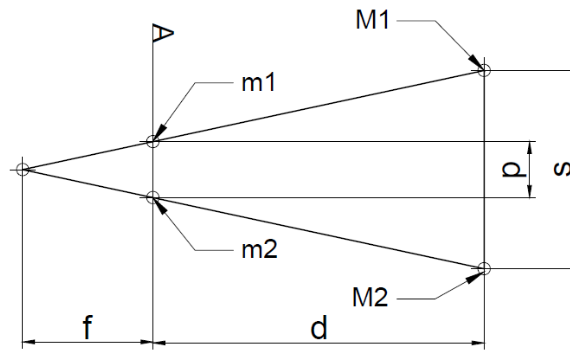


Figure 2.13.: Representation of the points in real world(M1 and M2) and in the image plane(m1 and m2).

are when the points, that the camera detects, use only one pixel per object. Which this premises the minimum distance that the camera will detect is when there is only one dark pixel between two bright pixels(the objects). This condition implies that the distance between the objects in the image plane must be at least 2 pixels of distance($p > 2\text{pixel}$). In the limit $p = 2\text{pixel}$, so with this condition and from the figure 2.13 we can obtain the minimum distance between objects in real world.

If an imaginary line cut the diagram horizontally in two, two similar triangles are found, so the relation between its sides is:

$$\frac{p/2}{f} = \frac{s/2}{f+d} \tag{2.6}$$

Which comes to:

$$s = \frac{f+d}{f}p \tag{2.7}$$

In $f + d$ the focal distance is 1.3mm and the distance of the objects will be usually around 1 meter, so the focal distance can be omitted. In the limit, the distance p between two points, has to be two times the pixel size, so changing the equation:

$$s_{min} = d\frac{2 \cdot 11\mu m}{1300\mu m} = 0.0169 \cdot d \tag{2.8}$$

Distance d(m)	Min. separation s(cm)
0,8	1,35
0,85	1,44
0,9	1,52
0,95	1,61
1	1,69
1,05	1,77
1,1	1,86

Table 2.5.: Values of minimum separation around the expected distance between the camera and the LEDs.

With the actual LED system in which the LEDs are separated by around 8 cm the maximum achievable distance is 4.73 meters, under best conditions. For the minimum distance in which

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

the camera will see the four lights, the calculation is similar. Using the diagram in figure 2.13, the distance $\mathbf{m_1 m_2} = p$ is the size of the sensor. In the biggest size the sensor has 128 pixels of $11\,\mu m$, that gives 1408 μm. Using the same equivalence as for the maximum distance, the closest the camera can be to the structure (which is 11 cm long) is 10.17 cm, according to:

$$s = \frac{f+d}{f} p \tag{2.9}$$

However, the biggest size of the structure is only seen when looking from the front, when the camera moves and change the angle of view, the distance between the points, seen by the camera, is not the 11 cm, so in some situations the distance can be even closer. Therefore, the range in which the camera would work should be between 10.17cm to 4.7 meters, under optimal conditions.

2.4. Tests

During the development of the software several test have been made. In each test some part of the software is tested. The following table shows the relation between the test that have been made, the file to compile and the description of the test.

Test	File	Description
Test 1	*TestUSB.c*	Communication between the camera and the USB host is tested. Basic communication is made, some text and random numbers are sent to the PC.
Test 2	*TestTracking.c*	Communication between the camera and the Teensy2++. The data from the camera is sent to the PC through the USB so it can be analysed.
Test 3	*TestACKs.c*	Camera configuration is tested. Some registers are set and read to check they actually have been changed with the orders from the microcontroller.
Test 4	*TestCameraData.c*	The data from the camera is stored and saved in the structures. The correctness of the algorithm that do the change is tested.
Test 5	*TestPOSIT.c*	With all the data from the camera, the POSIT algorithm is tested.

Table 2.6.: Relation between test's files and description.

Test 1: USB test

The first test checks the communication between the Teensy and the computer. The functions to communicate over USB were already implemented and can be found in the Teensy web page[4]. Because all the function related to USB were already implemented, the software of this test keeps really simple. The functions provided in the library, that are used, are `print`, `phex` and

[4] `https://www.pjrc.com/teensy/usb_debug_only.html`

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

phex16. They are capable of sending text and numbers of 8 or 16 bits respectively. In the test some sentences and two numbers are send as an example. In the PC side, the data from the microcontroller can be seen with the *hid_listen.exe* program.

After configuring the USB, the program wait 6 seconds until start sending data. This delay is used because the operating system has to load the driver for the new USB device before it can be used. After the delay the messages are sent using:

```
print("\nTesting....!\n");
print("8 bits number: ");
phex(123);
print("\n16 bits number: ");
phex16(16023);
print("\n");
```

Obtaining the the output in figure 2.14. As we can see the numbers are sent in hexadecimal format but everything work as expected, so now the work with the camera can start.

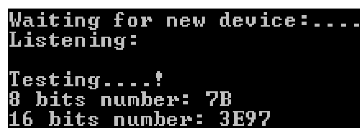


Figure 2.14.: Output of the hid_listen.exe program for the *testUSB.c*

Test 2: Camera connection

With the USB connection working the next step is to connect the camera. The software for this program is a continuation of the last one. As the UART is going to be used, the FIFO queue must be therefore activated and configured. So in the configuration step the UART and the FIFO are initialized. The interruption function for the UART stores the data in the FIFO, and turn on and off a LED so it can be used as sign of good operation. The interruption function is:

```
ISR(USART1_RX_vect)
{
   LED_ON;
   unsigned char rc = UDR1;
   fifo_push(&fifouart,rc);
   LED_OFF;
}
```

To turn on the camera the byte `0xEA` is sent using `uart_send_byte` function. Then, the message “Tracking V01” is expected to be received so, the data that is received from the camera is sent through the USB to the PC in order to be checked. The output of the program can be seen in figure 2.15. The camera message it is as expected

Test 3: Camera registers

Once the debugging messages can be analysed in the PC we can start working with the camera and the data it sends. Before the camera tracks the objects it is necessary to configure its registers.

upna
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

```
Waiting for device:...........
Listening:
Tracking V01:Tracking V01
```

Figure 2.15.: Output of the *TestTracking.c*

The registers are set with the `0x10` following by the number of the register to set and the new value for the register. For that purpose the next function is implemented:

```
void PACSetRegister(unsigned char RegNum,
                    unsigned char RegValueHigh,
                    unsigned char RegValueLow)
{
  uart_send_byte(0x10);
  _delay_ms(30);
  uart_send_byte(RegNum);
  _delay_ms(30);
  if( RegNum == REG_IW || RegNum == REG_IH)
  {
    uart_send_byte(RegValueHigh);
    _delay_ms(10);
  }
  uart_send_byte(RegValueLow);
  _delay_ms(30);
}
```

The camera send an ACK, which is a `0x10` code, after every register set command. To check that all the register have been set, the ACKs are checked and some registers are read as an example. As it can be seen in figure 2.16 all the ACKs arrive at the camera with the data of the first three registers.

```
Waiting for new device:...
Listening:
Tracking V01:
Tracking V01
ACKs: 1010101010101010101010101010101010101010
Values: 0A0076
```

Figure 2.16.: Output of the *TestACKs.c*

Test 4: Data and structures

When the camera is connected and configured, it starts sending data. The data that is received has to be stored in the structures with the functions provided to do that. In order to test these functions this test is made.

Once a full frame is received it is stored in the structure by the function `ProcessData`. Then, in order to debug, the data of the tracked objects are sent to the PC. In the following code it

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

can be seen that, if the frame is correctly processed then a loop checks every object, and if it is tracked it shows the data on the PC, as shown in figure 2.17. At this point the first research on the camera can be done.

```
if(ProcessData(<databuffer>, &<frame_structure>) == OK_ )
{
   for( i = 0; i<4; i++)
   {
     temp = Frame.obj[i];
     if( FrameFlagByte(FLAGBYTE_CONDITION, &temp) == FLAGBYTE_TRACEOK)
     {
         /* Send the formated data */
     }
   }
}
```

```
Listening:
Tracking V01:Tracking V01
161010101010101010101010101010101010101010930410

00 -> X: 037E| Y: 0304| Size:0454
01 -> X: 0368| Y: 052D| Size:0408

00 -> X: 0371| Y: 017D| Size:045F
01 -> X: 0355| Y: 044B| Size:0430
02 -> X: 0546| Y: 047D| Size:0544

00 -> X: 0367| Y: 0176| Size:0459
01 -> X: 0348| Y: 0443| Size:042D
02 -> X: 0538| Y: 0474| Size:053E

00 -> X: 0407| Y: 0228| Size:0426

00 -> X: 0342| Y: 0275| Size:047A
01 -> X: 0443| Y: 0456| Size:045E
02 -> X: 024A| Y: 046D| Size:0350

00 -> X: 0258| Y: 0300| Size:0631
01 -> X: 0376| Y: 052A| Size:0423
02 -> X: 0117| Y: 0537| Size:033B

00 -> X: 0530| Y: 0306| Size:0454
01 -> X: 0437| Y: 0512| Size:0410

00 -> X: 0553| Y: 0373| Size:0A57
```

Figure 2.17.: Output of the *TestData.c*

With the first data sheet provided by the manufacturer, the register's function were not clear, the limit values of the documented ones were not in the data sheet, and some registers were not documented at all. Therefore the first months, the goal was to understand the registers and how they affect the behavior of the camera. Although some of them were understood, they were not enough to achieve a correct behavior, the distance of tracking was too low, and the data the camera sent was not clear.

With the second data sheet, which was from another but close model, the research speeds up. And the true development of the head tracking system could start.

From this point on, the same base configuration(figure 2.7) for the camera will be used for every test, and only the values that are changed will be commented. The base configuration will be a

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

small modification of the recommended by the manufacturer in the data sheet from May 2006. The modifications are due to the example program, in which some registers were also changed, so they are also changed in our base configuration.

Register Num.	Name	Value
0x00	Gain 1	0xA
0x01	Gain 2	0x0
0x03	LPF	118
0x04	NY	1
0x05	IW	0x400
0x06	IH	0x300
0x07	-	0x88
0x0E	Obj. Assign mode	1
0x0F	Feature opt. enable flag	0x93
0x10	-	4
0x11	Baud rate	5
0x18	-	0x14
0x1A	-	30
0x1B	-	3
0x1E	Np_H	0
0x1F	Np_L	3

Table 2.7.: Base register configuration for camera operation.

With this configuration the first research can be done. The first step is to know how the camera sends the X and Y axes, which axis is the X axis and which one is the Y. The result can be seen in figure 2.18.

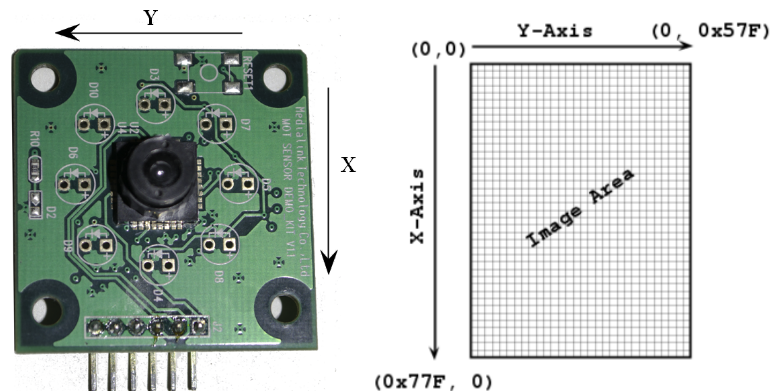


Figure 2.18.: Orientation and maximum limits of the image from the camera.

Once the orientation of the camera is known, the next step is to know which are the limits for the X and Y axes, which is to see which is the higher and lower value that the camera reports, which are 0x57F for the Y axis and 0x77F for the X axis. However, while doing the research, there were some issues with the data received from the camera. There are some forbidden values which are never reported. The received values from the X and Y axes are always in the ranges showed in the table 2.8. That means that if an object goes in the X axis, from 0x0 to 0x77F, when it reach,

upna
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

for example, the coordinate 0x7F then the next position will be 0x100, and that is the same for every block.

X axis	Y axis
0 - 7F	0 - 7F
100 - 17F	100 - 17F
200 - 27F	200 - 27F
300 - 37F	300 - 37F
400 - 47F	400 - 47F
500 - 57F	500 - 57F
600 - 67F	-
700 - 77F	-

Table 2.8.: Valid values for the X and Y axis (in hexadecimal)

This "jumps" in coordinates can be due to some malfunction of the hardware, a problem of the interpolation algorithm of the camera, or the data is sent as expected but needs more processing. If we focus on the data received and the ranges of the coordinates we can see that the last available coordinate is, in binary format:

$$\underbrace{XXXX\,XXXX}_{\text{Byte 2}} \quad \underbrace{0111\,1111}_{\text{Byte 1}}$$

The data of the camera could be said that is divided into blocks, indicated by the second byte. Each block is also divided in coordinates which are reported by the camera in the first byte. As the last available coordinate is always $0111\,1111$, the camera may work with signed integers even when the negative numbers are never used. In addition to that, if we sum up all the possible coordinates available in the ranges then it gives us: 1024 and 768 for the X and Y axis respectively, which are the image size configured in the registers. For these reasons the problems with the hardware or the camera's algorithm are rejected, and the camera and hardware are supposed to work as expected.

For the next steps in the project the coordinates have to be continuous. In order to fix it the coordinates received from the camera are processed in a simple way, the second byte is shifted 1 bit to the right and the first 0 from the first byte is deleted. This operation can be done with the next line of code, where CenterX and CenterY are variables of 2 bytes storing the coordinates.

```
X = ( (CenterX & 0xFF00) >> 1 ) | (CenterX & 0x007F);
Y = ( (CenterY & 0xFF00) >> 1 ) | (CenterY & 0x007F);
```

With this correction the values of the coordinates X and Y are continuous from 0 to 0x400 and 0x300 respectively. However, in order to use the POSIT algorithm, 4 lights are needed. While doing the tests some issues were noticed. The camera cannot track the four lights when the structure of lights is a bit far away from the camera. Sometimes the structure is reported as only one bright object, and even with the structure near the camera, most of the time the camera only detects 2 or 3 objects.

In order to make the system usable, it is necessary that the camera detects 4 objects from a distance of at least half a meter. The reasons to prevent that to happen may be:

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

- Bright lights: The light from the LEDs is so bright that the camera get overexposed, and the glow, from the different objects, intersects and makes the camera detects them as one object.
- Light beam: The light emitted from the LEDs may be wide enough so the camera see two light as one, and treat it as only one object.
- Blur due to the lens: The quality of the lens may blur the image and the objects. Even if the lights are focused they become blurry until they are detected as one.
- Lack of focus: Maybe some manufacturing issues make the image to be out of focus, generating blurry images as in the last option, blur due to lens.

Since the last two problems are out of our control, the system has to be improved trying to solve the first two problems, managing the lights and the camera parameters. To flatten the light, the LEDs are sanded down, so the light from them come in a more diffuse way, avoiding direct light to the camera. For the first problem with the parameters that we can touch are the gain of the sensor and the exposure time. For the gain of the sensor the registers that must be used are Gain 1 and Gain 2 , `0x00` and `0x01` respectively, while for the exposure time the LPF, Ny and Np_H and Np_L, registers `0x03, 0x04, 0x1E` and `0x1F`, must be used. The relation of these registers to get the exposure time can be found in the second datasheet and it is:

$$Exp.time = \frac{187 * (LPF + 1 - Ny) * 2 * Np}{\text{system clock}} \tag{2.10}$$

The Np register also affects the frame rate, so in order to affect only the exposure time the registers LPF and Ny should be used. Since the minimum of the LPF register is 110, and by default it is set to 118, the Ny parameter will be changed. It can be changed between 1 and the value of LPF (118 in our case).

The registers values that were tried can be seen in table 2.9. There can be seen how the system is improved, but not enough to work at normal distances. The first configuration is the default configuration recommended by the manufacturer. With these values the detection of the 4 lights can only be accomplished with really short distances between the camera and the structure (around 4∼6 cm), and only in some positions. As the lights keep getting confused in the camera, the next step, to reduce the amount of light that the camera detects, is to reduce the gain. This is done by taking the gain 1 value down to 0x3. With this change the camera start to detect the structure as one bright point easier, and getting 4 points of lights became mostly impossible. With this configuration can be seen that the problem is not with high gain in the sensor, so the next step is to reduce exposure time.

Since the decrease of the gain does not solve the problem, the exposure time is taken down, but compensated by higher gain. This is done by increasing the Ny register, the higher the value the lower the exposure time. In this case, the tracked objects just disappeared. Since they cannot be tracked, the exposure time is increased (fourth configuration). With this configuration the detection of the 4 objects becomes easier and the distance increases around 1 or 2 cm. The gain of the first register is at its maximum so to increase the gain, the second gain register should be used. However, the use of this second register has a bigger impact in the SNR, so this register has to be used carefully. With the second gain at `0xF` and the Ny at 20, the detection gets worse. That could be produced by the fact that, the gain has been increased, but the exposure time has been kept the same, so the light that the sensor detects increases. With this increase in the global

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

light, the objects get bright again and get tracked as one object. So the next step would be to take down the exposure to compensate the increase in the gain. In the configuration 5 the exposure has been compensated by the exposure time. However, with these values, the performance of the system decrease and the tracked object get harder to track again.

Considering the results, the best configuration for the camera registers is the third one, with gain 0xF and 0x0 and Ny 20. With this configuration the camera can track the four lights from a distance of around 8 cm up to 18 cm. Where the lights start to join and the camera sees them as 3 or 2 lights.

Nr.	Gain 1	Gain 2	Ny
1	0xA	0x0	1
2	0x3	0x0	1
3	0xF	0x0	50
4	0xF	0x0	20
5	0xF	0xF	20
6	0xF	0xF	40

Table 2.9.: Register configurations to solve overexposure images

Test 5: POSIT

With the track of the four objects, the POSIT algorithm can be tested. The first modification to the last code is to add some method to track how many objects are tracked. A counter is added, and while the structure is being processed the data of the X and Y coordinates are added to arrays, so they are ready to the POSIT algorithm. If the four objects are tracked, they are ordered in the same way as in the A matrix, and sent to the POSIT algorithm.

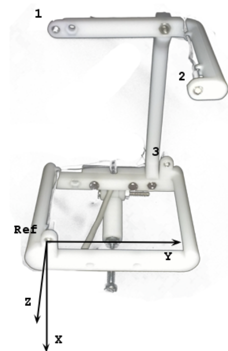


Figure 2.19.: LEDs structure and coordinates.

The coordinates of the camera should now be set. According to the orientation of the camera, the axes, showed in figure 2.19, are assigned with the origin in the lower left LED. The LEDs are orderer also in the way shown in the figure, and that is the order for the A matrix. The separation between the LEDs positions is 11 cm in X axis, 8.5 cm in Y axis and 9 cm in Z axis. With the

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

dimension of the structure, the A and B matrix are:

$$A = \begin{pmatrix} -11 & 0 & -9 \\ -11 & 8.5 & 0 \\ 0 & 8.5 & -9 \end{pmatrix} \qquad A^{-1} = B = \begin{pmatrix} -0.0455 & -0.0455 & 0.0455 \\ -0.0588 & 0.0588 & 0.0588 \\ -0.0556 & 0.0556 & -0.0556 \end{pmatrix}$$

The POSIT algorithm also needs the focal length of the camera, in pixels. Since the size of the pixels is $11\mu m \times 11\mu m$, and the focal length is 1.3 mm, the focal length in pixels is $\frac{1300}{11}$, 180.18 pixels.

With this data, the POSIT algorithm can be tested. According to [17], when the relation between the size of the object and the distance to the camera is above 0.5, the algorithm must converge in few steps. In our case the algorithm did not converge. In order to study the reasons, the *POSIT tester* software is used. With some data from the camera we can test the POSIT algorithm and save the evolution of the ε values in a file. With the algorithm of the camera several tests were made, however the debug information that can be obtained from the camera, in floating point operations, is limited. So in some scenarios, the values of the coordinates were introduced in the *POSIT tester*. The tests that were made are shown in table 2.10.

	X coordinates				**Y coordinate**			
Test Nr.	X_1	X_2	X_3	X_4	Y_1	Y_2	Y_3	Y_4
1	2AA	1B3	1B4	1A2	0EB	085	1DC	235
2	37C	217	208	31E	0F2	134	216	1F5
3	265	1DF	1C1	30C	053	0D1	1B6	19E
4	394	21E	1FE	324	06F	0C4	18D	19A
5	37B	1EF	1BD	306	062	0C0	19A	1A8
6	29E	12C	06F	272	064	130	1B3	22B

Table 2.10.: Firsts values for the POSIT tests.

In all these cases the algorithm diverges. With the data in the PC it can be analyzed. In the following graphs the variation of the first value of ε is plotted. This value should go to 0 when the algorithm converges. The graphs show the evolution of the POSIT algorithm until the divergence is clear. The divergence can occur in different ways, sometimes the divergence occurs at the first iterations, however in other cases it took more iterations, the algorithm can even oscillate, as will be shown in the next figures.

We can see how in tests 1, 3 and 6 (figure 2.20, figure 2.22 and figure 2.25) the divergence occurs in the first steps of the algorithm, between 10 and 15 iterations. However in test 2 (figure 2.21), the algorithm seems to diverge at iteration 33 but it is at the iteration 63 when it really diverges. The cases of the tests 4 and 5 (figure 2.23 and figure 2.24), are examples of oscillation in the algorithm. In this last case the algorithm will stay forever working and never improve.

Since the algorithm did not converge in any possible scenario, the problem may be the implementation of the algorithm, or the data introduced to the POSIT. In order to check the first possible source of the problem, the data from [18] is used. In order to use the coordinates, minor changes were done to the algorithm, the input parameters needed to be signed, and the correction of the center of the image were no longer necessary.

$$A = \begin{pmatrix} -56 & 0 & 0 \\ 0 & -56 & 0 \\ 0 & 0 & 56 \end{pmatrix} \qquad A^{-1} = B = \begin{pmatrix} -0.0179 & 0 & 0 \\ 0 & -0.0179 & 0 \\ 0 & 0 & 0.0179 \end{pmatrix}$$

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

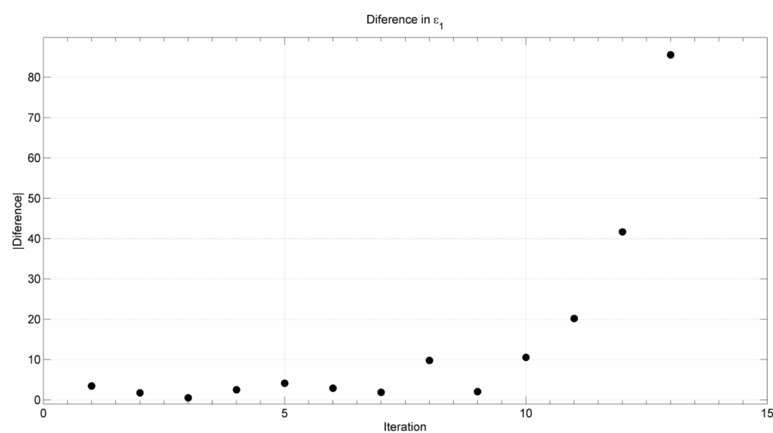


Figure 2.20.: Result of the first test of the POSIT.

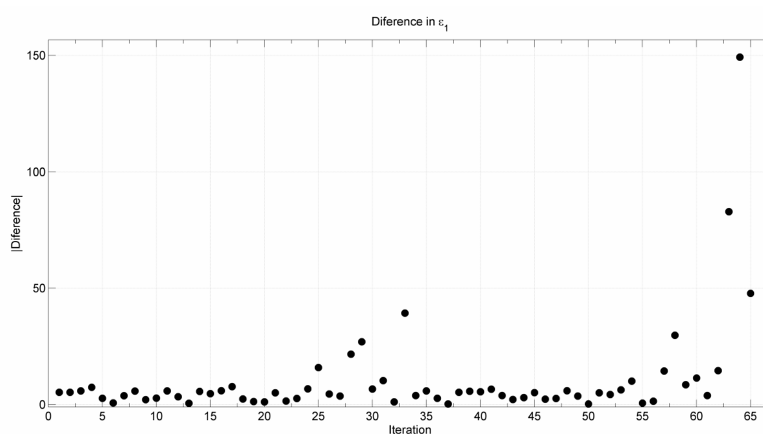


Figure 2.21.: Result of the second test of the POSIT.

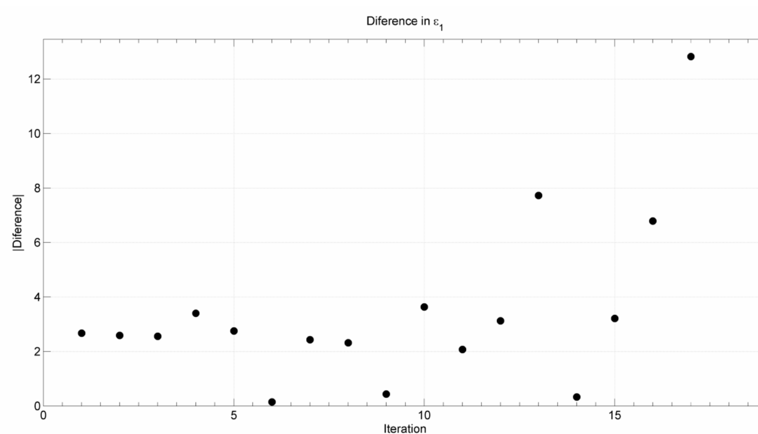


Figure 2.22.: Result of the third test of the POSIT.

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

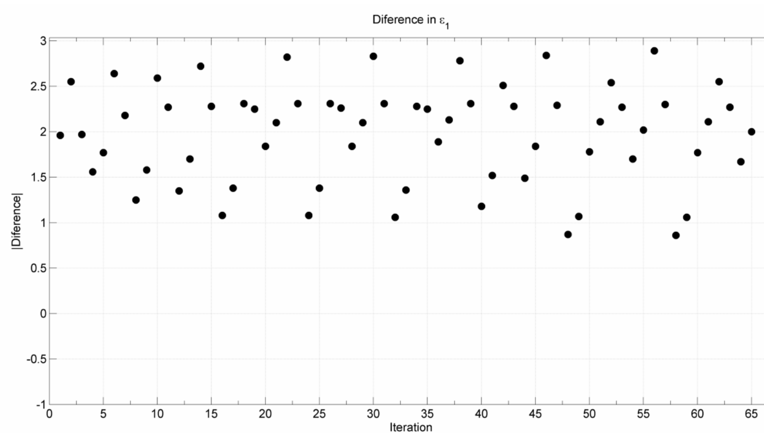


Figure 2.23.: Result of the fourth test of the POSIT.

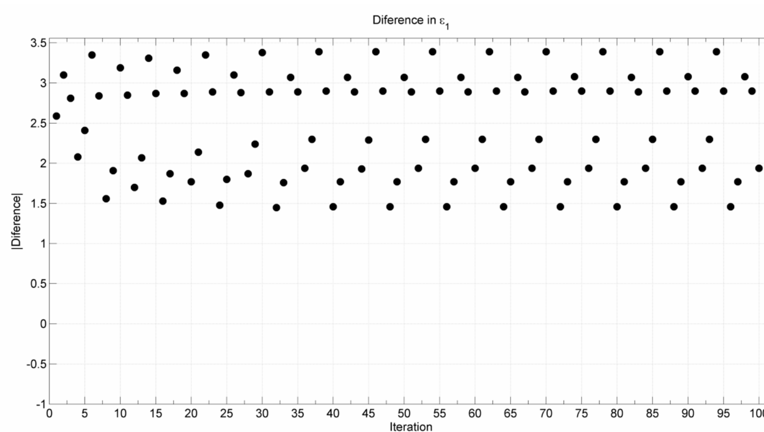


Figure 2.24.: Result of the fifth test of the POSIT.

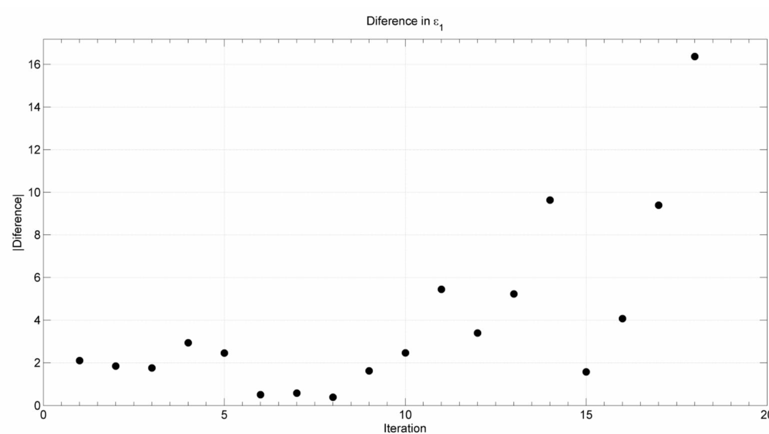


Figure 2.25.: Result of the sixth test of the POSIT.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

X	Y
-4	29
-180	86
-5	-102
76	137

Table 2.11.: Example data input for the POSIT algorithm.

With the coordinates of the objects that the example tracks, and the matrices for the POSIT algorithm, the only parameter left is the focal length. In the example use 640, the image width, as the effective length. With the data of the example, the algorithm converges in a few steps, as can be seen in figure 2.26, and the error is low almost since start.

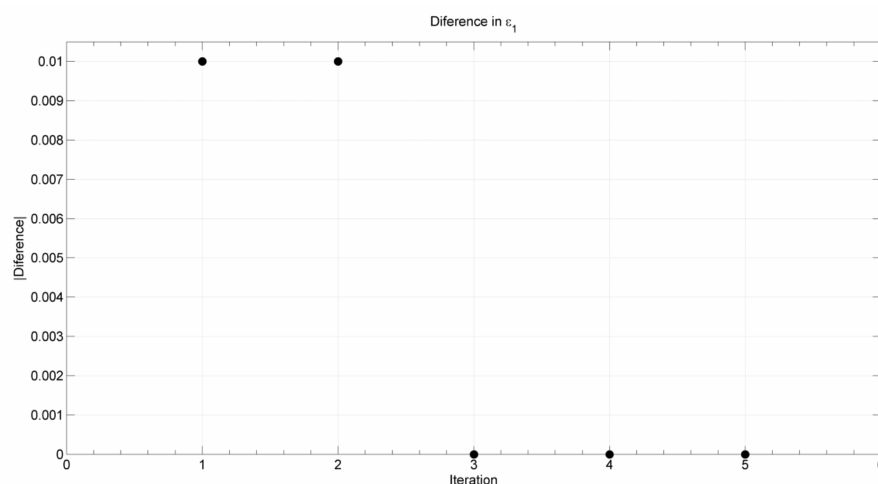


Figure 2.26.: Error evolution with the example data in the POSIT algorithm.

Since with the data of the example the algorithm converges, and gives the same values as in the example, the algorithm seems to work. So the only source of error is the data the POSIT algorithm is using. The value of the focal length in the example is much bigger than in our case. The first try is to increase the focal length to an arbitrary number. With a focal length of 1000, the algorithm always converges, and in few steps. So the problem can be the focal length, which is not the correct one.

The problem with the focal length is that it is referred to the sensor size of 128x96, however when doing the interpolation the image from the camera has a size of 1024x768. The values of the tracked object pass from being in the range 0-128 to the range 0-1024 in the X axis, for example. In order to have the same field of view, with the new size of the image, the relation between the size of the sensor and the focal length has to be the same in both cases. Since the size of the image is 8 times bigger than the size of the sensor (in pixels), the focal length has to be increased accordingly. If the normal focal length is 118.18 the effective focal length for the POSIT algorithm should be 945.44.

With the new focal length the same tests are repeated, but this time with the value of the new effective focal length. In all the cases the algorithm converges in few steps. Only the first example is shown as the convergence is similar among the examples.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

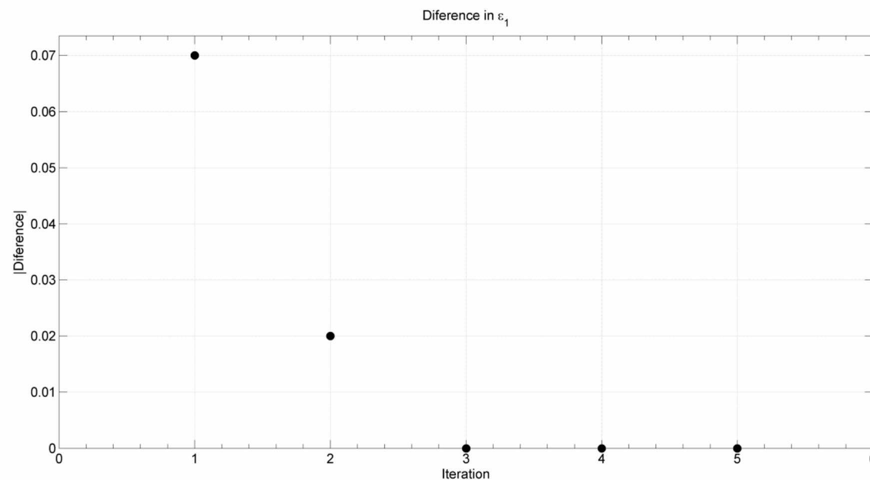


Figure 2.27.: Error evolution with the new effective focal length, for the first test data.

In figure 2.27 can be seen how the error, for the first test's set of data, evolve. As can be seen the algorithm converges, and it does for every set of data. So the next step is to introduce the new data in the microcontroller and move the pointer.

When the POSIT algorithm converges it creates a 3x3 rotation matrix. From this matrix, the move of the mouse has to be calculated. Since the XY plane of the reference system of the LED structure coincides with the monitor plane, and the Z vector of the camera points in the direction in which the camera is pointing at, the coordinates of the Z vector, referred to the LED structure's coordinate system, are used. With this situation, the X and Y coordinates in the structure's reference system are the Y and X coordinates for the mouse pointer.

In the rotation matrix, the last row is the Z vector of the camera, in the structure axes, so the X and Y component of that row are the displacement of the point of view from the center. This is a simple way to implement it, since the mouse will move according to the head pitch and yaw. However, to make the system working with, where the person is pointing at, is a little bit more complicated. And, since the camera does not work from user operation distance, the solution would be unable to be tested.

The values of the rotation vector are in the range between 0 and 1, because the values of the vector are normalized. The values that the mouse admits are between 0 and 2048. However, the values for the X and Y coordinates of the Z vector may not go to 1, because that would mean that the camera is oriented in the X or Y axis, and in that orientation the camera would not see the LEDs.

Therefore the values of the movement of the mouse should be approximated, i.e.: in this case, where the camera can only track objects from a distance of around 20 cm, in a screen of 15" (32x20cm), the maximum angle, to point , to one limit of the screen is 38.6° and 26.5°. With these angles the projection of the X and Y components are at maximum 0.44 and 0.62 respectively. If the values of this component are restricted to the range [0, 0.44] and [0, 0.62], and normalized so the maximum is 1024 (from the center), the mouse coordinates that have to be sent are:

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

$$\mathrm{X}_{\mathrm{mouse}} = \frac{R_x}{0.44} \cdot 1024 \quad (2.11)$$

$$\mathrm{Y}_{\mathrm{mouse}} = \frac{R_y}{0.62} \cdot 1024 \quad (2.12)$$

These values are the displacement of the point of view from the center. So the values that the algorithm is actually sending as pointer coordinates are, 1024-$\mathrm{X}_{\mathrm{mouse}}$ and the same for the Y coordinate. That is because in the USB descriptor the screen coordinates are in the range [0, 2048] and not in [-1024, 1024].

However, this way of sending data did not work, because the values of the coordinates exceeded the limits (0.44 and 0.62) and the values are truncated to them. In some cases the values fall into the range, in that case two things happen: in some range of data the system work as expected, but only in a few range, and in other cases the system did not converge as expected. That means that the POSIT algorithm is not converging to a good pose. Once the system is working as expected, and the moves are smooth, if suddenly the moves become awkward, should be because the algorithm did not converge properly. The reason for that could be that small changes in the actual pose imply big changes in the solution of the algorithm.

If the limitation for the values and the normalization, are eliminated (they are no longer forced to be in the ranges [0, 0.44] and [0, 0.62]), the system improves. In the X axis of the screen the movement is smooth in a big range, while the Y axis keeps the same value, or the variations are small, for every pose whatever is the position of the camera. The rest of the values in the rotation matrix were tested and no one gives positive results. The X coordinate of the vectors in the rotation matrix , which is the Y coordinate in the screen, does not change even when a change in the X axis occurs.

This bad pose estimation could occur because of the interpolation done by the camera DSP. The camera only has 128x92 pixels, and it gives images of 1024x786. That means that the camera has to interpolate some pixels, in the process some error may occur, and that errors affect to the algorithm result.

2.5. Support programs

While researching the camera, some support programs have been used and implemented. Some ones are already implemented by others while in some cases, when custom programs are needed, they are implemented for the project. Four programs are used:

- hid_listen
- HID Descriptor Tool[5]
- Debug GUI
- POSIT Tester

The first program, *hid_listen*, was used to show on the screen what the Teensy sends through USB. The program for the microcontroller and the *hid_listen* can be found in the Teensy web

[5] http://www.usb.org/developers/docs/hidpage/

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

page[6]. This program connects to the Teensy, receives the data from the debug endpoint and shows it in a console. This is the main resource to debug the software in the microcontroller.

The *HID Descriptor Tool* is software provided by USB organization, which is useful to check the correctness of the HID descriptors. It comes with examples of different common uses, such as mouse, keyboard or joystick. This program helps with the modifications of the descriptor of the mouse, it can check the correctness of the size of the data and the order or the items.

Debug GUI

For the initial debugging the hid_listen is enough, however in the last steps of the project, more graphical debugging is necessary. For that purpose the debug GUI is implemented, this GUI is based in the Qt libraries, which provide methods to implement it easily. The GUI has to provide a graphical representation of the data, so the basics functions that the program has to do are:

- Create the GUI
- Connect to the USB Device
- Get the data from the device.
- Treat the data and convert from a burst of data to coordinates of objects.
- Shows the data.

The data from the USB is received through the debug endpoint in plain text, that is why the data has to be formatted. Since the device never stops sending data, and the painting may be slow, the multi-threading approach is taken. In this approach there are two threads:

- **Main program:** Which renders the GUI and paints the objects representing the information from the camera. This program will create the other thread.
- **USB Thread:** The USB thread is the one connecting to the USB, receiving de data and sending it to the main thread so it can be painted. The data is sent through the USB with few control of what is in every packet. That means that in one packet of data of the USB there may not be a full frame of the camera. So this thread maintain a control of when the frame starts and ends, so only the necessary information is sent to the main program.

The communication between the microcontroller and the GUI needs a protocol to separate the frames and the data, so the GUI can treat them and paint the data. The frame starts with `'III '`(space included) then the object that is tracked is followed by 2 characters and a space. After that, the three coordinates beginning with x, y and s for X coordinate, Y coordinate and Size, are separated with a space. The data from the four objects are included if they are tracked. Finally the end of the packet is `' XX'`. A example of a frame, formatted the way just commented would be:

```
III 00 x0249 y012B s0515 01 x050B y0215 s0478 02 x031F y042E s0458
03 x050A y0f78 s0531 XX
```

[6] https://www.pjrc.com/teensy/hid_listen.html

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

With the all the concepts regarding to the GUI, it can be implemented. The first step is to decide the layout of the GUI. The GUI is composed of 2 text boxes, one paint area and the start button. In figure 2.28 the position of these elements can be seen. The first text box is used to see the plain text that the main program gets from the USB thread, usually a frame from the camera. The second text box is used for debug messages from the Debug GUI, instead of the data from the USB. The button just starts the USB thread so the program connects to the device and starts receiving data. The paint area is where, with the data from the camera, the objects are going to be drawn. Inside this paint area the bounds of the camera are painted to get a reference of the view of the camera.

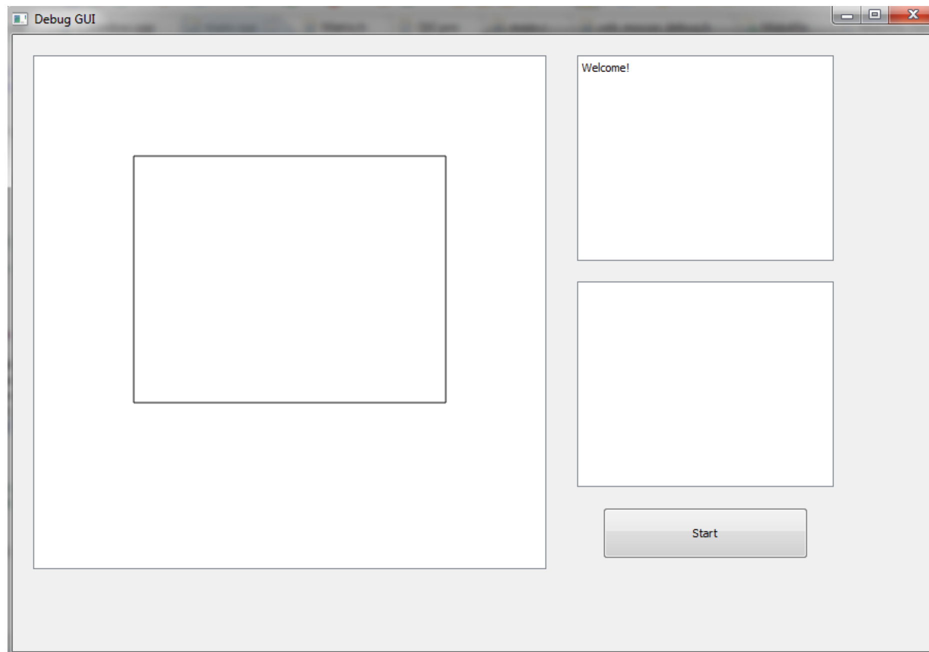


Figure 2.28.: Debug GUI layout.

Once the GUI is implemented and working, the function of the program is quite simple. When the USB thread is started, it connects to the USB device with Vendor ID `0x16C0` and Product ID `0x47F`, at the endpoint 4, which is the debug endpoint. Once it is connected, it starts receiving data. The thread waits until a full frame of the camera is received, before sending the data to the main program. That means that the data that the thread has received has to have the start and final marks (`'III'` and `'XX'`). Then the data stored by the USB thread is cut from the head mark to the final one, and it is sent to the main program to be treated. In the main program the frame is analyzed and the data from the objects of the camera are set of the objects to be painted, so they are drawn like the camera sees them. The USB thread stops receiving data while the GUI is still processing the last frame. When the processing finishes the USB thread starts receiving data again.

In order to coordinate the two threads, the *Qt* library implements the *signals* (source of the signal) and *slots* (destination). The signals has to be connected to one or various slots. The slots are the function that are going to be executed when the corresponding signal is emitted. This program has three signals:

- Signal 1: Connects the USB thread and the GUI. This signal sends to the GUI the data

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

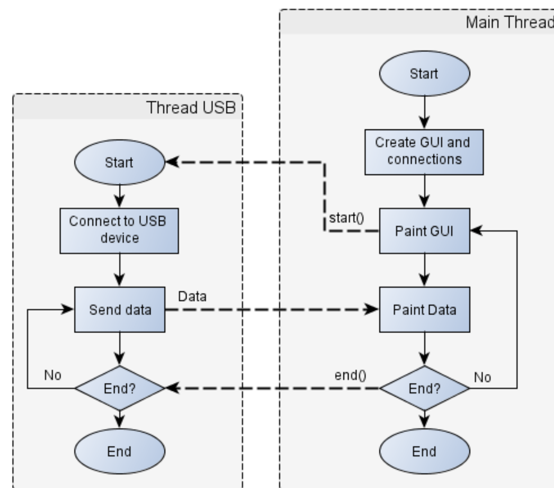


Figure 2.29.: Debug GUI flowchart

from the USB to be processed by the GUI.

- Signal 2: Connect the GUI to the USB thread. This signal is used to tell the USB thread that the processing of data has finished and it is waiting for more USB data.
- Signal 3: Connects the GUI to the USB thread. It is used to start the USB thread.

The *signals* and *slots* that are used in the program are in table 2.12.

Signal	Slot	Description
DataFound(String)	ChangeText(String)	When the USB thread has a full frame it sends it to the GUI so the data in painted and showed.
DataProcessed()	StopWaiting()	This signal is sent when the GUI ends processing the data and is available to receive new data from the USB.
clicked()	start()	This signal is used to start the USB thread when the start button is pressed.

Table 2.12.: *Signals* and *slots* of the Debug GUI

This program is used to represent the data from the camera. It shows where the camera see the objects. In figure 2.30 we can see how this data in shown. In the paint area, the circles represent the positions of the tracked objects. In the first text box, the data from the USB can be seen. In this example there is data about the coordinates of the objects with debug data of the POSIT algorithm. In the second text box, the debug information is about the program, is added to know how the program extract the coordinates from the USB data.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

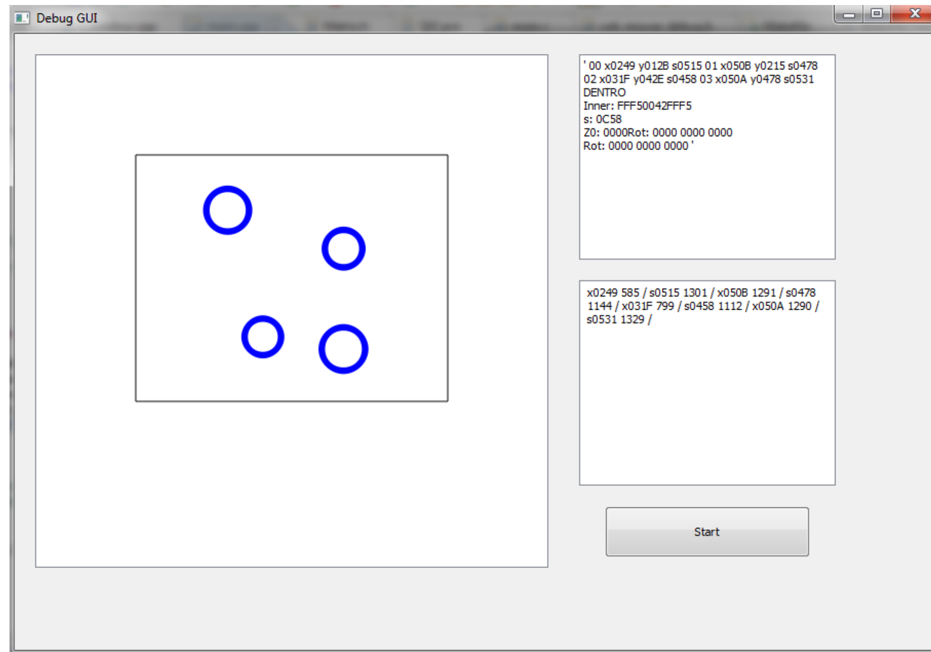


Figure 2.30.: Example of the Debug GUI, with some data.

POSIT Tester

The POSIT tester is a debug program for the POSIT algorithm. It is needed to analyze the data of how the algorithm is working. The debug information that can be obtained from the Teensy can only be integers and characters. Since the POSIT algorithm uses floating point numbers, the debugging though the USB becomes difficult.

The program therefore consists of a copy of the POSIT algorithm implemented in the AVR microcontroller with all the debug messages. The debug messages include all the steps in the algorithm in all the iterations. With the debug messages a *.txt* file is generated with the value of the eps value, so then can be analyzed, in this case with a *Matlab* script. The *Matlab* script, calculate the difference between the ϵ values between iterations and represent it, so the convergence can be easily viewed.

The implementation of the algorithm in the PC makes possible to change the values of the matrices A and B, or the focal length, easier and faster than in the microcontroller. This way the algorithm can be improved and corrected easily.

2.6. Future improvements

The purpose of the characterization and analysis of the system is focused on the application of tracking objects. More exactly in the tracking of a LEDs structure, so it can be used as a head tracking device. The requirements that the system must fulfil are easy to use, configure and install and it should be comfortable.

In the part of ease of use and installation, the system is already simple. It can be used as plug-and-play device, so once it is connected the system, it starts working as a mouse. Only with the movement of the head the pointer moves. However the movement of the pointer is awkward,

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

it can be improved by smoothing the transition between positions, or by implementing a shaking detection algorithm, so even when the head moves a bit, the pointer stays in the same place. The difficult part of this algorithm is to know when the user has changed the pose on purpose, or it was a shaking fit. Also in the actual design, the move of the mouse is done with the yaw and pitch of the head, however it would be better if the mouse moves to where the user is pointing regardless of the yaw and pitch, i.e.: If the user has the screen, lower than the camera, he would need to point down to the head, to look at the center of the screen. With actual implementation, when he looks down, the mouse would not be in the center of the screen but at the bottom.

For the ease of configuration, there are two important improvements. The first one may be a configuration endpoint in the USB so the camera registers and the AVR could be configured from the PC. That would need an extra software for the PC where the registers were set and sent to the AVR through another USB endpoint. This improvement is not so important because, once the system is configured to one scenario it would work without any changes, and if the user is not an expert the change in some values can make the system to fail. The second improvement in configuration would be some calibration method for the first time user. One option for the calibration method would be similar to the calibration with touchscreens, the user is asked to point at 4 points in the corners of the screen, with the data of rotation in X and Y for each of those points, the limits of the rotations are calculated and therefore the following values are limited to the limit previously calculated.

In the comfort area, the main task is to increase the distance at which the camera still tracks the full structure. With the current achievable distance, the use of this system in head tracking system is impossible, since the user has to be too close to the screen, and that can be harmful to his eyes. Also, if the position is not good it can be harmful for his back and neck. Therefore this area should be improved. However this improvement is not made by software but by hardware, changing the light sources, the layout or by changing the sensor.

In more special parts, the algorithm could be improved to be executed in less time with more efficient algorithms. The hardware is a prototype, so the wiring is poor. In future versions the system can be mounted on a PCB with better traces. For the camera, using the algorithm with only the 128x92 pixels, without the interpolation could be an improvement in the system.

In the debug part, the software provided is useful for its purpose. However they can be improved to have more user friendly interfaces, or to add more functionalities.

2.7. Summary and conclusions

Since the start of the project the goal was to assemble the devices in order to implement a head tracking device. The steps that were made were:

- Camera disassembly and reverse engineered (Section 2.1)
- Hardware design to connect the AVR to the camera (Section 2.1)
- Implement the software for the AVR in order to communicate with the camera and the USB Host. (Section 2.2)
- Analyse camera's data and registers(Section 2.4)
- Implement the POSIT algorithm (Section 2.2)

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

The main problems during the development were found in the reverse engineering. Since, once the hardware part was understood, the registers of the camera were more difficult. Also, the data provided by the manufacturer was incomplete until the last months of the development. Once the new complete data were delivered by the manufacturer the process speeds up. With the new datasheet the camera is configured to work with the 4 lights. Furthermore, POSIT algorithm is implemented and a bit improved in performance, for the AVR microcontroller.

From the tests we can get that even when the algorithm converges, the position of the camera and lights are not suitable for a head tracking system. This system could be tested with more bright and distant set of lights, maybe to track bigger and distant objects, or with less bright and near lights. For example the set for a small device, such as smartphone or tablets, where the distance between the device and the user is smaller.

Since the algorithm converges for small distances between the camera and the LEDs, the movement of the pointer is very nervous. The distance of the camera and structure, that makes the algorithm converges are small, and the structure use a big part of the image. That implies that small moves make the structure to not fit in the image, causing the algorithm to fail. That makes the algorithm to work only in small distances.

Because the development with the data provided by the manufacturer was difficult, the project also focuses in setting up debug software, so if this project is continued, the work of debugging is easier. These debug software can be used with every device that is programmed to send the data in the necessary format (Section 2.5), and with the proper Vendor ID and Product ID.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Bibliography

[1] Enabling Devices, "Assistive technology - products for people with disabilities," [Accessed: 17.06.2013]. [Online]. Available: http://enablingdevices.com/

[2] Independent Living Source, "Keyboards and mice," [Accessed: 17.06.2013]. [Online]. Available: http://www.iltsource.com/Assistive_Technology_Products_s/25.htm

[3] Infogrip, Inc., "Products," [Accessed: 17.06.2013]. [Online]. Available: http://www.infogrip.com/products/

[4] ZYGO-USA, "Sb-4h sip and puff switch with hardware," [Accessed: 17.06.2013]. [Online]. Available: http://www.zygo-usa.com/usa/index.php?page=shop.product_details&flypage=vmj_naru.tpl&product_id=314&category_id=139&option=com_virtuemart&Itemid=11

[5] Spectronics, "Catalogue," [Accessed: 17.06.2013]. [Online]. Available: http://www.spectronicsinoz.com/catalogue

[6] Hypertec Ltd., "url=http://www.keytools.co.uk/, note = "[accessed: 17.06.2013]"."

[7] EnableMart, "Shop our computer accessibility," [Accessed: 17.06.2013]. [Online]. Available: http://www.enablemart.com/computer-accessibility

[8] Pretorian Technologies, "Mouse alternative," [Accessed: 17.06.2013]. [Online]. Available: http://www.pretorianuk.com/mouse-alternatives

[9] A. Czajka, P. Strzelczyk, and A. Pacut, "Making iris recognition more reliable and spoof resistant," [Accessed: 17.06.2013]. [Online]. Available: http://spie.org/x14548.xml

[10] Tobii Technology GmbH, "Solutions for gaze interaction, development and integration of eye tracking," [Accessed: 17.06.2013]. [Online]. Available: http://www.tobii.com/en/gaze-interaction/global/solutions/

[11] Pertech, "Eye tracker Type L EyeTechSensor," [Accessed: 17.06.2013]. [Online]. Available: http://en.pertech.fr

[12] Seeing Machines Inc, "About faceLAB 5," [Accessed: 17.06.2013]. [Online]. Available: http://www.seeingmachines.com/product/facelab/

[13] D. E. Alonso, "Eye tracking search marketing 2009," [Accessed: 17.06.2013]. [Online]. Available: http://consultoriaenred.com/marketing-online/eye-tracking-search-marketing-2009

[14] L. Plunkett, "Report: Here are kinect's technical specs," [Accessed: 17.06.2013]. [Online]. Available: http://kotaku.com/5576002/here-are-kinects-technical-specs

[15] C. Foster, "Xbox kinect: Microsoft's key to the living room?" [Accessed: 17.06.2013]. [Online]. Available: http://readwrite.com/2013/01/07/xbox-kinect-microsofts-key-to-the-living-room

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

[16] P. J. Stoffregen and R. C. Coon, “Teensy usb development board,” [Accessed: 17.06.2013]. [Online]. Available: http://www.pjrc.com/teensy/

[17] D. F. DeMenthond and L. S. Davis, “Model-based object pose in 25 lines of code.”

[18] A. Kirillov, “3d pose estimation,” [Accessed: 17.06.2013]. [Online]. Available: http://www.aforgenet.com/articles/posit/

[19] G. R. Bradski and A. Kaehler, *Learning OpenCV: Computer Vision with the OpenCV Library*.

[20] Adaptive Tech Solutions, “Adapters, switches & mounts,” [Accessed: 17.06.2013]. [Online]. Available: http://www.adaptivetechsolutions.com/cat-adapters-accessories.cfm

[21] u. n. . A. . Mirametrix Inc., title=S2 Eye tracker.

[22] NAC Image Technology, “Eye tracking products,” [Accessed: 17.06.2013]. [Online]. Available: http://www.nacinc.com/products/Eye-Tracking-Products/

[23] SensoMotoric Instruments, “Smi gaze and eye tracking systems,” [Accessed: 17.06.2013]. [Online]. Available: http://www.smivision.com/en/gaze-and-eye-tracking-systems/home.html

[24] Carnegie Mellon University, “Cmusphinx wiki,” [Accessed: 17.06.2013]. [Online]. Available: http://cmusphinx.sourceforge.net/wiki/

[25] Julius development team, “Open-source large vocabulary csr engine julius,” [Accessed: 17.06.2013]. [Online]. Available: http://julius.sourceforge.jp/en_index.php

[26] Aachen University, “Rwth asr - the rwth aachen university speech recognition system,” [Accessed: 17.06.2013]. [Online]. Available: http://www-i6.informatik.rwth-aachen.de/rwth-asr/

[27] Spokentech, “Welcome to the zanzibar openivr project,” [Accessed: 17.06.2013]. [Online]. Available: http://www.spokentech.org/index.html

[28] LumenVox LLC, “Lumenvox automated speech recognizer (asr),” [Accessed: 17.06.2013]. [Online]. Available: http://www.lumenvox.com/products/speech_engine/

[29] Nuance Communications, Inc., “Dragon for pc,” [Accessed: 17.06.2013]. [Online]. Available: http://www.nuance.com/for-individuals/by-product/dragon-for-pc/index.htm

[30] Vocapia Research, “Speech-to-text technology,” [Accessed: 17.06.2013]. [Online]. Available: http://www.vocapia.com/speech-to-text-technology.html

[31] NaturalPoint, Inc., “Premium head tracking for gaming,” [Accessed: 17.06.2013]. [Online]. Available: http://www.naturalpoint.com/trackir/products/trackir5/

[32] J. C. Lee, “Johnny Chung Lee - Projects - Wii,” [Accessed: 17.06.2013]. [Online]. Available: http://johnnylee.net/projects/wii/

[33] Multimedia Information Processing Group at Kiel University, “Bias - basic image algorithms library,” [Accessed: 17.06.2013]. [Online]. Available: http://www.mip.informatik.uni-kiel.de/tiki-index.php?page=BIAS

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

[34] libusbx, "libusbx expand your usb potential," [Accessed: 17.06.2013]. [Online]. Available: http://libusbx.org/

[35] Qt Project Hosting, "Qt project." [Online]. Available: http://qt-project.org/

[36] H. A. Rodrigo, L. Carceroni, and C. M. Brown, "A fully projective formulation to improve the accuracy of lowe's pose-estimation algorithm."

[37] T. Petersen, "A comparison of 2d-3d pose estimation methods," 2008.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

List of Figures

1.1. Big push buttons 2
1.2. Keyboard layouts 3
1.3. Sip-puff systems examples. 3
1.4. Joysticks mice. 4
1.5. Name and year printed on spine. 4
1.6. Pedal mouse buttons. 5
1.7. Mouse trackpads 5
1.8. Eye tracking, pupil's features[9] 6
1.9. Examples of eye tracking 7
1.10. Head tracking examples. 8
1.11. *FaceLAB^{TM}5* 8
1.12. Heat map generated by a eye tracking system. 8
1.13. Kinect system for Xbox.[15] 9
1.14. Image of the Teensy2++[16] 15
1.15. Points in real world and their projection in the image plane[17] 16
1.16. Image of the LED structure used in this project. 17
1.17. Includes configuration in Eclipse for autocompletion with *Qt* libraries. 19

2.1. Frontal image of the camera with the PINs numbered as in the schematic. 20
2.2. Schematic configuration of the voltage regulator with the stability capacitors. . . 21
2.3. Implementations of the voltage divider with the resistors and the Zener diode. . . 22
2.4. Connection diagram of the Teensy2++ and the camera 23
2.5. Software structure divided in modules. 23
2.6. Representation of the FIFO queue with its pointers. 25
2.7. Representation of a matrix build as an array of arrays. 27
2.8. Representation of a matrix build in only one array. 27
2.9. Full software block diagram. 28
2.10. Initialization procedure. 28
2.11. Flowchart to set camera's registers. 32
2.12. Flowchart from the main loop. 33
2.13. Representation of the points in real world(M1 and M2) and in the image plane(m1 and m2). 37
2.14. Output of the hid_listen.exe program for the *testUSB.c* 39
2.15. Output of the *TestTracking.c* 40
2.16. Output of the *TestACKs.c* 40
2.17. Output of the *TestData.c* 41
2.18. Orientation and maximum limits of the image from the camera. 42
2.19. LEDs structure and coordinates. 45
2.20. Result of the first test of the POSIT. 47
2.21. Result of the second test of the POSIT. 47

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

2.22. Result of the third test of the POSIT. 47
2.23. Result of the fourth test of the POSIT. 48
2.24. Result of the fifth test of the POSIT. 48
2.25. Result of the sixth test of the POSIT. 48
2.26. Error evolution with the example data in the POSIT algorithm. 49
2.27. Error evolution with the new effective focal length, for the first test data. 50
2.28. Debug GUI layout. 53
2.29. Debug GUI flowchart 54
2.30. Example of the Debug GUI, with some data. 55

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

List of Tables

1.1. Some register of the camera. 13
1.2. Structure of the data received from the camera. 14
1.3. Some register of the camera. 15

2.1. Electrical requierements for the data communcation 21
2.2. Design parameters for the interface microcontroller-camera 22
2.3. Pipes configuration for USB communication. 29
2.4. Camera necessary characteristics for the analysis. 36
2.5. Values of minimum separation around the expected distance between the camera and the LEDs. 37
2.6. Relation between test's files and description. 38
2.7. Base register configuration for camera operation. 42
2.8. Valid values for the X and Y axis (in hexadecimal) 43
2.9. Register configurations to solve overexposure images 45
2.10. Firsts values for the POSIT tests. 46
2.11. Example data input for the POSIT algorithm. 49
2.12. *Signals* and *slots* of the Debug GUI 54

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

List of Abbreviations

ASR	Automatic Speech Recognition
BSD	Berkeley Software Distribution
CPC	CamPoseCalib
DLT	Direct Linear Transform
FIFO	First In First Out
IR	Infra Red
POS	Pose from Orthography and Scaling
POSIT	Pose from Orthography and Scaling with Iterations
SOP	Scaled Orthographic Projection
SR	Speech Recognition
TTS	Text To Speech
UART	Universal Asynchronous Receiver and Transmitter

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

A. List of provided files

Along with the Master thesis, some files are provided. They are separated in folders:

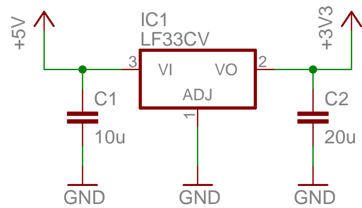
- **AVRprogram:** This folder contains the final code use in the project, it includes all the changes commented in the tests.
- **DebugSoftware:** Here are the files or the code to compile the programs used to debug the microcontroller and the POSIT algorithm.
  - *DebugGUI:* Includes the files to compile the Debug GUI for the microcontroller. This code should be compiled using the instructions in the introduction.
  - *HID Descriptor Tool:* Is the software from the USB webpage, as it comes from the web.
  - *PACTest:* The files for testing the POSIT algorithm in the PC. It only needs a GCC compatible compiler.
  - *hid_listen.exe:* Is the software provided by the Teensy manufacturer in order to debug it.
- **Docs:** Documents of the PAC7001, the first and second datasheet provided by the manufacturer.
- **MasterThesis:** The source code and the files necessaries to compile this pdf.
- **Tests:** In this folder the files used for the Tests are present, with all the files named in table 2.6, the file *TestCameraDataALL.c* is provided, where all me modifications commented in the last test are included. This files can be just changed from the file in the project (folder ARVprogram) and compile it to use the files.
- **teensy.exe:** Is the program provided by Teensy web page, to program the microcontroler.

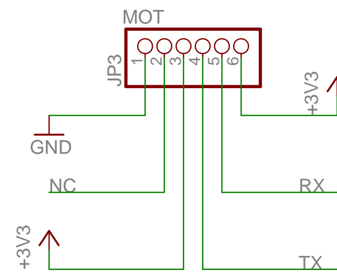
upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

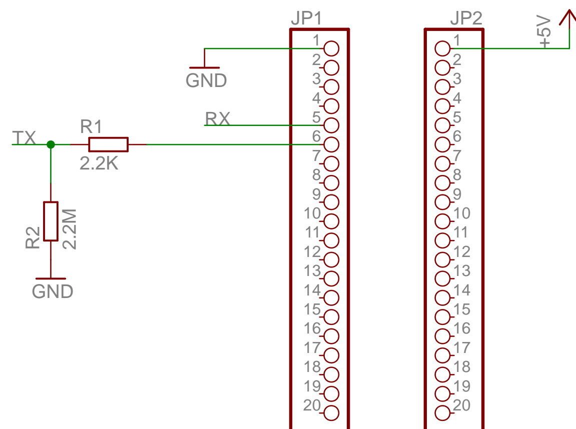
B. Hardware schematic

In the next page all the hardware connections and schematic are included.

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira







TITLE: MOTConnection

Document Number:

REV:

Date: 02/06/2013 13:10:59

Sheet: 1/1

upna
Todos los derechos reservados
Eskubide guztiak erresalbatu dira