INSTITUTO TECNOLÓGICO DE COSTA RICA

ÁREA ACADÉMICA INGENIERÍA MECATRÓNICA

PROYECTO DE GRADUACIÓN: Documento Final

"3D scanning and detection of objects for a subsequent manipulation by a collaborative robot"

Institution where project will be developed: Duale Hochschule Baden Wuertemberg (DHBW) University. Karlsruhe, Germany.

Profesor asesor: Prof. Dr. Juan Luis Crespo Mariño

Profesor guía: Prof. Dr. Clemens Reitze

Ejecutor: Alejandro Alpízar Cambronero

Carné: 200929524

Fecha: 22-06-2016

Content

Figures content

Dedicatoria

"A mis padres que estuvieron durante todo momento dándome apoyo y fortaleza, y que me hicieron creer que toda meta es posible de alcanzar."

I. Introduction

The present document is a brief description of what the project "3D scanning and detection of objects for a subsequent manipulation by a collaborative robot" will be about, and how the solution will be approached. The project is developed as a final requirement to obtain the degree of "*Licenciatura*" in Mechatronic Engineering at the Tecnológico de Costa Rica.

On the first part, the context of the project will be presented. It will be introduced different applications for 3D mapping technologies that can be found nowadays, as well as the different algorithms that are commonly used in this type of applications. It will be presented an outlook of some of the obstacles that might occur when handling this type of technology, and also the main authors and documents that were used to build a prospect of the realization of the project.

On the consequent part, it will be defined the problem to be resolved, with the environment where it will be developed. Also, the main problem will be divided in several steps, to facilitate the achievement of the overall project. Finally, it will be presented the motivation for the interested part to develop a project of such nature.

The next section will describe a strategy proposed to approach a solution for the entire project. It will be explained in more detail what each stage of the solution consist, and how every one of them is intended to be resolved.

Ishikawa diagrams will be used as a tool to identify different problems present on the work, and the possible root causes that may be inducing each problem. After this, the Objectives section will present both the general and specific objectives for this project, together with a specific indicator for each objective.

In the "Project execution" section, it will be detailed a list of steps to realize, to achieve the goals of the project. After this, a Gantt Diagram will be presented as an approximate to distribute time efficiently for the project. After this, a small summary of the expenses of the project will be presented, to understand the impact and feasibility of the project.

On the Solution's detailed description section, it will be described with detail all the steps that were taken to develop the project and to achieve the desired objectives. This section will be sub-divided to present the solution for each phase independently, with a final integration phase that connects all the solutions developed.

After presenting the solutions developed, an analysis section will be described, where the strengths and weaknesses of the project will be described. Followed by this section, the conclusions of the project will be stated and after this, it will be presented a section that mentions different recommendations and future work that can be developed after this project, to give continuity and achieve more goals on the future.

Finally, it will be presented a list of the sources and authors used to develop this document, as well as the important documents in the corresponding Appendix section.

II. Context of the project

In this section, it will be presented some of the most common uses that 3D mapping has, not only in industry but on the Research and Development area (R+D). The ability to obtain tridimensional information of certain space has been increasing over years, and different technologies to achieve this are applied nowadays. Some of the techniques and algorithms used will be presented on this document, as well as some of the obstacles that usually present in this type of applications.

The evolution of robotics brings new opportunities for human development. Every progress made in the research of new technologies brings not only benefits, but also makes noticeable the deficits that had not been detected previously on the various areas of growth among robotics and related areas (Sofge, Potter, Bugajska and Schultz, 2003). Robotics develops in a variety of shapes and functionalities, and it is evident that acquiring abilities inspired in nature are one of the main goals among researchers. Giving a robot the ability to interact with the world in a more real approach is one of the main interests in robotics. One of the abilities desired in robots, that is inspired in nature and also in human abilities is, the sense of sight. To achieve these, there are several attempts that are still on the spot of researchers, and that have grown in the last years.

The 3D scanning has become the attempt to document the real world and understand it better. There have been developed several techniques among time. As Ebrahim (2014) states:

> 3D laser scanning developed during the last half of the 20th century in an attempt to accurately recreate the surfaces of various objects and places. The technology is especially helpful in fields of research and design. The first 3D scanning technology was created in the 1960s. The early scanners used lights, cameras and projectors to perform this task. (p.4)

In the beginnings of robotics and development of technology, "blind" machines were helping humans to fulfill their needs, but with the advance on both of them, the needs increased, and with them the demands of evolved abilities for this robots. "Over the last years, various research groups have made substantial progress in recognition and manipulation of everyday objects" (Krainin, Henry, Ren and Fox, 2011, p.1), and many different techniques have been developed for acquiring this 3D information. In this document, some of the techniques for 3D scanning will be discussed as a reference.

There is for example, the use of RGB-D cameras. These specialized cameras capture images as a common RGB camera, but they also capture depth information per pixel. They get important visual information that is very helpful for loop closure detection (which will be discussed further later). However, using RGB-D cameras has its disadvantages. They have limited distance for information acquisition, the noise level on the measured distances can be problematic depending on the application, and they also have a restricted view angle (Henry, Krainin, Herbst, Ren and Fox, 2010).

There are many other restrictions or problems when using a camera technology for acquiring 3D information, such as the light conditions in which the applications is developed. The lack or excess of light or the uncontrolled conditions of it may cause shadows or faulty information, which may affect the measurements of depth among objects and scenarios. To avoid this problematic, laser technologies have been implemented to get depth information, that is not affected in the same way as cameras by the light on the environment (Valencia, Teniente, Trulls & Andrade-Cetto, 2009).

There are many attempts developed with both camera and laser technologies. Both of them have their own advantages and disadvantages. By one side, the cameras provide 3D information with less cost for the hardware. Collet, Berenson, Srinivasa and Ferguson (2009), for example, developed a vision system with a medium resolution camera that was able to detect the position of particular objects, and give further instructions to a Universal Robot to grip those objects. However, their method needed a previous training phase for the vision system, in which the computer was given the information of the possible objects to be observed. Needless to explain, this is a very restricted approach if there is a necessity of identifying a wide variety of objects.

The project that is to be developed has two main phases: on the first stage, the intention is to develop a vision system with laser technology for creating a 3D image of a given surface with a variety of objects. The system will be used in an indoor environment, however it is desired to measure the scope of the solution under different conditions, especially illumination, to measure the capability of the system to operate under uncontrolled conditions and which would be the restrictions of the system for this variables. On this first stage, it is also desired to apply vision techniques that will allow the system to identify, segment and locate the objects on the given surface (it could be for example a table or a box). This stage will take most part of the project, since it is the goal to create an optimum system which will be time and memory efficient. The second stage comes after having the information classified. Once the needed information is processed, it will be transmitted to a Universal Robot (UR), which will be programmed to grab each item and complete a function (dispose, package, pile, classify the item, etc.). Both stages will be discusses with more detail on this section, to understand better the context in which they will be developed.

First of all, the environment for the project needs to be defined. There are several applications currently for 3D mapping. They can be mainly classified within two categories: outdoor and indoor applications. For outdoor applications we can mention mainly navigation, either street navigation or building navigation. Despite building navigation could happen within doors, it can be considered outdoor, since there are no illumination controlled conditions and also a wide variety of static and dynamic obstacles may appear. The application desired is going to be operated on indoor environments, so there is no need to deepen on outdoor applications, however there are important contributions on 3D mapping on previous studies for outdoor environments that will be taken into consideration, especially the ones from Henry et al. (2010), Valencia et al. (2009) and Hornung, Wurm, Bennewitz, Stachniss and Burgard (2013).

For indoor applications, there are still several conditions to be specified for this project. Inside mapping could be implemented for static scenes or for scenes where objects can move, which will lead the investigation to a completely different course. For the purpose of this application, a static scene will be the situation of study. Therefore, we need to study the strengths and weaknesses of 3D scanning laser methods for indoor static environments.

As Henry et al. (2010) mention in their study:

> Building accurate, dense models of indoor environments has many applications in robotics, telepresence, gaming, and augmented reality. Limited lighting, lack of distinctive features, repetitive structures, and the demand for rich detail are inherent to indoor environments, and handling these issues has proved a challenging task for both robotics and computer vision communities. Laser scanning approaches are typically expensive and slow and need additional registration to add appearance information (visual details). Vision-only approaches to dense 3D reconstruction often require a prohibitive amount of computation, suffer from lack of robustness, and cannot yet provide dense, accurate 3D models. (p.13)

Hence, it needs to be taken into consideration the characteristics of specific components that might be troubling when scanning. For example, any part that has lack of distinctive features among its body could be challenging to scan, since there are no points of reference when computing the full 3D image. There is also an important need to take into account the type of light that will be used on the working area, as well as the time for the scanning and the amount of memory required for each scan, without giving up resolution of the objects of interest. For this, different algorithms previously experimented will be studied, to determine the best one that satisfies the limits established for time and memory consumption.

As mentioned before, considering the illumination is important, despite the application using a laser scan and not a camera. Laser scans are a type of Time-of-Flight sensor. They use a light source, which could be a LED or in this case a laser. They also have an array of pixels that detect and measure the incoming light and its correspondent phase. This will be used to determine the distance from a specific point on space. The measure of received light depends not only on the distance of the object, but also on the reflectivity of the object that is been evaluated. Therefore, the light present on the surroundings does affect the measurements and should be considered. To get a better resolution of the scanned objects, an option would be to utilize a higher power laser, but this would increase the power consumption and overall cost of the system. Another option to avoid this would be to reduce the scanned area or to extend the scanning time of the object (Gokturk, Yalcin and Bamji, 2004). There are then three variables that need to be considered and pondered very cautiously: resolution of the images, cost of operation and cost of the system, and time invested on the scan.

To approach the study to a more real situation, it needs to be considered that there a different technologies within laser scanners, every one of them with certain restrictions. There are also a wide variety of algorithms that have been implemented to fit every requirement according to each specific application or need. Schwarz (2010), for example, mentions two variations of LIDAR systems. The first variation functions with a single laser that is directed to rotating mirrors (this refers to the usual LIDAR systems). However, there is a variation for high definition scanning that uses a rotating head with 64 lasers. Every laser works on a frequency of 20 KHz. This makes possible to retrieve higher amounts of information than common methods, which at the same time allows the system to get better resolution on the outcome images. Schwarz also mentions that some of the most common applications for these systems are robotics, mapping, navigation and manufacturing. This last application would be a clue for this project, since one of the possibilities for exploiting a system such as the one to be developed would be manufacturing itself. Since the project is not developed for a company but for a university, it doesn't have a specific use or end. On the contrary, it seeks to give use to such technologies expanding the knowledge and researching the possibilities of use for the system. Henry et al. (2010) introduce the reader to pure laser-based ICP technologies, and discuss the robustness for 3D point clouds obtained with SICK scanners and 3D Velodyne scanners.

The information supplied by the university DHBW indicates that the sensor available is a light section sensor from Leuze-Electronic.de. However, there is no further information regarding the sensor. The very first step to be taken care of would be acquiring specific information on the sensor, on how it functions and which algorithms it uses at the present time. Valencia et al. (2009) use a Leuze RS4 sensor for urban scanning and mention a sensor noise of ±5cm in depth estimation, but this information is subject to the applications and the environment, which are for this study very different than the conditions present for Valencia et. al (2009). For the specific application on this project, it is desired a much smaller noise than the one mentioned here.

A 3D scanning process is usually integrated by three main stages: 1) Obtaining continuous frames from an object with depth information from different positions, 2) detection of loop closure from information acquired, and 3) integration of all the scans into one whole 3D object. (Henry et al., 2010 and Weise, Wismer, Leibe and Gool, 2009). A loop closure is basically the detection of reference features on an object that determine the complete lecture of that object. It happens when the sensor has reached a point where the scanner started recording information and has enough key features that ensures the closure of the object. This loop closure has a common problem that comes with the "accumulation of registration errors" (Weise et al., 2009) when connecting the different frames previously obtained by the depth sensor. Every frame connected to the next one may add a small error in degrees, but this will entail to an important error, that will cause the complete object not to fit spatially. Henry et al. (2010) propose to use the detection of loop closure as an additional constraint that will describe better the spatial distribution of the frames obtained. Nonetheless, as Weise et al. (2009) explain, this would cause a forced and artificial bending of the object to compensate this error. They on the other hand propose to distribute the error globally, distorting the entire model. This technique is also used by Krainin et all. (2011) when they state:

> To prevent all of the loop closure error from occurring in a single frame when a second connected component comes into view, only one connected component can be matched into each frame. The error is distributed over the whole structure in a separate relaxation step. (p.18)

This technique helps solving the loop closure problematic, without forcing any solution that might be inefficient or unnatural for the object.

Once the data is obtained from the 3D sensors, there is a need to store this information, for treating the image and obtaining useful information when required. This is conformed by both the previously mentioned stage (frame alignment and loop detection) and also the optimization of the data stored to use the memory efficiently. As Hornung et al. (2013) describe, "memory consumption is often the major bottleneck in 3D mapping systems". In their work they describe a method for discretizing the space in same size cubes to represent the scanned area. One of the drawbacks of this method is that there needs to be known previously the size of the volume to be studied and a 3D grid for that size needs to be previously initialized. This implies memory consumption that might be unnecessary, and for outdoor applications or high resolution scans, the amount of memory demanded could be counterproductive. For this project, an outdoor application is not the case. However, a high resolution could be a characteristic desired in the system. Voxels is the name that the authors give to the same size cubes.

Hornung et al. (2013) also compare the voxels method with one of the most common method for 3D mapping: Point clouds. The advantage of voxels over point clouds falls in memory consumption. Also, the point clouds only represent the measured points, and do not allow the system to differentiate unmapped areas from free areas. On the counterpart, voxel representation makes a difference between both conditions. They further explain their OctoMap integrated framework, which functions with octrees very similar than with voxels, while keeping the memory consumption at a minimum level. Their implementation is open source, and is available on the web for several platforms.

Another approach for 3D representation is explained by Henry et al. (2010) with the commonly used surfels. In their own words:

> A surfel consists of a location, a surface orientation, a patch size and a color. As more point clouds are added to the surfel representation, we follow rules… for updating, adding, and removing surfels. Surfels store a measure of confidence, which is increased through being seen from multiple angles over time. Surfels with low confidence are removed from the representation.

With this method, the point clouds are reduced to a more compact representation, preserving the important information while using less space in memory.

Krainin et all. (2011) make a comparison for the use of surfels against the previously used triangle meshes. The triangle meshes provide occlusion information, but when attempting to update a map it can become a more complicated problem, since this method requires maintaining connections between vertices. This is not a problem with the use of surfels since they can be separately updated, and they can be transformed into one mesh when desired. They also mention the benefit of the surfel resolution being updated when the object is closer to the sensor. Another advantage of the surfels that Krainin et all. (2011) mention is that it can use "visibility confidence" which basically means to give a value of confidence to each surfel depending on how many times it has been observed by the sensor and from what angles it was observed. This increases the confidence of the lecture for the system.

Henry et al. (2010) claim that the Iterative Closets Point method (ICP) is commonly used for 3D frame alignment. Valencia et al. (2009) mention non-probabilistic methods to solve the frame alignment problem. They mention the ICP and variations of the ICP to be used for the alignment of two scans. However, they suggest probabilistic methods to be better, since they allow the correct distribution of the loop closure error. They use the delayed-state Extended Information Filter (EIF) algorithm as well as the delayed-state Extended Kalman Filter (EKF).

Developing a system for the given Leuze sensor at the DHBW will need to consider all the different algorithms and techniques commonly used in order to conceive the best balance between resolution of the 3D images obtained given a purpose for the application, cost of operation and cost of the system, and time invested on the scan. All this variables strongly depend on the application in which the system will be used, and the cost the owner of the system is minded to invest.

Despite this application is being developed for investigation, and will not be functioning on a real industry environment, it is desired for the system to be energy and memory efficient, while giving the most efficient behavior. On the future, this type of applications will give robots a more accurate understanding on their surroundings, and with this, accomplish more complicated tasks. As Krainin et all. (2011) mention accurately:

> The goal of our work is to develop techniques that enable robots to autonomously acquire models of unknown objects, thereby increasing their semantic understanding. Ultimately, such a capability will allow robots to actively investigate their environments and learn about objects in an incremental way, adding more and more knowledge over time.

Achieving goals like this will expand the uses of robotics on the coming years exponentially. Hopefully, this will allow robots to move around us as any other human being, and also perform any type of action as we would, like navigating or performing any job to help people with our daily tasks.

In this section, many of the applications for 3D mapping were mentioned, to locate among them the specific application that wants to be developed on this project: an indoor, static application, to build 3D maps of objects present on a specific surface or area, with a light section sensor. Some of the main algorithms were mentioned, like the use of surfels and how to solve the loop closure problem, the use of Octrees to store information, and the improvement on the use of memory when storing the information required. The main variables to have present during the realization of the project were discussed: memory consumption, external light influence, material influence, cost of operation, cost of the system and time spent by the system when scanning an area.

III. Definition of the problem

On the overview section, it will be explained the interest from the involved entity, in this case, the Duale Hochschule Baden-Wurttemberg University (DHBW) located in Karlsruhe, for developing the project that is presented in this document. It will be mentioned the different stages that integrate the overall project, and what is wanted from each of this stages. On the "Problem Synthesis" section, the project will be explained in a simple phrase, that can be easily understood, and that describes the main idea to be developed on the project.

i. Overview

The university Duale Hochschule Baden-Wurttemberg (DHBW) in Karlsruhe, Germany has a Robotics and Automation laboratory that is part of the mechanical engineering faculty, a division of the engineering or "Technics Department".

The DHBW owns a light section sensor from Leuze-Electronic.de. The sensor is currently used by a demonstrator to pick rings lying unsorted in a box. The system is running well, but it is only a demonstrator for these rings. Therefore, other objects can't be handled by the system. They also have possession of one ABB IRB1600, two IRB140 and from UniversalRobots an UR5 and UR10. At the moment the system functions like this: The universal robot uses the light section sensor to make a scan of a box that contains several metal rings. It then picks the rings and stacks them properly.

The project will be divided in several steps that will be explained next:

Step1: Collaborative Robot

Understand the functioning of the collaborative robots available at DHBW, their differences, advantages and disadvantages. Learn on the specification for this robots and how to program them. Understand the robot demonstrator which uses the light section sensor to detect metal rings, pick and stack them.

Step 2: Light section sensor

It is desired to understand how the light section sensor works, its features and limitations and how to get signals from it. The system is working at the moment to detect metal rings; however there is no specific information on how it is done. Several testing will be carried out to understand better the functioning of the system: how it performs the scanning, and what communication protocol it uses to send this information to the receiver. It is also important to understand what process it uses to build the frame alignment to create the 3D images.

Step 3: Object detection, recognition

A first goal is to identify and understand the algorithms used to detect the metal rings at the moment, and compare them, with different methods found in literature. It is desired to develop algorithms to identify different objects with OpenCV. Also, it will be implemented algorithms to identify different objects with Matlab or Octave software, to compare performance.

Step 4: Combined work

The overall objective is to integrate the robotic system, sensor reading algorithms and computer vision system to perform a complete function. The UR will aid rotate the sensor for the initial lecture, the information will be processed by the system and data will be sent back to the UR, so it can identify the different objects present on the work area, decide how to pick up and pick up the objects, for further manipulation (reposition, stack, relocate or store objects)

Step 5: Documentation

As a last step, it is intended to document the process that let the robot handle new or additional objects. A user manual will be written so future investigations can be made and new users will be able to manipulate and/or modify the 3D scanning and manipulating system for different purposes.

Additional considerations need to be mentioned:

The goal of the project is to detect and handle new objects with the 3D scanning system and the collaborative robot. For choosing different objects for the tests, it is desired to pick different and challenging objects that can be handled with one gripper. However, it can be an option to pick also objects that need to use different grippers (fingers, suction cups, among others), which would elevate the complexity of the system because the robot needs to make the decision of which gripper would be more appropriate to use. The objects to handle by the robot will be produced or bought, depending on the requirements. Also, the grippers can be used as supplied by the university, or they can be adapted or designed new if needed to accomplish the objectives of the project.

The motivation for developing such a system is based on the availability of the hardware (light section sensor and universal robot) at the DHBW, but also on the necessity to increase knowledge and uses on this type of systems that use 3D scanning technology. At the moment, the university has a very restricted use for the sensor, but the uses of it can be increased and enhanced by developing further investigation. With the option of scanning multiple objects, the applications also augment. Scanning objects in the given environment for obtaining 3D images can be used for sorting different objects, assembling entire components, augmented reality, service robots and others. The ability to learn about the surroundings and about different objects allows robots to get a better notion of the world, and with this, new applications can be developed. Lai and Fox (2009), for example, synchronized 3D readings with Google 3D Warehouse models. With this, a robot was able to identify real world objects and classify them using as base similar 3D models found on this database. And with the growing use of 3D technologies, understanding the world will be easier every day.

ii. Problem synthesis

To develop an integrated system that is able to scan a surface to get 3D images, and with this information identify objects present and give instructions to a Universal Robot to pick up the objects.

After this section, the reader should have a clearer view of the motivation that led to develop this project, and an overview of the subdivision that will integrate the project. Also, the problem synthesis is a very concise idea of what the project will be about.

IV. Solution approach

This section is a discussion about the problems to be resolved to achieve the goal of the project. It will be presented the situation in which the equipment is found at the moment, and the solution will be divided into different steps that will help to direct the developing of the project in an efficient and ordered solution.

In the last years, 3D technologies have been increasing. Nowadays, it is common to hear about 3D printing, modeling and scanning. They are all separate technologies; however they complement each other, expanding the uses for all of them. Some of the most common uses for 3D scanning are navigation, assembling, telepresence, gaming, augmented reality, and in general terms, interaction of a robot with the world. For this project, the DHBW University wants to expand the uses of a 3D scan that they own, so they can have more complex uses of the sensor, and develop different applications with it.

At the moment, the light section sensor from Leuze-Electronic.de that the university owns has a very restricted use. The sensor is handled by a robotic arm which moves the sensor over a scanning area, in their case, a box. The sensor captures the depth information and transmits the information to a computer that analyzes the information. After getting the data, the system is capable of detecting a specific type of metal rings, pick the rings with the universal robot, and stack these rings. The application is not simple. On the contrary, it shows several engineering challenges; however, the system is only capable of working with the given rings, and cannot operate with any other objects, making the application too restricted, and wasting capabilities of the sensor and the UR.

The DHBW wants the ability of the system to be enhanced. It is desired to have the system functioning as it is at the moment, with the added value of scanning and identifying different type of objects, not only the metal rings. The goal is to experiment with different type of objects that have challenging shapes, and also with different materials. After scanning the work area, the system should be able to identify the different present objects, take decisions on which objects the robot can take and take them. When taking the objects, the final action may vary since this is not a real application and the system might be used for different ends in real life. In this case, the robot might be programmed to pile the objects, or just store them on a different box to demonstrate the capabilities of the robot to interact with the environment and the different objects.

As the system is functional at the moment, it will be used as a reference for improvement. Several tests will be run and documented on the initial condition of the system. Important data that need to be taken into account are: time for scan, overall performance of the system, ability of the system to detect metal rings, ability to accurately locate and pick up rings, influence of external light on the performance, ability to detect rings from different materials (not metal). The goal is to establish these results as a first parameter, and eventually match or surpass the results. However, it will be taken into consideration that the time for achieving proper 3D scans is proportional to the complexity of the objects being scanned.

As it was mentioned in the previous section, the problem consists on several steps that need to be resolved in order to achieve the objective of the project. For a better approach of the investigation, these steps will be implemented sequentially, documenting the resolution on every step, and considering the necessary variables that present on each stage.

Step1: Collaborative Robot

This stage is for getting to know the universal robots available at DHBW. As mentioned before, there are four different models:

- ABB IRB1600
- ABB IRB140
- UniversalRobots UR5
- UniversalRobots UR10

The purpose is to get familiarized with each robot, their capabilities and restrictions, and how they are programmed. One of the models will be selected for the remaining of the project according to their capabilities and easiness for programming, as well as the ability to adapt the sensor to the given robot.

Step 2: Light section sensor

The light section sensor from Leuze-Electronic.de is currently operating. This stage is intended for understanding how the sensor works, at both hardware and software level. It is a main objective to research about the communication protocol that the sensor uses in order to determine if it is the best option for this type of applications, or if there is an alternative for reaching higher speeds for data acquisition. It will be also of interest to determine what technique the sensor uses for getting the depth information on the environment, and what algorithms it implements to merge the various images to create the 3D figure.

Step 3: Vision system: Object detection, recognition

After having created a 3D scan of the work space, the computer will only have one whole 3D image that includes any object over the table and the table itself. The system cannot differentiate between objects, base surface and noise by itself. Further algorithms need to be applied to the resulting image to isolate the objects from the rest. The image needs to be filtered if necessary to reduce sensor noise and the image needs to be conditioned. After obtaining separate objects, they need to pass through a vision system program that identifies each object and classifies them by physical characteristics. Important features to keep in record when classifying the objects would be: the main dimensions of the object, presence or absence of outstanding features (like handles), and if the object was detected as a solid or hollow object. This information will be helpful when making decisions for handling each object. The system should also have information of the gripper that is being used to determine if the robot can handle a certain object with the tool available, or if a different tool would be necessary for an action to be taken. If a different tool is available that would suffice the needs to handle the object, the robot should change the tool, but if there is no tool that will meet this requirements, the system should inform the user about the situation, and identify the object that leads to an exception.

Step 4: Combined work

The overall objective is to integrate the robotic system, sensor reading algorithms and computer vision system to perform a complete process. The UR will aid move the sensor for the initial lecture, the information will be processed by the system and data will be sent back to the UR, so it can identify the different objects present on the work area, decide how to pick up and pick up the objects, for further manipulation (reposition, stack, relocate or store objects) as it is desired or needed (the final action depends on the application that is given to the system and the specific company's needs). In this final step, the mechatronic abilities become more evident, because there is a need to integrate mechanics from the UR, the communication stage between sensor, computer and robot, electrical knowledge for both sensor and robot, control for the complete system and computer vision for the analysis and interpretation of the information acquired. All these steps should have the other steps in consideration when developing each one of them, since it is desired to have an integrated system, and not a segmented system that may present conflict with previous or further steps.

Step 5: Documentation

As an important part of the project, all the development will be documented on a report parallel to the progress, for each development achieved on every stage of the project. This will be to procure order when performing the project, and to allow a high standard when developing the project, as it is expected from an engineer from the Instituto Tecnológico de Costa Rica.

It is presented below a diagram that contains each of the variables that will be considered on each stage to determine the best solution possible. Every variable will be measured and compared with other methods, or devices when available.

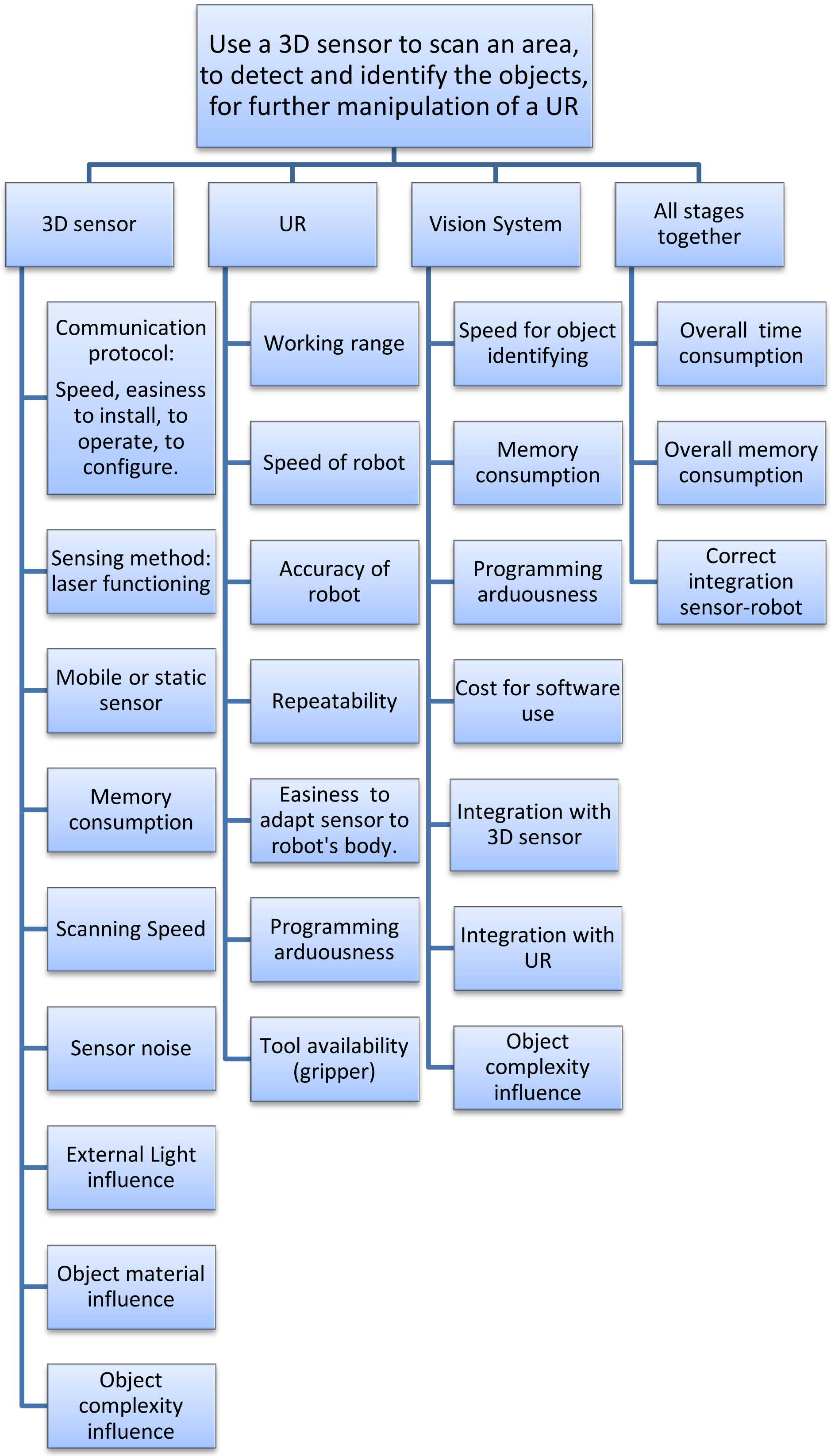

Use a 3D sensor to scan an area, to detect and identify the objects, for further manipulation of a UR
3D sensor
UR
Vision System
All stages together
Communication protocol: Speed, easiness to install, to operate, to configure.
Sensing method: laser functioning
Mobile or static sensor
Memory consumption
Scanning Speed
Sensor noise
External Light influence
Object material influence
Object complexity influence
Working range
Speed of robot
Accuracy of robot
Repeatability
Easiness to adapt sensor to robot's body.
Programming arduousness
Tool availability (gripper)
Speed for object identifying
Memory consumption
Programming arduousness
Cost for software use
Integration with 3D sensor
Integration with UR
Object complexity influence
Overall time consumption
Overall memory consumption
Correct integration sensor-robot

Figure 1 Variables to consider during realization of the project

On the previous section, it was presented several steps that need to be followed to achieve a proper solution for the project. Also, on the Figure 1 was given an overview of the variables that need to be taken into consideration. These variables will be studied during the development of the project, and well documented to understand the functioning of the system, and how the environment affects the implementation of the system.

V. Cause – Effect Diagrams (Ishikawa)

On this section it is presented three diagrams that name different causes for certain problems. The problems cited were identified as the situations that may present during the development of the project, and hence need to be premeditated, to avoid or reduce their effect for the resulting system.

Three main problems were identified that may present on the developing of the project. The Ishikawa diagram that links together cause and effect for each problem is showed below:

Problem 1: System is not able to recognize objects

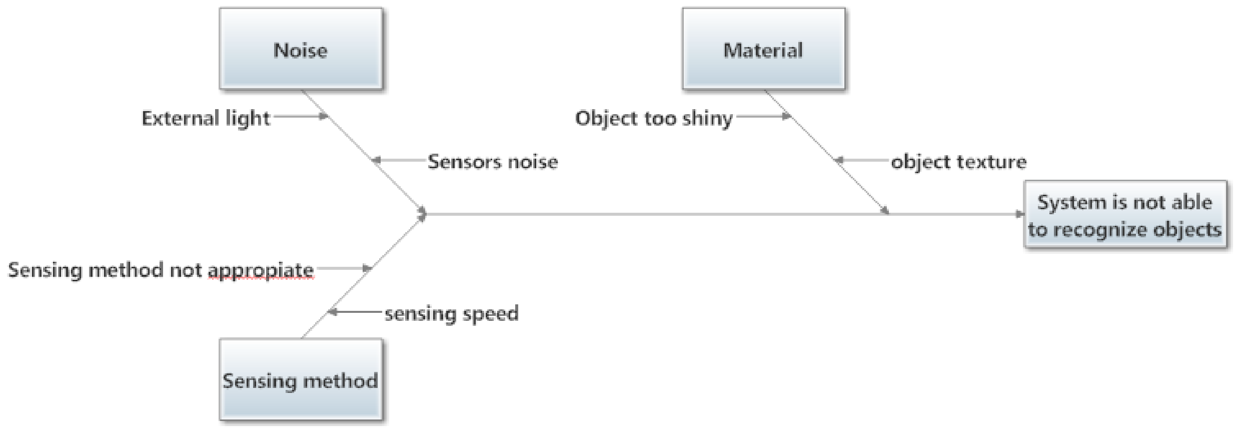


Figure 2 Problem 1

Problem 2: System takes too much time to scan and recognize objects

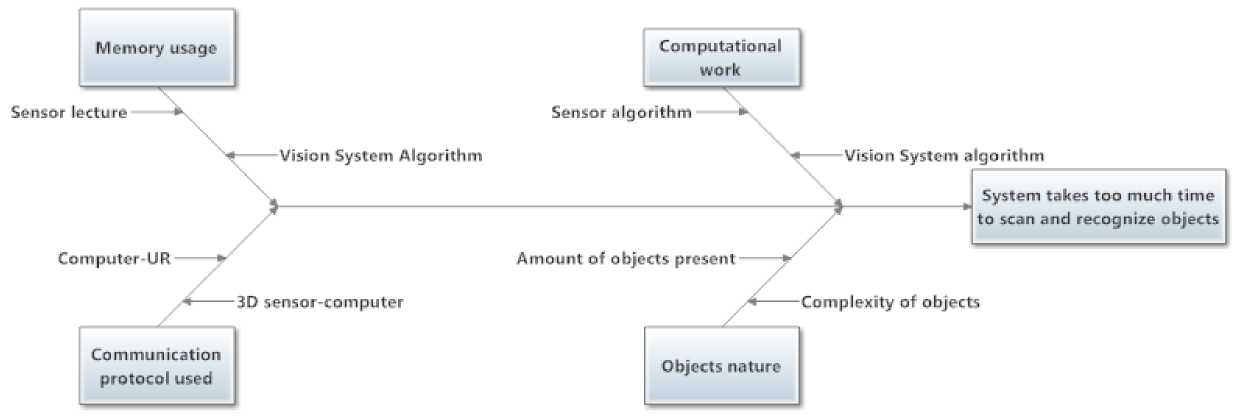


Figure 3 Problem 2

Problem 3: Robot is not able to grab objects accurately

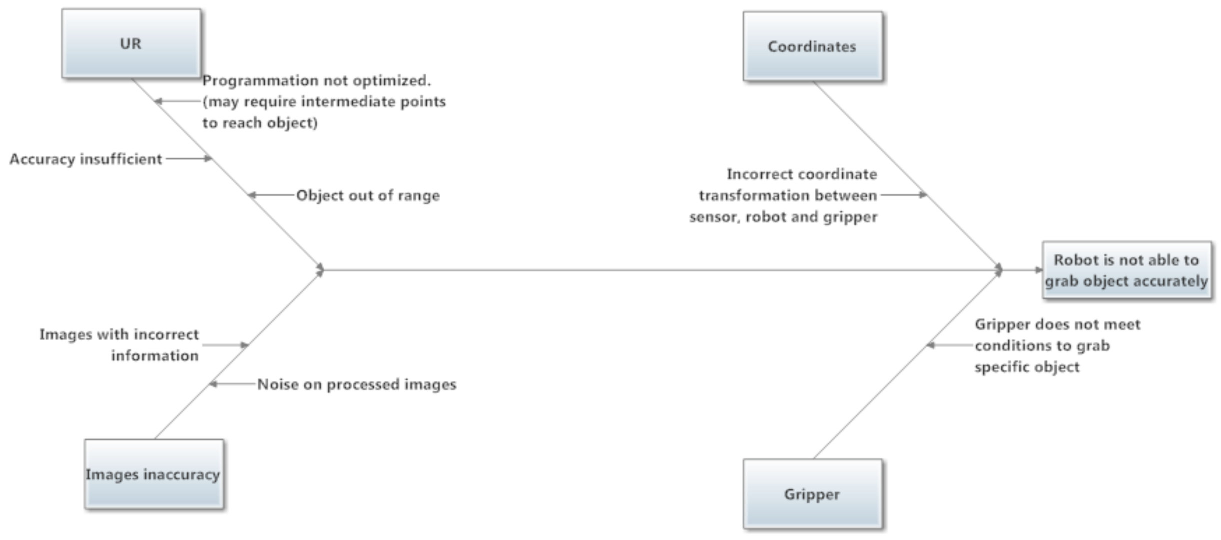

UR
Coordinates
Programmation not optimized. (may require intermediate points to reach object)
Accuracy insufficient
Object out of range
Incorrect coordinate transformation between sensor, robot and gripper
Robot is not able to grab object accurately
Images with incorrect information
Noise on processed images
Gripper does not meet conditions to grab specific object
Images inaccuracy
Gripper

Figure 4 Problem 3

On the diagrams showed previously, it was presented different variables that will be taken into consideration during the performance of the project. They will guide the solution desired, to accomplish specific goals.

VI. Objectives

On this section the general and specific objectives of the project will be presented. They will be considered the spine of the project among the process, and will guide the overall procedure and goals to be obtained. For each objective, it is given a specific indicator that makes easy to identify if the goal was properly achieved, or which points should be enhaced.

i. General Objective

To develop an integrated, low cost system with a 3D sensor, Vision Systems and a Collaborative Robot, that allows the system to identify objects on a given work area, for further manipulation of the Robot.

Indicator: The system is able to identify a series of different object using the 3D sensor from Leuze Electronic.de, and the Universal Robot retrieves the information necessary to grab the objects that it is supposed to grab, when the required conditions are met.

ii. Specific Objectives

- Implement the use of a light section sensor from Leuze Electronic.de to acquire 3D images.

Indicator: Visual indicator. The 3D information is sent to a computer, and it is visually possible (on screen) to identify the objects that were present on the work area.

- Implement a vision system environment that is capable of treating information acquired from the light section sensor to identify objects present on the work station.

Indicator: The system will identify with visual markers (on screen) the different objects found on the working area. Also, characteristic features from the objects should also be identified, for further usage and manipulation (dimensions, special features present like handles, etc.)

- Program the Universal Robot to handle de light section sensor to scan the desired area.

Indicator: The robot moves the sensor over the scanning area, so that the sensor is capable to realize the needed scan procedure.

- Implement a code that allows the Universal Robot to interpret the resulting information and grab the different present objects. (Optional as a further objective to achieve).

Indicator: The robot interprets correctly the information resulting from the Vision System Stage, and is capable of identifying which objects it can take, which objects it should take, and take the objects.

The general objective presented above is the main goal of the project. The specific objectives are previous goals that need to be accomplished for fulfilling the main objective. After having specified the objectives of the project, it is easy to have a clearer view of the steps that need to be taken.

VII. Project execution procedure

On this section, it will be presented a more detailed list of steps in which the solution of the problem was divided. The main project is simplified into more specific tasks, in order to achieve a better solution. The project will be divided in three main stages and one final integration stage.

The main stages are:

Stage 1: Light section sensor from Leuze Electronic.de functioning
Stage 2: Universal Robot programming
Stage 3: Vision System developing

And the final stage would be:

Stage 4: System integration for the 3D sensor, the Universal Robot and the Vision System for identifying and further grabbing objects.

Each stage will be explained below:

1. Light section sensor from Leuze Electronic.de functioning

This task will have the following steps:

a. Recognize de sensor and read the Technical Documentation available for the light section sensor from Leuze Electronic.de.
b. Research on possibilities for programming and establishing communication with the sensor. Make simple tests with the sensor on place.
c. Build a simplified prototype to simulate the environment where the sensor will be operating (structure for holding sensor and electronics) and man operated to simulate universal robot.
d. Implement communication with the sensor with interface for the user.

    e. Develop tests with different objects, varying the previously discussed variables (color, external light, shape and others) and document results.
    f. Study results and improve performance.

2. Universal Robot programming
    a. Recognize URs available and read Technical Documentation. Select the UR to use according to its characteristics.
    b. Develop different programs to understand the capabilities of the UR.
    c. Program the UR to move the sensor according to the process defined on the stage for the Light section sensor.
    d. Compare results from the ones obtained on Stage 1, and improve performance in both programming of the Light section sensor and the UR if necessary.

3. Vision System developing
    a. Gather results obtained from Stage 2.
    b. Study different programming languages that could solve the problem (At the moment it's in mind to use MatLab/Octave and Open CV).
    c. Develop Vision System to evaluate results and obtain the information needed. The solution may be developed with several programming languages, to compare performance.
    d. Compare results obtained with the different codes, and improve the programming.
4. System integration for the 3D sensor, the Universal Robot and the Vision System for identifying and grabbing objects.
    a. Study results obtained from Stage 3.
    b. Develop code to communicate information obtained by the Vision System to the Universal Robot.
    c. Integrate results from all 4 stages.
    d. Develop tests to check functionality.
    e. Study results and improve the whole system.

After having a list of detailed steps to be followed, the time available for the development of the project should be distributed. This will be done in the next section.

VIII. Chronogram for activities.

In this section, it will be presented an estimated distribution of time for developing each of the previously mentioned tasks, in order to achieve the objectives of the project. It will be used a software tool for developing Gantt diagrams to have a visual distribution of the tasks among time.

To achieve a successful accomplishment of the project, the rationing of all resources is elemental, and this includes the correct use of time available. Specially, since the project is being developed in a foreign country, with limited resources, a correct distribution of tasks among time is essential. It is presented below a table and a Gantt diagram with an estimated planning for the realization of the previously mentioned tasks.

Table 1 Tasks chronogram

Task	Task Mode	Task Name	Duration	Start	Finish	Predecessors
1	Auto Scheduled	Stage 1.a Recognize sensor and read Technical Documentation	3 days	Mon 26/10/15 08:00 a.m.	Wed 28/10/15 05:00 p.m.	
2	Auto Scheduled	Stage 1.b Research programming options. Tests to sensor	4 days	Thu 29/10/15 08:00 a.m.	Tue 03/11/15 05:00 p.m.	1
3	Auto Scheduled	Stage 1.c Build simplified prototype for simulation	2 days	Thu 29/10/15 08:00 a.m.	Fri 30/10/15 05:00 p.m.	1
4	Auto Scheduled	Stage 1.d Implement communication with sensor. User interface.	10 days	Wed 04/11/15 08:00 a.m.	Tue 17/11/15 05:00 p.m.	2
5	Auto Scheduled	Stage 1.e Develop tests	2 days	Wed 18/11/15 08:00 a.m.	Thu 19/11/15 05:00 p.m.	4,3
6	Auto Scheduled	Stage 1.f Evaluate and improve	3 days	Fri 20/11/15 08:00 a.m.	Tue 24/11/15 05:00 p.m.	5

Table 2 Tasks chronogram (Continuation)

7	Auto Scheduled	Stage 2.a UR Technical Documentation	2 days	Wed 25/11/15 08:00 a.m.	Thu 26/11/15 05:00 p.m.	6
8	Auto Scheduled	Stage 2.b Develop test programs for UR	2 days	Fri 27/11/15 08:00 a.m.	Mon 30/11/15 05:00 p.m.	7
9	Auto Scheduled	Stage 2.c Program UR to move sensor	3 days	Mon 04/01/16 08:00 a.m.	Wed 06/01/16 05:00 p.m.	8
10	Auto Scheduled	Stage 2.d Compare and improve	4 days	Thu 07/01/16 08:00 a.m.	Tue 12/01/16 05:00 p.m.	9
11	Auto Scheduled	Stage 3.a Gather results from stage 2	1 day	Wed 13/01/16 08:00 a.m.	Wed 13/01/16 05:00 p.m.	10
12	Auto Scheduled	Stage 3.b Research programming languages for Vision System	2 days	Thu 14/01/16 08:00 a.m.	Fri 15/01/16 05:00 p.m.	11
13	Auto Scheduled	Stage 3.c Develop Vision System	12 days	Mon 18/01/16 08:00 a.m.	Tue 02/02/16 05:00 p.m.	12
14	Auto Scheduled	Stage 3.d Compare and improve	4 days	Wed 03/02/16 08:00 a.m.	Mon 08/02/16 05:00 p.m.	13
15	Auto Scheduled	Stage 4.a Study results from stage 3	1 day	Tue 09/02/16 08:00 a.m.	Tue 09/02/16 05:00 p.m.	14
16	Auto Scheduled	Stage 4.b Develop UR code	4 days	Wed 10/02/16 08:00 a.m.	Mon 15/02/16 05:00 p.m.	15
17	Auto Scheduled	Stage 4.c Integrate results	2 days	Tue 16/02/16 08:00 a.m.	Wed 17/02/16 05:00 p.m.	16
18	Auto Scheduled	Stage 4.d Tests to check functionality	4 days	Thu 18/02/16 08:00 a.m.	Tue 23/02/16 05:00 p.m.	17
19	Auto Scheduled	Stage 4.e Study results and improve	7 days	Wed 24/02/16 08:00 a.m.	Thu 03/03/16 05:00 p.m.	18

Notes:

The start day will be on Monday, October 26, 2015.

The vacation period is from December 1st to December 31st, 2015.

The estimated date to finish the project is on Thursday, March 3, 2016.

i. Gantt diagram for tasks

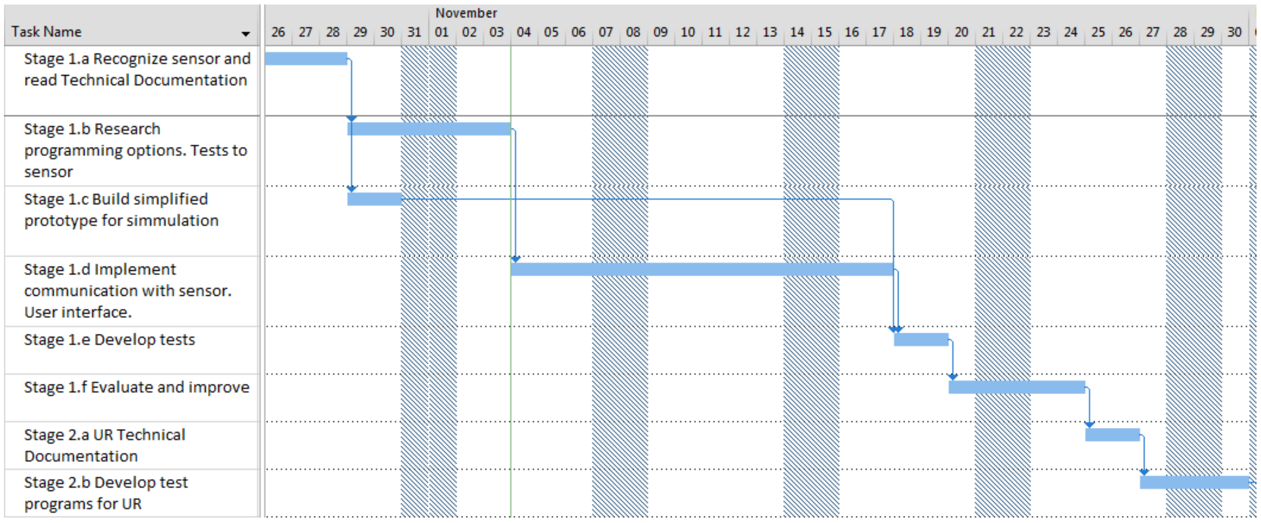


Figure 5 Chronogram for November

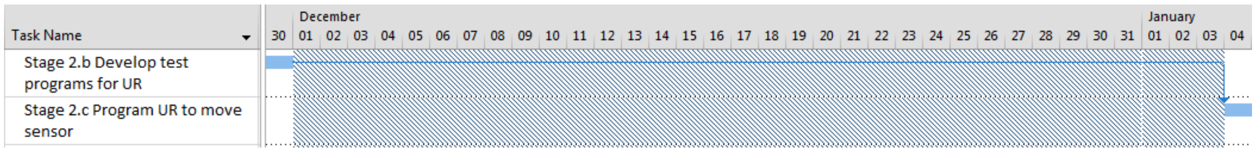


Figure 6 Chronogram for December

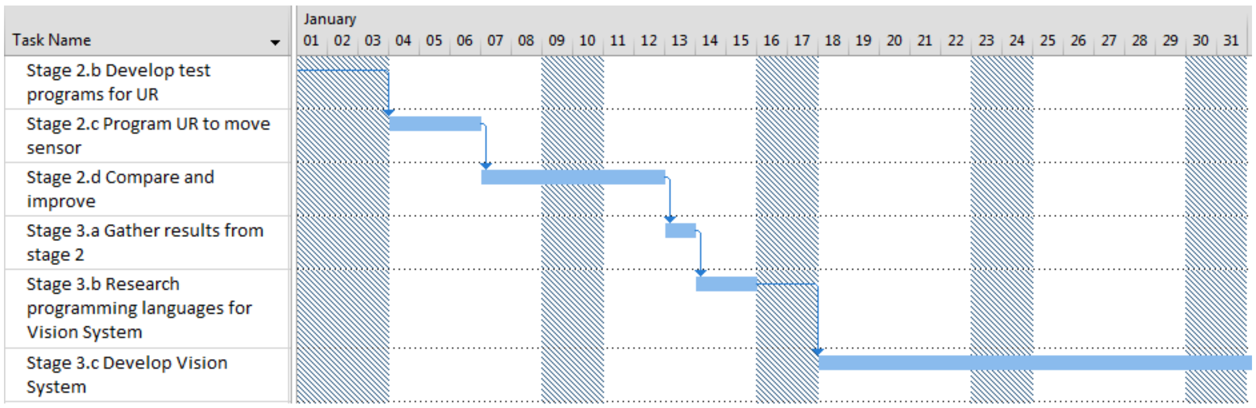


Figure 7 Chronogram for January

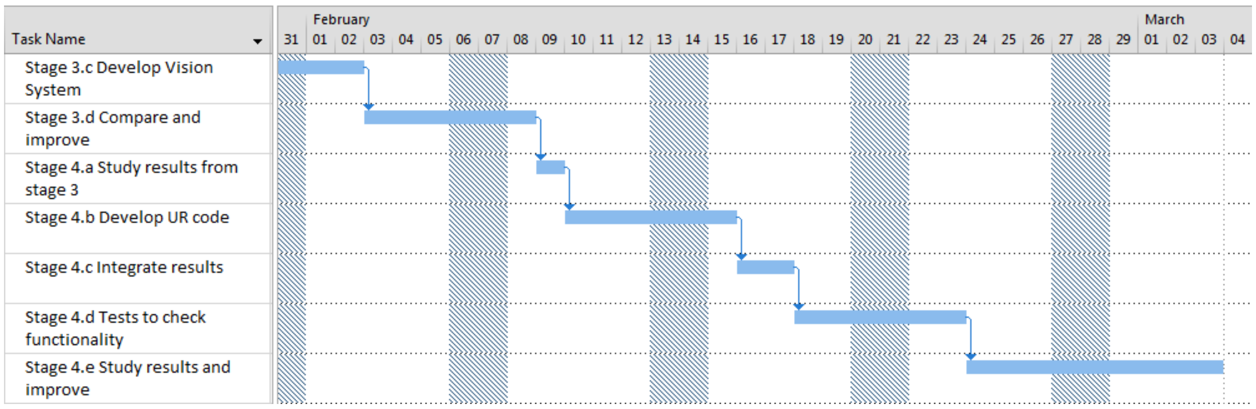


Figure 8 Chronogram for February and March

Note:

Non-working days and vacation period are shown in bold on the Gantt diagram.

On this section was presented an estimated distribution of the tasks to be developed. Each task was given a certain time to be completed, estimating the difficulty of the task. The chronogram here presented is a guide for developing the project, and is intended to easily identify when a task gives opportunity to advance faster, or on the contrary, when a specific task will be taking more time than estimated, to make previous adjustments.

IX. Budget and expenses for the project

On this section, it will be presented the known expenses that are present for the developing of the project. The information was given by Professor Dr. Clemens Reitze from the DHBW, and is presented here to calculate a normal investment when intending to develop a project such as the one desired. If there are any other expenses that may present further, they will be included to the total list.

The project to be developed has specific costs that are expected: the cost for the Universal Robot, the light section sensor from Leuze Electronic.de, the computer to operate both sensor and UR, and others. While the DHBW from Karlsruhe already owns the UR to be used and the 3D sensor, it is convenient to mention the investment made by the university, so that a third party can visualize a budget needed to reproduce a project like the one to be developed.

Hence, a breakdown of the outgoings is presented next:

Table 3 Lists of costs

Item	Price (€)
UR10	23870.00
UR5	17870.00
Gripper	1378.00
2D Image Recognition	8571.00
3D Image Recognition	9714.00
Fittings, small parts	2000.00
Total	63403.00

Notes:

1) The "Image Recognition" prices are for hardware and software:

- Leuze Sensor (3D)
- CCD Camera (2D + 3D)
- IR PC with touchscreen
- Halcoon library, runtime license
- FIPS IR Software from Faude.de

2) The budget presented here is for all the inversion the university DHBW has made for developing the project, among other projects that use the same parts. It is not the real cost for developing a project, because the final project will not necessarily use all the components listed before. A complete analysis of costs will be detailed when the project is completed.

X. Solution's detailed description

In this section, a detailed step-by-step description of the solution will be explained. As it was described before, the execution of the project is divided in 3 main phases due to its complexity:

i. Light section sensor
ii. Object detection and recognition
iii. Universal Robot

And a final fourth stage which integrates the first three, which was named "Combined work" stage.

Hence, the description of the solution will also be sectioned in these phases, and a final integration of all of them. The solutions will be presented in the order they were developed, but first, a general investigation of communication protocols and definitions will be presented, which will be used for both phase 1 and 3 of the solution.

i. Selection of the Robotic Arm

The DHBW University has for different robotic arms in the Robotics and Automation Laboratory:

- ABB IRB1600
- ABB IRB140
- UR5
- UR10

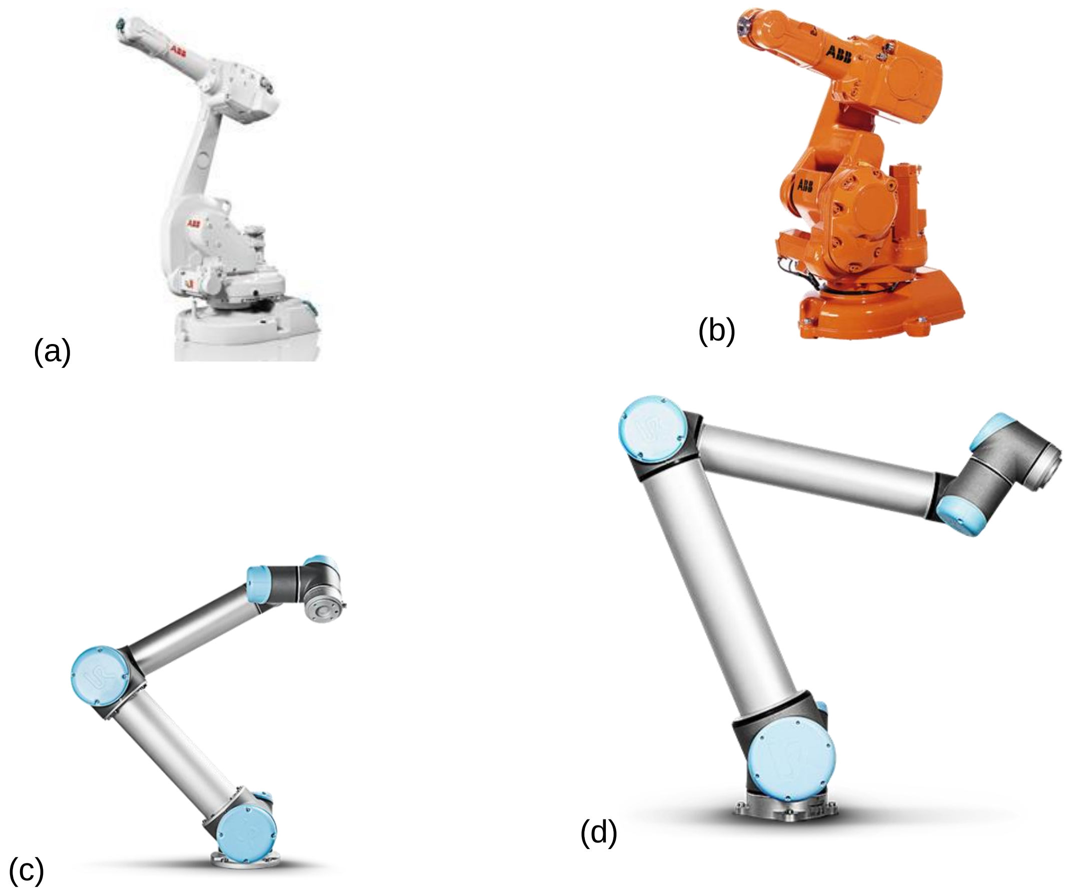


Figure 9 Robotic arms at DHBW: (a) ABB IRB1600. (b) ABB IRB140. (c) UR5. (d) UR10.

The ABB robots where at the moment used for different projects from other research students, and both Universal Robots where available for immediate use. The difference between both UR5 and UR10 are in dimensions and capacity, where the UR10 is bigger and can handle more weight. The programming of both robots makes no difference since they run on the same language and environment, and the final decision of which UR would be the best choice depends on the specific application the robot will be developing and its requirements. Since the application developed is for research only (not a real application in the industry), and the parts that will be handled by the robot represent no high demand on power, the UR5 was selected for working on the developing of the project during the semester. However, the solution developed will be possible to use on both URs.

After selecting the Robotic Arm that will be used for the development of the project, a brief research on the communication protocols that can be used for achieving the objectives will be presented.

ii. Communication (Transport) Protocols

IP Communication provides the ability to send and receive information between different devices, for a variety of purposes. It is an efficient service model when used properly. The idea of implementing a network with a certain protocol is to hide to the end user the physical network, and the technologies and architecture used, while securing a reliable and complete communication (Chunyan Fu). To understand better the protocols available, and select the best ones for the given applications to develop, a further investigation will be presented.

A communication protocol is composed by the client and the server. They can also be named as "Host 1" and "Host 2" if a differentiation between both is not needed. The client is the host that requests to receive certain information. On the counterpart, the server is the center of the information, where the message or data is stored. After the client requests information, the server will follow some specific steps to send a response to the other side. The steps to follow depend on the specific protocol that is used for the communication. The most common protocols are the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP). Both have different characteristics that could be an advantage or disadvantage depending on the application. Hence, both will be presented next with their characteristics.

1. UDP Protocol

The User Datagram Protocol is an unreliable but fast datagram service (Walrand and Chunyan Fu). The protocol does not establish a connection between server and client. There is no confirmation of received packages, and the packages may be lost, delayed or received incorrect. It is commonly used in applications where the need of precision in the message is not drastic, for example video or audio streaming. The advantage of UDP over TCP is that it does not increase the delay over IP. However, any conflict with lost packets must be resolved by the application itself. There is no buffer on any side of the communication (sender or receiver). Finally, there is a limit for the size of the messages sent and received.

2. TCP Protocol

The Transmission Control Protocol is widely used in many IP communication applications. It is not as fast as UDP. However, it is more reliable with the information exchanged between hosts. A connection is established, which allows a flow control of the packets using a buffer and handling errors that may occur when sending messages, for example: lost packets, delayed packets and duplicated packets. On this protocol, a specific port is linked to only one application. Also, it is possible to handle different ports by the same application, when there is need to handle communications with different hosts simultaneously. The size of the messages can be arbitrary.

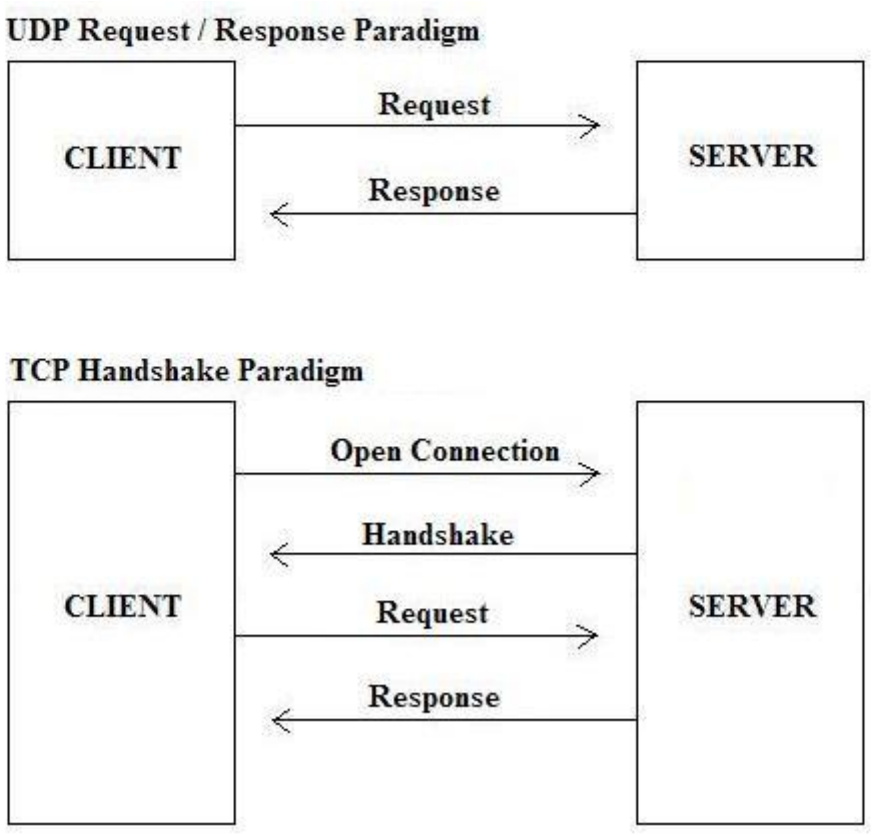


Figure 10 Image from: National Instruments. www.ni.com

As it is showed in the image, perhaps the main difference between UDP and TCP is that while UDP is connection-less and the server just sends the information when it is requested, the TCP establishes a connection, which allows the server to ensure that the client received the packet as expected, and if an error occurred, resend the packet until it is received properly.

For the development of this project, there are two different Ethernet communications needed. The first one is between the Leuze sensor and the central computer, where the laser sensor functions as the server and the PC as the client. The second communication is between the PC and the UR Robot, where the UR behaves as a client and the PC will be working as a server. A diagram is presented below:

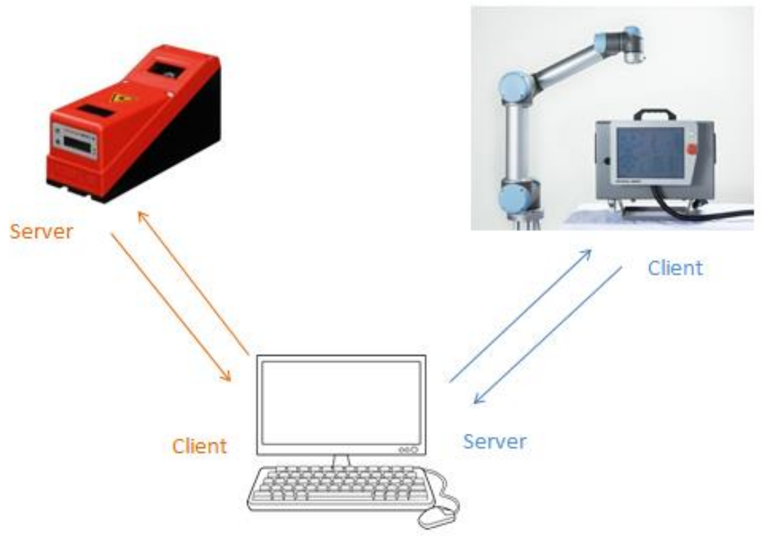


Figure 11 Communication diagram

The next diagrams represent the main differences between a UDP and a TCP connection:

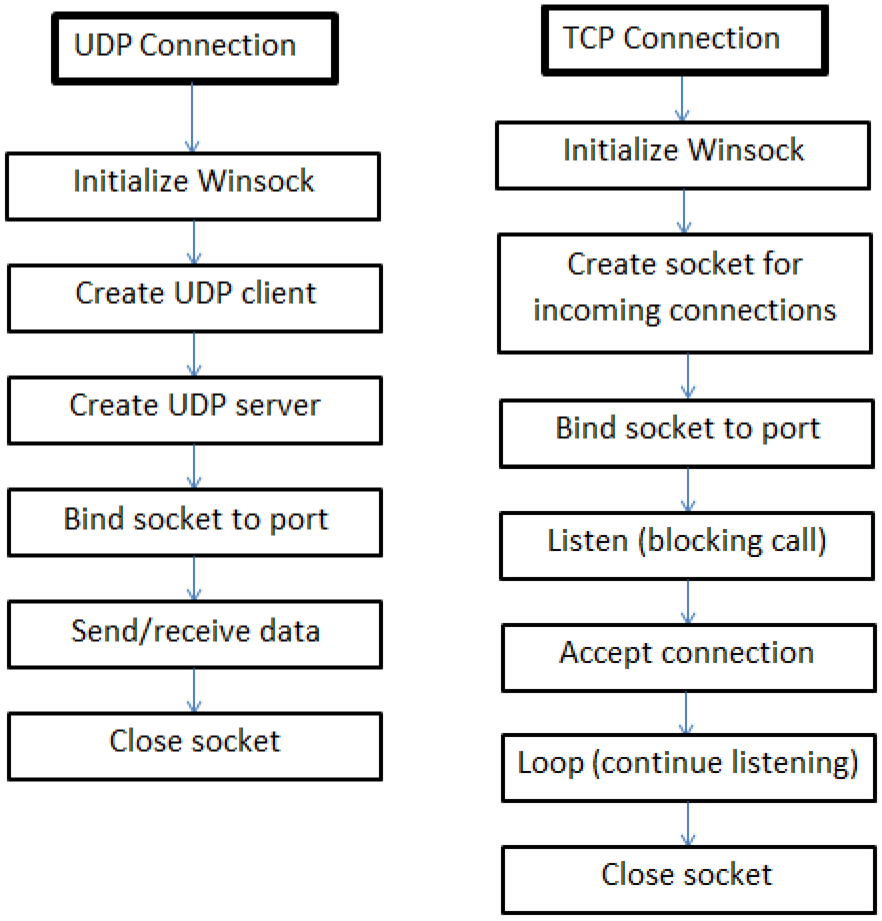


Figure 12 UDP and TCP Diagrams

To understand the previous diagrams better, an explanation of the different phases will be described below.

Both UDP and TCP start by initializing Winsock, which is the use of Windows Sockets DLL. This enables the use of other calls on Winsock, and also ensures that the system supports it. After initialized, the next step is to create the sockets that will be used.

On UDP, there must be created sockets for both sender and receiver. Since there will not be any established connection, every packet sent to a determined socket should be addressed to a previously created socket, with known IP address and port number. On the counterpart, the TCP needs to create only a socket for incoming connections. The bind phase is the one that ties in the IP addresses and Port numbers to a previously created but empty socket. This is for both UDP and TCP communications.

The UDP communication is at this point ready for exchanging information. Any packet sent from the server to the socket linked to the client will be addressed to the known IP and port, and vice versa. However, if there is any conflict on the communication, for some examples: one of the ends gets unexpectedly disconnected, the port number is incorrect, one of the ends is currently busy transmitting or receiving data. In any of these and other cases, the message will be simply sent by the sender, but the receiver will not take in the packet. The sender will neither get a report of the failure, causing lack of communication and lost information. To avoid this, the program should have an error control on both sides.

After the TCP socket has performed the "Bind" function, it is time for the server to start listening. The Listen function is a blocking call. A blocking call is a function that, when executed, remains at that point of the program until a certain value is returned, or a time-out is reached. On the specific case of the Listen function, the program of the server will remain waiting to receive a message from another host (a request from the client). It will remain listening as far as the time-out variable is not reached, after this it will return an error value. However, if a request from the client is received, the program continues its course, and runs the Accept function. At this point, a connection is made between server and client. After a connection is successful, both client and server can send and receive messages to the other part.

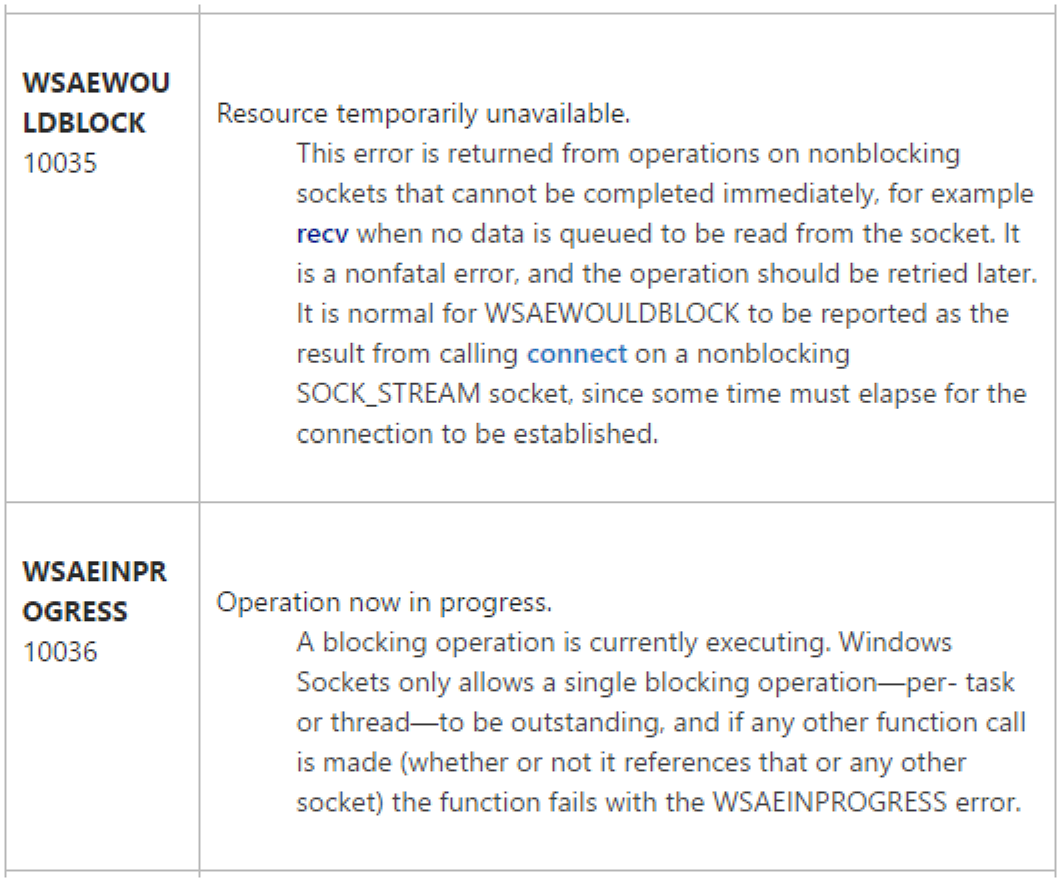
The TCP protocol uses a buffer; this allows sending messages with no restriction on size. The protocol itself makes a revision of the information received, and if a message was lost on transit, or received incorrect, the message will be requested again to the counterpart. Also, if there is a saturation of packets on the receiver, the buffer will store the data and deliver it accordingly.

At the end of both UDP and TCP, the sockets must be closed, to avoid further conflicts.

When deciding which protocol is better for each communication needed, it is clear that the TCP would be preferable, since it assures the correct sending of the packets, and therefore no information will be lost on the process. However, when making a revision of the technical information of the Leuze sensor, it is described that "the LPS communicates with the process control via UDP/IP" (Leuze Electronic, 2014), hence UDP protocol will be implemented for the communication between the LPS and the computer. The UR presents no restrictions for the IP communication, so the TCP will be implemented between the UR and the computer.

3. Error Codes for Socket Programming

When programming for socket communications, the WSAGetLastError() function returns to the user the last error that occurred when executing a WSA function, such as the ones previously explained. An extract of a couple of error codes is shown below.



WSAEWOULDBLOCK 10035	Resource temporarily unavailable. This error is returned from operations on nonblocking sockets that cannot be completed immediately, for example recv when no data is queued to be read from the socket. It is a nonfatal error, and the operation should be retried later. It is normal for WSAEWOULDBLOCK to be reported as the result from calling connect on a nonblocking SOCK_STREAM socket, since some time must elapse for the connection to be established.
WSAEINPROGRESS 10036	Operation now in progress. A blocking operation is currently executing. Windows Sockets only allows a single blocking operation—per- task or thread—to be outstanding, and if any other function call is made (whether or not it references that or any other socket) the function fails with the WSAEINPROGRESS error.

Figure 13 Error codes for Socket Programming

A complete table of errors can be found at MSDN Microsoft (n.d.) and the interpretation of the error depends on the code that is returned by the function. For example, if the code received is 10035, referring to the error "WSAEWOULDBLOCK", it means that the function used at that point of the program would try to remain expecting a reply, while the socket itself is not capable of doing that, if it was configured as a non-blocking socket. The example given for the recv() function on this error needs to be taken on account, since this function will be used for the socket communications amply.

iii. Setting up the programming environment

On this section, different software for developing the project will be mentioned. This first stage of the project is important, since it will constitute the base for developing the whole project. It is also important to have in mind that from the DHBW University, it is desirable to create portable software that will allow the use of the software developed in this project with different Operative Systems, or easily make the conversion to a desired OS. It is specially desired to have the ability to run software on Windows and Linux OS, as they are the most common.

1. Programming language

The software to be developed will be programmed in C or C++ language, depending on the needs of each program. Both languages are easily adapted for functioning on Windows and Linux, as the program can define the OS at the beginning of the program, and include the necessary libraries for its correct functioning.

2. Text editor

For writing the code, any text editor can be used, for example Eclipse or ConTEXT. For this project, ConTEXT was selected for its friendly environment to the programmer; however this does not affect in any way the program itself as it is only a tool for writing and comprehending the code easier.

3. Builder and compiler

The program MSYS 2015 will be used for building the applications and programs needed for the project. It is basically a supplement of MinGW, and has certain advantages over cmd shell. It is "a collection of GNU utilities such as bash, make, gawk and grep to allow building of applications and programs which depend on traditionally UNIX tools to be present" (jonY, 2008). For this project, the most useful utility will be "make". After installing MSYS, it is important to add it to the Path variable, for its proper functioning.

4. Makefile

The makefile is a generic file created to give instructions to Msys on how to build a program. There exist a wide variety of commands for these makefiles. A tutorial on how to construct these files can be found at GNU (n.d.). The resultant makefiles will be also attached at the appendix D of the document. This file basically indicates to Msys where to find the libraries to link, what name to confer to the object and executable, and also indicates the dependencies between files.

There are also tools that assist on creating these makefiles, for example Cmake. This software will be discussed later, when working on the Image recognition phase of the project.

At this point, the user has set up the environment for programming and has an appropriate makefile for creating executable programs. On the next sections, a detailed explanation of the process of programing the three main phases of the project will be explained. These phases are:

- Programming the Light Section Sensor Communication
- Programming the UR Communication
- Programming the Vision Phase

iv. Programming Light Section Sensor Communication

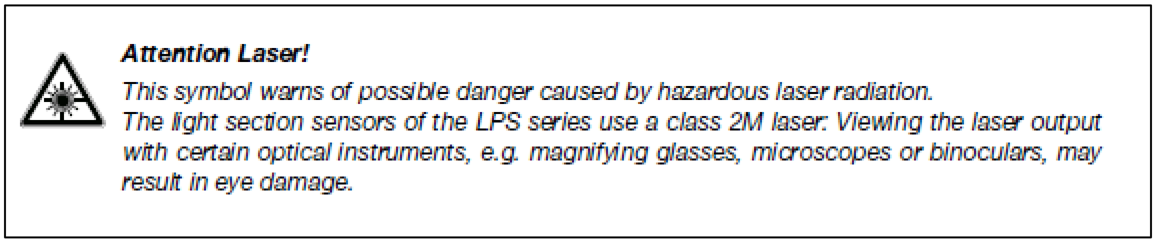
On this section, it will be developed the steps taken for programming the communication between the Leuze Light Section Sensor and the host computer. To achieve this communication, the sensor must first be studied, its technical characteristics, the protocol used by Leuze for the communication and the different possibilities that the sensor offers to the user, to define the scope and flexibility that the sensor will offer for the desired objectives.

Important information about the Light Section Sensor is summarized in the following section.

1. Important safety indications

Before starting to work with the Leuze sensor, it is important to first understand the safety measures and indications for handling the item.

On the next image it is explained the symbol used on the technical description provided by Leuze, and also present on the sensor shield. It is important to respect the indications. Failure to follow these instructions may lead to damage.



Attention Laser!

This symbol warns of possible danger caused by hazardous laser radiation.
The light section sensors of the LPS series use a class 2M laser: Viewing the laser output with certain optical instruments, e.g. magnifying glasses, microscopes or binoculars, may result in eye damage.

Figure 14 Laser warning

Approved purpose of the sensor

Common uses for the light section sensor are for example (but not restricted to):

– 3D Measurement of moving objects: for example in a transporting band
– Gripper control: location of the gripper with real coordinates in working area

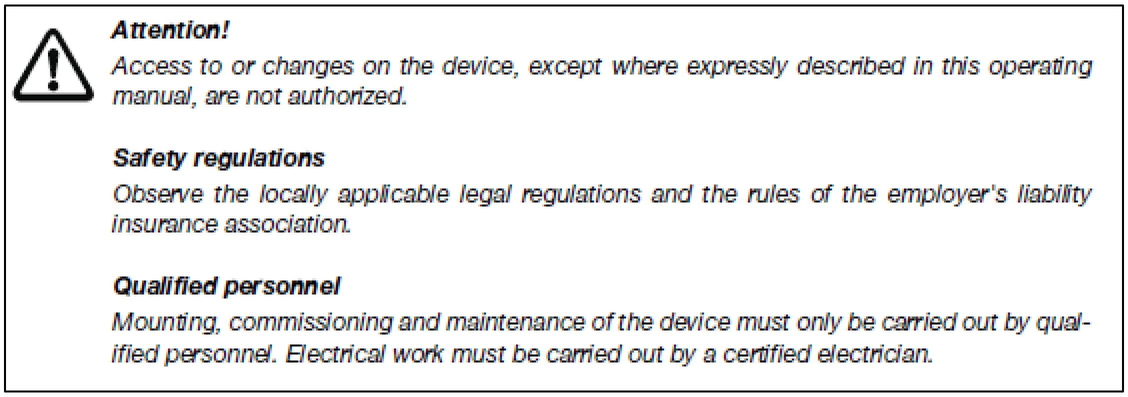
On the counter part, completely forbidden uses include:

– Rooms with explosive atmospheres
– Operation for medical purposes

For the project, the intended use is for both gripper control and 3D measurement. The difference in regard to the given example is that the objects will not be moving, the sensor itself will be moved by the robotic arm assistant, allowing the program to make a 3D scan instead of receiving only a 2D profile.

Leuze Electronic's safety regulations

It is important to read carefully and always follow the indications given by the sensor's manufacturer. These indications should always be accessible by the user. The indications are presented next:



Attention!

Access to or changes on the device, except where expressly described in this operating manual, are not authorized.

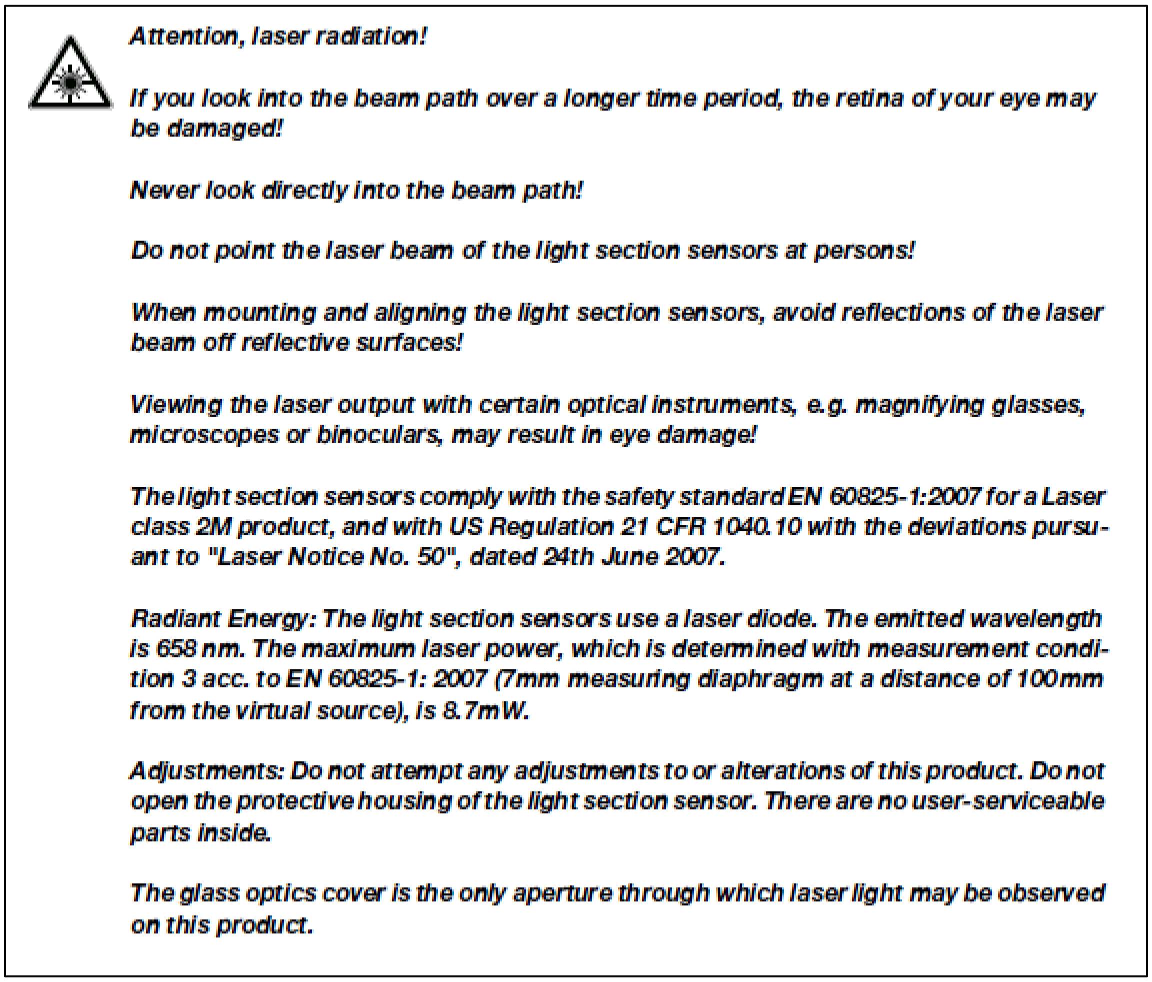
Safety regulations

Observe the locally applicable legal regulations and the rules of the employer's liability insurance association.

Qualified personnel

Mounting, commissioning and maintenance of the device must only be carried out by qualified personnel. Electrical work must be carried out by a certified electrician.

Figure 15 Laser prevention instructions Part 1



Attention, laser radiation!

If you look into the beam path over a longer time period, the retina of your eye may be damaged!

Never look directly into the beam path!

Do not point the laser beam of the light section sensors at persons!

When mounting and aligning the light section sensors, avoid reflections of the laser beam off reflective surfaces!

Viewing the laser output with certain optical instruments, e.g. magnifying glasses, microscopes or binoculars, may result in eye damage!

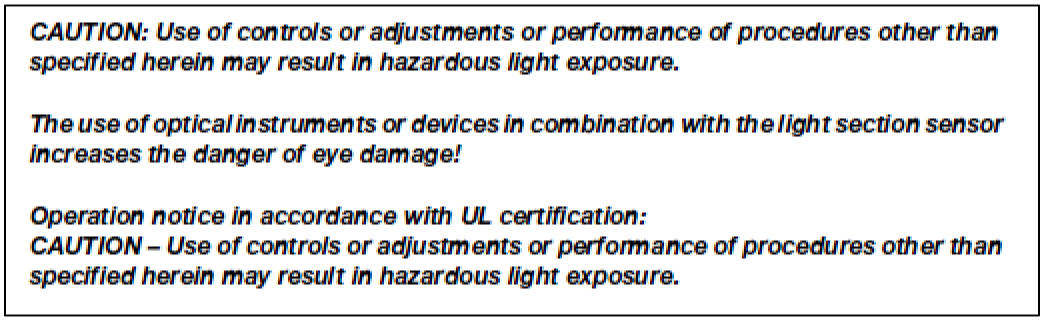
The light section sensors comply with the safety standard EN 60825-1:2007 for a Laser class 2M product, and with US Regulation 21 CFR 1040.10 with the deviations pursuant to "Laser Notice No. 50", dated 24th June 2007.

Radiant Energy: The light section sensors use a laser diode. The emitted wavelength is 658 nm. The maximum laser power, which is determined with measurement condition 3 acc. to EN 60825-1: 2007 (7mm measuring diaphragm at a distance of 100mm from the virtual source), is 8.7mW.

Adjustments: Do not attempt any adjustments to or alterations of this product. Do not open the protective housing of the light section sensor. There are no user-serviceable parts inside.

The glass optics cover is the only aperture through which laser light may be observed on this product.

Figure 16 Laser prevention instructions Part 2



CAUTION: Use of controls or adjustments or performance of procedures other than specified herein may result in hazardous light exposure.

The use of optical instruments or devices in combination with the light section sensor increases the danger of eye damage!

Operation notice in accordance with UL certification:
CAUTION – Use of controls or adjustments or performance of procedures other than specified herein may result in hazardous light exposure.

Figure 17 Laser prevention instructions Part 3

These indications where extracted directly from Leuze Electronic (2014) as they are not to be modified, and should be followed exactly as they are given by the manufacturer.

2. Operation Principle of the Light section sensor

This section is intended to explain how the laser sensor works, the limits it may present on given circumstances and possible solutions to these limits.

On the next image, it is possible to observe a scheme of the functioning of the sensor.

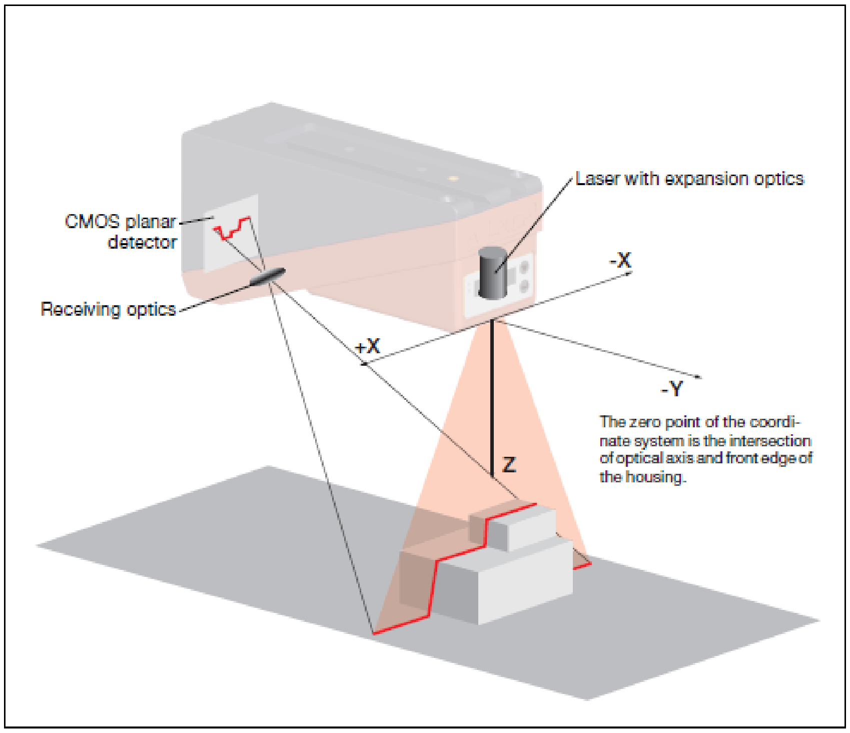

CMOS planar detector
Receiving optics
Laser with expansion optics
-X
+X
-Y
Z
The zero point of the coordinate system is the intersection of optical axis and front edge of the housing.

Figure 18 Leuze LPS functioning scheme

Image from Leuze Electronic (2014)

As it is shown, the laser beam is projected over a surface from the laser. The laser line will be projected to the CMOS planar detector accordingly to a position relative to the point of reflection of the beam on the surface. This allows mapping the gradient of the laser beam proportional to the real 2D cut. The sensor then converts these values to a certain position, both in X and Z coordinates, that correspond to the real life surface. With one isolated lecture, a 2D profile will be obtained. When making continuous lectures, and if there is a relative movement between the sensor and an object, it is possible to create a 3D map of the readings.

There is one common problem that may present when making lectures, and it is called "Occlusion".

As the scanning of a surface is based on the reflection phenomenon from the laser beam to the CMOS detector, if the beam is obstructed partially or completely, it will not reach the destination as it is expected. This can be better explained when observing these two images:

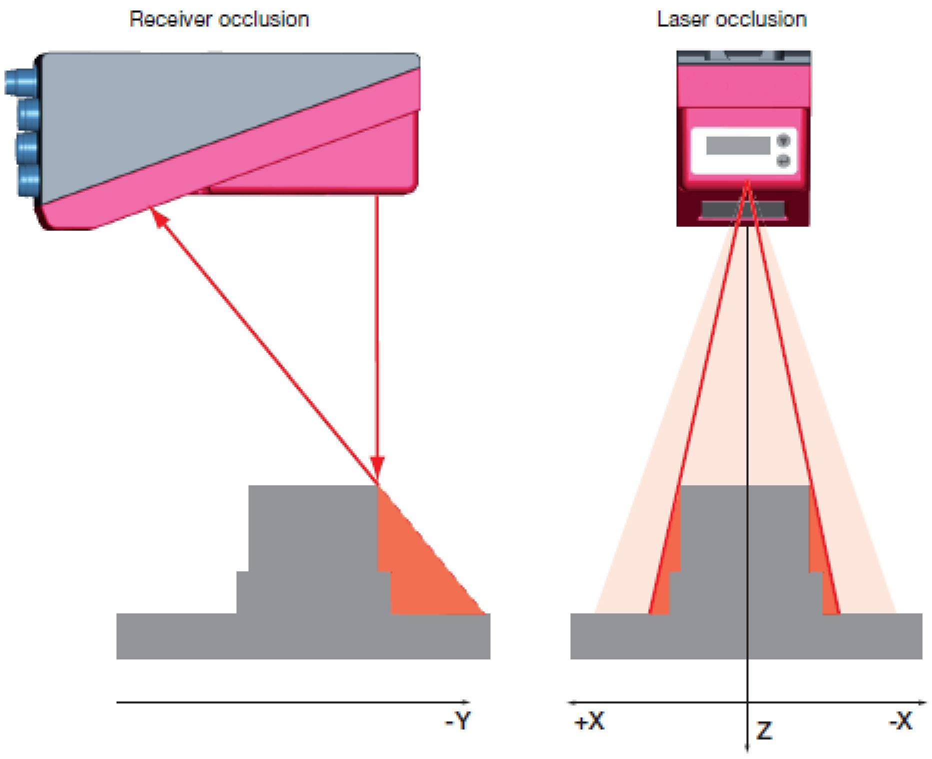


Figure 19 Occlusion on Leuze LPS

There can be mainly two types of occlusion. On the first image it is possible to observe a "Receiver occlusion" and on the second image a "Laser occlusion".

Receiver occlusion

As it can be observed on the first image, given certain characteristics of an object present on the scan area, when the laser beam reaches a certain point of the object, it may be obscured by the object itself when reflecting to the CMOS, creating a "shadow area" that will never reach it, and hence it will not be mapped. This occurs specially when there are very high features on an object. The red area on the image shows all the area that will not be mapped by the sensor due to the upper right corner of the object.

Laser occlusion

In the second image, the occlusion is a little bit different as in the first. In this case, it's the laser beam that will never reach certain points of the object, as they are obscured by the object itself. The laser beam will effectively be mapped by the sensor, but as it never reaches the features hidden by the top right corner, they will never be shown on the CMOS. This occurs because the laser beam is projected with an angle from the sensor.

Leuze Electronic offers a suggestion for avoiding this type of problems, depending on both cases.

Possible measure for receiver occlusion

One option is to align the sensor (or the object) in a position that allows measuring all the features from the object. Another is to use a second sensor that faces 180° opposite on the Z axis. This allows the sensor to receive information from both sides, capturing the information that was missing on the first sensor.

Possible measure for laser occlusion

For this type of occlusion, an easy solution is using multiple light section sensors. They are programmed in cascade so they don't affect the others, and this way is possible to access most features of the objects.

Measure adopted for this project

The DHBW University owns only one light section sensor, so the use of multiple sensors is not an option for the moment. Also, the idea for the project is to receive objects in random order and of different shapes, so aligning the objects or the sensor for a specific object and position is not viable either.

The actual project will be developed knowing the existence of these phenomena; however, a very feasible solution will be exposed in the Recommendations and Future work's section.

Resolution of the sensor

The following graphic shows the relation between the distance sensor-object vs. the resolution of the lecture in mm.

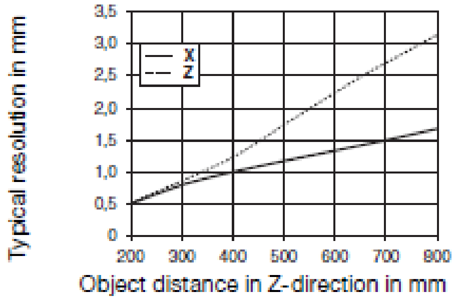


Figure 20 Distance sensor-object vs. Resolution of lecture

Image from Leuze Electronic (2014)

As it can be seen on the image, the resolution is higher when the sensor is closer to the object. This means that it is easier to obtain more accurate images when closer to the object. This will be further discussed in the Recommendations and Future work's section for a possible variation of the application developed that might enhance the performance of the scanning and the time needed for the whole process.

It is also important to know the ranges within the sensor can work properly. The Z range goes between 200 to 800 mm, and the maximum length of the laser line is 600mm (it decreases when Z distance decreases). Also, the response time of the sensor or for a measurement is of 10ms.

3. Mounting and connection of the sensor

The correct mounting of the sensor is relevant for acquiring proper lectures of a surface. On this section it will be mentioned the steps taken for achieving a proper mounting of the sensor and how to connect it.

Alignment

On the next image, the alignment of the sensor will be explained.

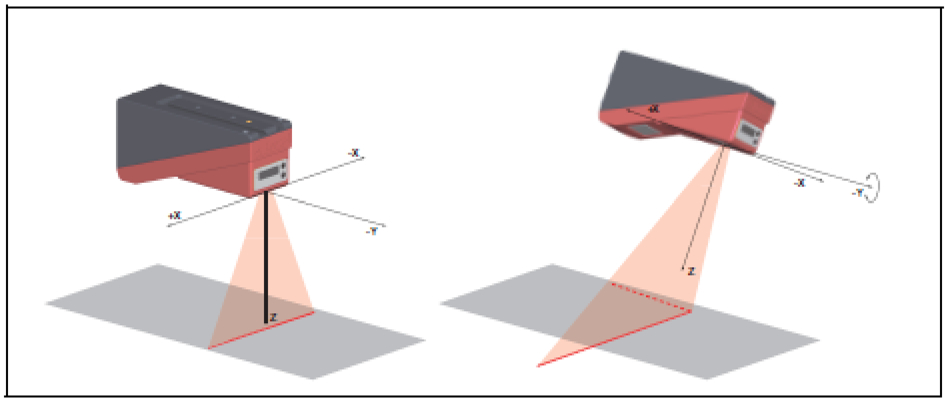


Figure 21 Alignment of Leuze LPS

Image from Leuze Electronic (2014)

The sensor should be mounted as the first image shows, aligned on the Y axis, with the measuring surface parallel to the XY plane of the sensor. The sensor comes with a screen that shows the Z distance at the beginning, middle and end of the laser beam. The three measures should be equal, indicating a correct alignment of the sensor.

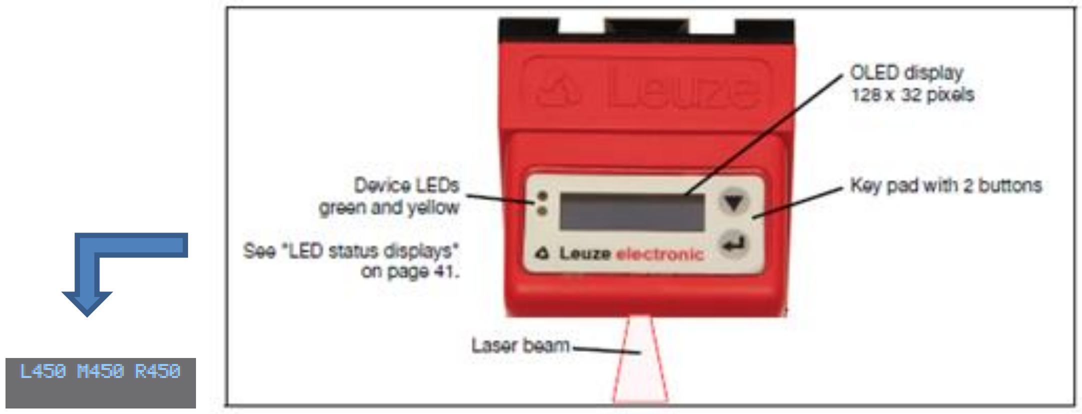

L450 M450 R450
Device LEDs green and yellow
See "LED status displays" on page 41.
OLED display 128 x 32 pixels
Key pad with 2 buttons
Leuze electronic
Laser beam

Figure 22 Screen output for aligning

When aligning the sensor, the screen should show values as the ones in the image before, equal for the Left (L), Middle (M) and Right (R) value. The numeric value corresponds to the distance between the sensor and the measuring surface in mm. If the sensor shows 000(mm) it means that the measuring surface is out of the measuring range of the sensor.

Mounting base

For the first phase of the project, the objective is to make measures with the sensor, but the UR will be programmed on a further phase of the development. For this reason, a temporal mounting base will be assembled. This base will be stationary, and the relative motion between the sensor and the objects will be produced by manually translating the objects as if they were on a transporting band.

The resulting base was built with materials available at the storage room of the Robotics and Automation laboratory from the DHBW. It will not be for permanent use, but only for making tests to the sensor, and taking simple measurements. The picture below shows the resulting base. On the second picture, it is shown an L shaped plate that was machined to connect the sensor with the base structure. The sensor has holes for its proper mounting with screws, so the plate was machined with this purpose.

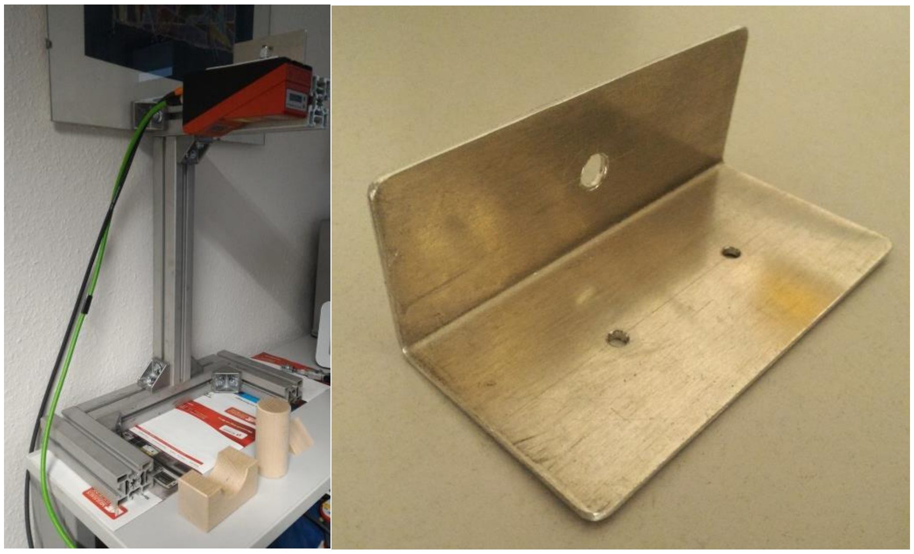


Figure 23 Temporal mounting base for LPS

It is important to place the safety labels of the laser beam on a visible place, and to mount the sensor in a position that the laser beam will not reflect directly to the eyes of the operator.

Electrical connection

The following image shows the connectors of the Leuze sensor.

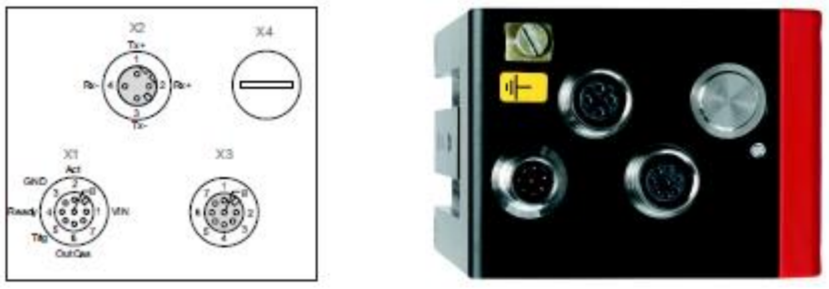


Figure 24 LPS connectors. Image from Leuze Electronic (2014)

For the purpose of this project, only the X1 and X2 terminals will be used. The X1 terminal is the Power supply connection and the X2 terminal is the Ethernet connection between the sensor and the PC.

On this first phase of the project, the sensor will be powered by a source of +24V DC, with a restricted current of 2 A.

4. Communication Protocol Structure

After studying the basics of the Leuze LPS sensor and how to set it up, it is time to study the protocol that the sensor employs for the Ethernet communication. On this section, an explanation of the protocol and the most common codes that will be needed for the project will be explained.

Measure Mode and Command Mode

First of all, the sensor can work in two different modes:

- Measure mode:
  On this mode the sensor can be configured to measure on "Free Running", where it will be measuring periodically by a previous set frequency or on "Input

Triggered", where it can receive an external signal that will trigger a single measure from the sensor.

– Command mode:

  On command mode, the sensor will receive instructions and configuration parameters from the Ethernet connection via the X2 connector and a UDP communication.

For this project, the Command Mode was chosen as the best option, as it enables the user to configure parameters while operating the sensor, and gives more flexibility for programming the interface of the program.

Header and User Data

The protocol that the LPS employs is integrated by 2 parts, the Header and the User Data. The Header is 30 bytes long, and it is distributes as showed next:

Header:

Byte 0-1	Byte 2-3	Byte 4-5	Byte 6-7	Byte 8-9
Start seq. 1	**Start seq. 2**	**Fill Char**	**Command No.**	**Fill Char**

Byte 10-11	Byte 12-13	Byte 14-15	Byte 16-17	Byte 18-19
Packet No.	**Fill Char**	**Transaction No.**	**Status**	**Encoder H**

Byte 20-21	Byte 22-23	Byte 24-25	Byte 26-27	Byte 28-29
Encoder L	**Fill Char**	**Scan No.**	**Type**	**No. of User Words**

Table 4 LPS Header

The User Data's size depends on the information sent by the sensor to the PC.

Start Seq. 1 & 2

Start sequence 1 and 2 have both a value of 0xFFFF. They are the indicators of a new message.

Fill Characters

Every Fill Character has a value of 0x0000, and they are used to separate the different information sent or received on the Header.

Command Number

These 2 bytes specify the command sent from the PC to the sensor, as well as the response from the sensor to the PC. The different values that can be on this space will be discussed further.

Packet Number

It’s a value used by the manufacturer.

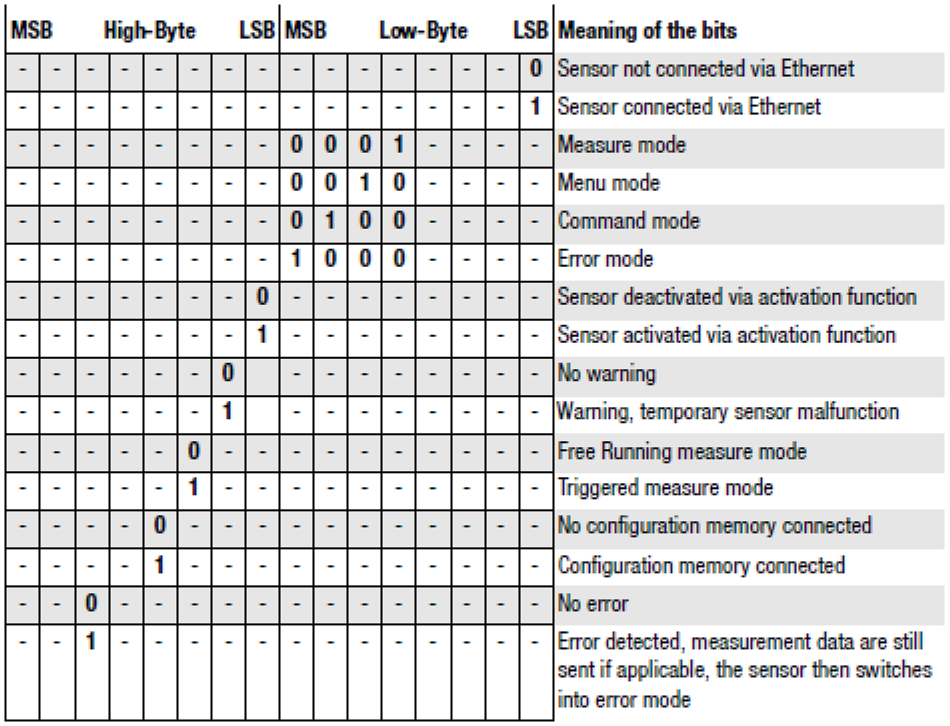
Transaction number

In Measure mode, the value of Transaction No. will always be 0x0000. However, for Command mode, the value will be the same value of the Command Number that the sensor is answering for.

For example: if the command sent by the PC is 0x434E (Connect to Sensor), when the sensor acknowledges with a response, the Transaction number will have the value of 0x434E.

Status

This word indicates the status of the sensor. A table with the different values is presented next:



MSB			High-Byte				LSB	MSB			Low-Byte				LSB	Meaning of the bits
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	Sensor not connected via Ethernet
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	Sensor connected via Ethernet
-	-	-	-	-	-	-	-	0	0	0	1	-	-	-	-	Measure mode
-	-	-	-	-	-	-	-	0	0	1	0	-	-	-	-	Menu mode
-	-	-	-	-	-	-	-	0	1	0	0	-	-	-	-	Command mode
-	-	-	-	-	-	-	-	1	0	0	0	-	-	-	-	Error mode
-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	Sensor deactivated via activation function
-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	Sensor activated via activation function
-	-	-	-	-	-	0		-	-	-	-	-	-	-	-	No warning
-	-	-	-	-	-	1		-	-	-	-	-	-	-	-	Warning, temporary sensor malfunction
-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	Free Running measure mode
-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	Triggered measure mode
-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	No configuration memory connected
-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	Configuration memory connected
-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	No error
-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	Error detected, measurement data are still sent if applicable, the sensor then switches into error mode

Table 5 Status interpretation of LPS

Table from Leuze Electronic (2014)

Encoder High & Low

The encoder is used for sensors with encoder input: however, it will not be implemented on this project.

Scan Number

This value indicates the number of single consecutive measures that have been processed. The X and Z values from a single lecture are identified with the same Scan number.

Type

Its value is 0x0010, and indicates that it handles 16 bit data.

Number of user Data

It defines how many words of User Data are to be received on the actual packet. The values can be 0, 1, 2, 3, 376 or 480.

User Data

When measuring, 376 values will be returned to the PC with X or Z values. The resolution of the lectures is 1/10 mm. The X values are signed values in two's complement that go from 0 to 65535. Values between 0 (0x0000) and 32767 (0x7FFF) represent positive values, and values between 32768 (0x8000) and 65535 (0xFFFF) are negative values, where 65535 represents -1.

Commands

The Leuze LPS has a wide list of commands for configuration and measuring of the sensor. Some of the most important commands are listed below and their use will be explained afterwards. The Acknowledge response is also listed, which would be the response from the sensor to the PC. If the command is acknowledged, the response is 'Ack'=0x4141 and if the command is not processed the response is 'Not Ack'=0x414E

In measure mode

Action	Command	Ack	Not Ack
Connect to sensor	0x434E	0x4141	0x414E
Disconnect from sensor	0x4443	0x4141	0x414E
Enter command mode	0x3132	0x4141	0x414E
Exit command mode	0x3133	0x4141	0x414E
All other commands	-	-	0x414E

Table 6 Commands in measure mode

From this commands, the execution should be like this:

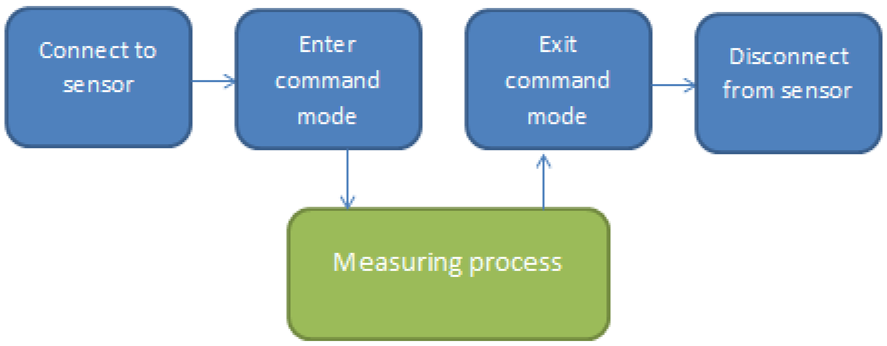


Figure 25 LPS general algorithm

The measuring process is shown separately because is the complete algorithm that will be elaborated for measuring objects, sending data to the PC and storing the data with the proper format. It will be detailed later.

In command mode

The complete list of commands when operating in Command Mode can be found in Leuze Electronic (2014). A list of the commands that will be used for the purposes of this project are listed next; however, for future extent of this project, it is important to make a revision of the other available commands, to understand the extent and possibilities offered by the LPS sensor.

Action	Command	Ack	Not Ack
Set laser Gate	0x0001	0x4141	0x414E
Trigger single measurement	0x0003	0x4141	0x414E
Get X coordinates	0x0011	0x0012	-
Get Z coordinates	0x0013	0x0014	-
Get ZX coordinates	0x005F	0x0060	-

Table 7 Commands in command mode

Set laser Gate

This command is used to toggle the state of the laser. When the laser is ON it will change its state to OFF and vice versa. It is important to consider the use of this command if the sensor is going to be at some point of the process looking to a direction that is not the measuring surface, to avoid injury by looking at the laser beam.

Trigger single measurement

The trigger command executes one single measurement at the moment the command is received. It stores both X and Z values (376 values for each coordinate) in the sensor buffer.

Get X Coordinates

This command is a request from the PC to the LPS to send a packet with the X coordinates acquired in the previous measurement. 376 signed Values will be sent to the PC, each within a range between 0 and 32767 for the positive values, and between 32768 and 65535 for the negative values. The packet is transmitted to the PC with a command number of 0x5858.

Get Z Coordinates

This command is a request from the PC to the LPS to send a packet with the Z coordinates acquired in the previous measurement. The Z coordinates are 376 unsigned values the can have a value in a range between 0 and 8100. The packet is transmitted to the PC with a command number of 0x5A5A

Both X and Z coordinates will be transmitted with the same Scan Number when they belong to the same lecture.

Get ZX Coordinates

This command transmit both X and Z coordinates to the PC; however, this command is only available for the LPS 36HI, and for the project the sensor available is the LPS 36. Hence, this command will not be implemented in the project.

Set Single Inspection Task Parameter

This command, with the command number 0x006D can configure a series of parameters of the sensor like:

- Name of inspection task: name used for the actual measurement (Parameter ID: 0x0BB9).
- Exposure Duration of the laser: configuration of laser exposure in µs depending on the brightness of the objects to be measured. A normal exposure is of 261 µs. For bright objects 97 µs are recommended, for dark objects a mayor exposure of 655 µs is advised and for normal to bright objects it is recommended an exposure of 328 µs approximately (Parameter ID: 0x0BBD).
- Detection range of the LPS: the X coordinates and Z coordinates detection ranges can be separately modified to ignore values out of the range of interest (Parameter ID: 0x0BBF for X coordinates and Parameter ID: 0x0BC0 for Z coordinates).

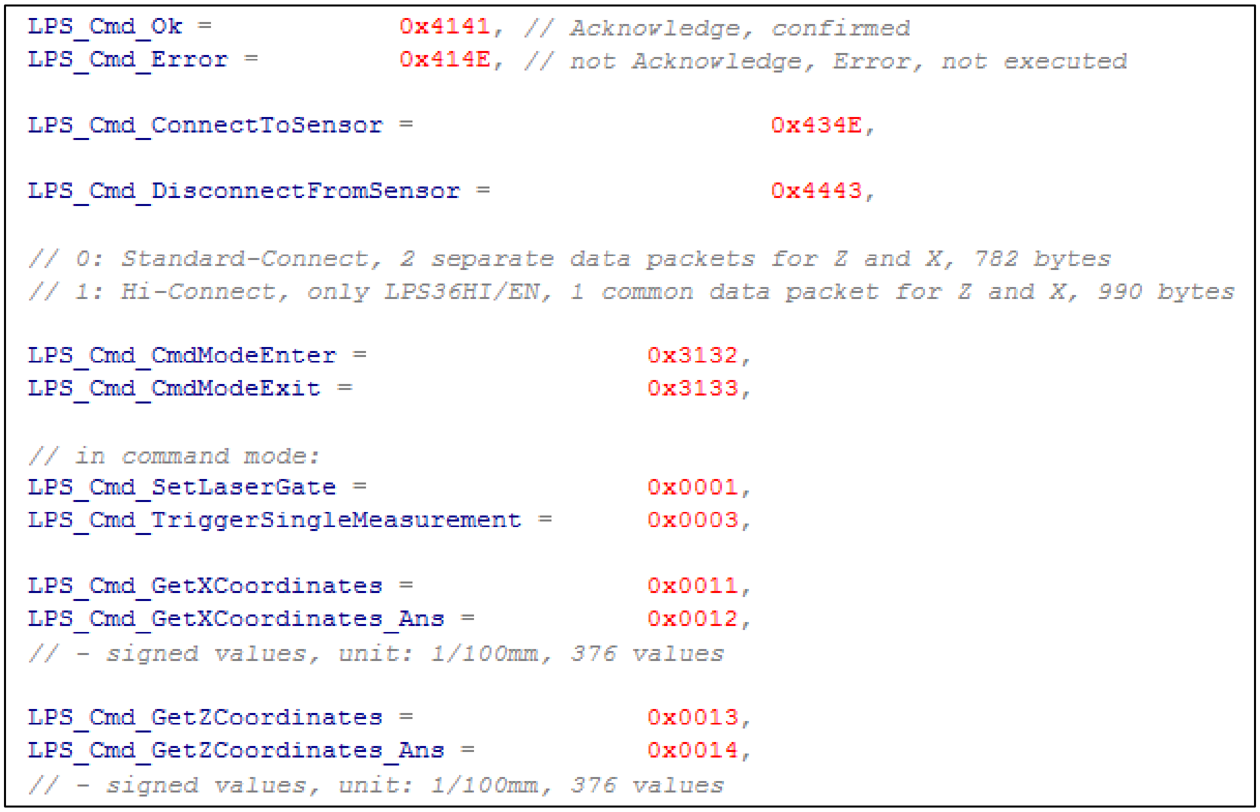
These task parameters are configured using 3 words from the User Data. The first word is the parameter ID to be modified, and the next two words define certain values depending on the parameter to be modified.

5. Programming the LPS communication

The programming of the LPS communication requires order and simplicity. The protocol to be used (UDP) has to follow specific rules, which can be found on several tutorials. Some useful tutorials and examples can be found in Silver Moon (2011) and Silver Moon (2012).

Header file

Most of the commands mentioned previously are the ones that will be used in the program. Knowing all these commands, a header file with the name "leuze-lps.h" was created to define a name for each command, and make the interaction with the programmer friendlier. A sample of the header is shown below:



```
LPS_Cmd_Ok =              0x4141, // Acknowledge, confirmed
LPS_Cmd_Error =           0x414E, // not Acknowledge, Error, not executed

LPS_Cmd_ConnectToSensor =                           0x434E,

LPS_Cmd_DisconnectFromSensor =                      0x4443,

// 0: Standard-Connect, 2 separate data packets for Z and X, 782 bytes
// 1: Hi-Connect, only LPS36HI/EN, 1 common data packet for Z and X, 990 bytes

LPS_Cmd_CmdModeEnter =                      0x3132,
LPS_Cmd_CmdModeExit =                       0x3133,

// in command mode:
LPS_Cmd_SetLaserGate =                      0x0001,
LPS_Cmd_TriggerSingleMeasurement =          0x0003,

LPS_Cmd_GetXCoordinates =                   0x0011,
LPS_Cmd_GetXCoordinates_Ans =               0x0012,
// - signed values, unit: 1/100mm, 376 values

LPS_Cmd_GetZCoordinates =                   0x0013,
LPS_Cmd_GetZCoordinates_Ans =               0x0014,
// - signed values, unit: 1/100mm, 376 values
```

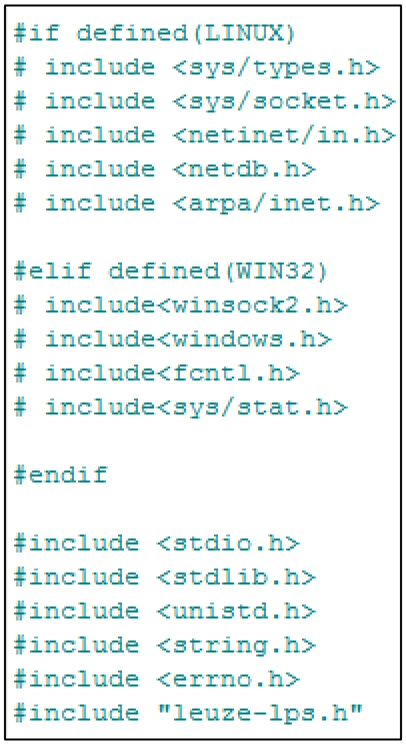
Figure 26 Sample of header file

Also, a Header structure with the format and size previously explained was defined.

After having defined all the variables needed for commands, the programming of the sensor communication was developed. A file with the name "sensor_client.c" was elaborated. Its different sections will be explained next:

Libraries inclusion

On the first section of the code, all libraries included are listed. A conditioning evaluation of the OS of the system defines whether to include libraries for Windows or for Linux, and finally those libraries needed for both. On the next image is shown how the program includes the needed libraries depending on the OS making the program portable.



```
#if defined(LINUX)
# include <sys/types.h>
# include <sys/socket.h>
# include <netinet/in.h>
# include <netdb.h>
# include <arpa/inet.h>

#elif defined(WIN32)
# include<winsock2.h>
# include<windows.h>
# include<fcntl.h>
# include<sys/stat.h>

#endif

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <errno.h>
#include "leuze-lps.h"
```

Figure 27 Included libraries

IP Configuration

After including the corresponding libraries, the algorithm for UDP communication was developed for connecting the sensor with the PC. An IP address of 192.168.28.50 was defined for the sensor with a Port Number on 9008. The port for the PC was defined as 5634. The communication between both hosts will happen in these two ports. The PC's IP was defined as 192.168.28.51 as it needs to be on the same subnet. Both hosts use a subnet mask of 255.255.255.0.

Use of Wireshark to read Ethernet data

After programming and debugging the UDP communication and configuring the proper parameters for the sockets, the communication was finally established. To ensure that the correct commands where send to the sensor, a Sniffer program was installed on the PC. With the use of the free software Wireshark, the data sent and received by the Ethernet connection of the PC was monitored. After a series of tests an correction of the order of the bytes when sending them to the sensor because they were in Big Endian format and were supposed to be in Little Endian format (Little byte first), the following data was captured by Wireshark:

Word	1	2	3	4	5	6	7	8	9	10
Sent	FFFF	FFFF	0000	4E43	0000	0000	0000	0000	0000	0000
Response	FFFF	FFFF	0000	4141	0000	0000	0000	4E43	1001	0000
Word	11	12	13	14	15					
Sent	0000	0000	0000	0000	0000					
Response	0000	0000	0000	1000	0000					

Table 8 Sent and received message from LPS

The command number (Word No. 4) of the message sent is 0x434E (4e4E43 in Little Endian format) which represents the command "Connect to Sensor". On the response, the Transaction Number is 0x434E, as it is the response from the received command. The Command number of the response is 0x4141, which acknowledges that the connection was successfully established.

First successful lecture

After implementing a simple lecture with the sensor and importing the X and Z coordinates to a file, a graphic was created using a simple data tool, in this case Microsoft Excel.

From the scene shown below, a profile was obtained.

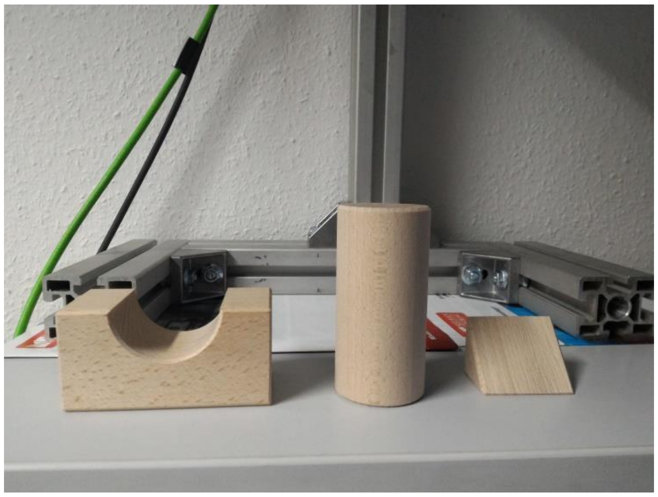


Figure 28 First scanning scene

From these objects, the following profile of a single lecture was obtained:

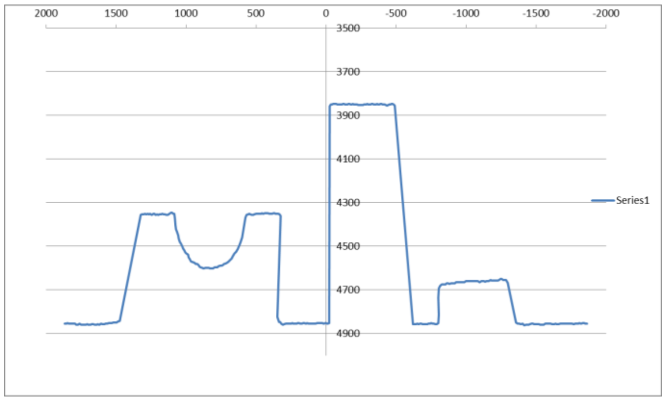


Figure 29 Obtained profile. Signal z in 1/0 mm. Signal x in 1/10 mm.

The units on the graphic are in 1/10mm units and represent the X and Z distances between the sensor and the objects on the table. The X and Z axis don't have the same scale, which makes the image appear distorted. It is also observable the phenomenon of occlusion present on the lecture, that was previously explained.

Improvement of the code for 3D measuring

After reading one single profile and storing it to a file, it is time to simulate a continuous lecture with the sensor, with relative movement between the sensor and the objects on the measuring surface. As the sensor at this point doesn't have the capability to move, the measuring surface will be disposed as showed on the next image:

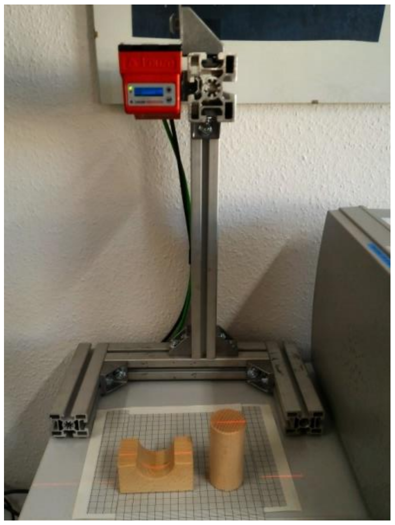


Figure 30 Measuring manual surface with millimetric bottom for simulating movement

As it can be observed on the image, a millimetric sheet was fixed to the surface of the table. Over this sheet, another millimetric sheet was placed (not fixed). The objects will be placed over the second sheet, allowing it to be manually moved every one millimeter and making one lecture from the sensor on each position.

The code was modified to request permission to the user to make a new scan every time, or to terminate the lectures. Every time a scan is made, the values are stored at the end of a *.txt file with a name that the user indicated at the beginning of the program. The final result is a file with multiple lectures of the sensor. The Y coordinate, which is not provided by the sensor, was pseudo-assigned as the number of lecture, starting from lecture number 1, until the user decided to terminate the program at lecture number N.

Graphical interpretation of data

For interpreting the data acquired, more specialized software than Microsoft Excel is needed. GNUPlot and MeshLab are both free software that allow visualizing 3D data. Both will be tested and analyzed to choose the optimum program. This will be used only for easily visualizing the obtained data after a complete lecture, and does not affect the results of the project. An idea about the data visualizing will be proposed on the Recommendations and Future work's section.

GNUPlot

The GNUPlot software is free software that is compatible with most common OS. It is used for graphing and visualizing functions and data. It is controlled by console, which allows the user to modify parameters and modify information of the input data. After investigating on the different commands and possible the configurations that the program uses, two types of images where obtained: Meshed images and Dots images.

The Meshed image creates a solid surface with the input 3D points that where introduced, and allows the user to visualize the result as a whole object as showed next:

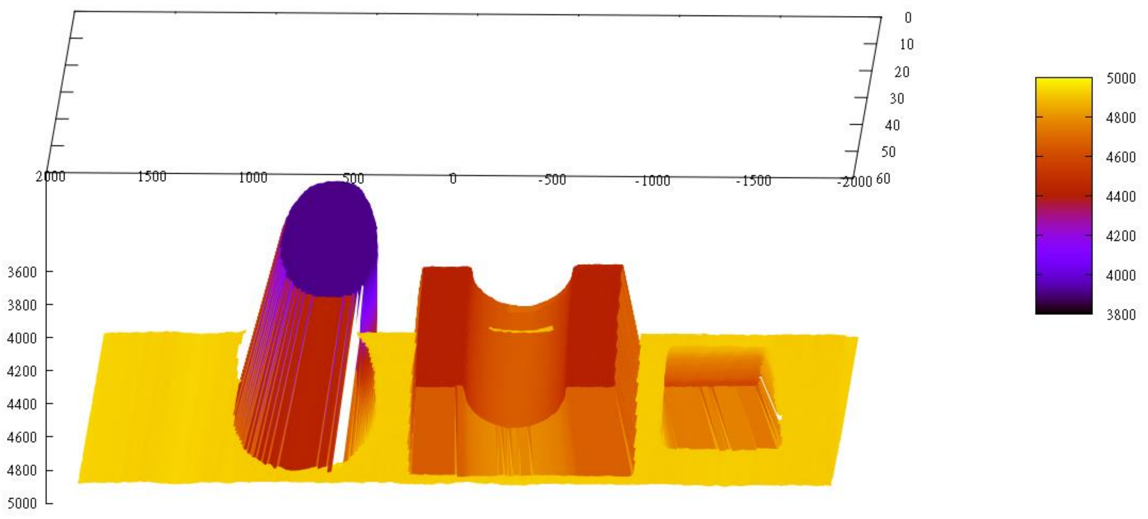


Figure 31 Mesh of obtained image

When rotating the graph, this was observed:

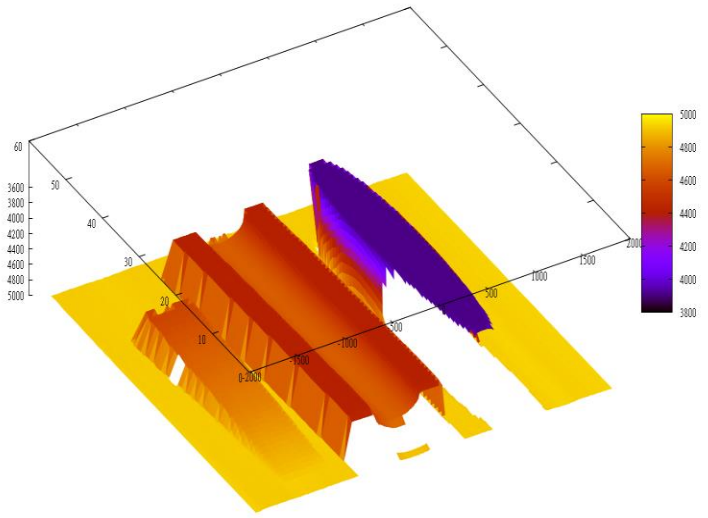


Figure 32 Mesh obtained, second view

As it can be appreciated, the figures present some "holes" on the surface. This occurs because of the previously mentioned occlusion phenomenon, which doesn't allow scanning completely the figures on the measuring surface.

The Dots image presents the same results, but it is a more exact representation of the results, as it shows the data in a 3D space as it is on the input data. The image below shows a Dots image obtained in another test:

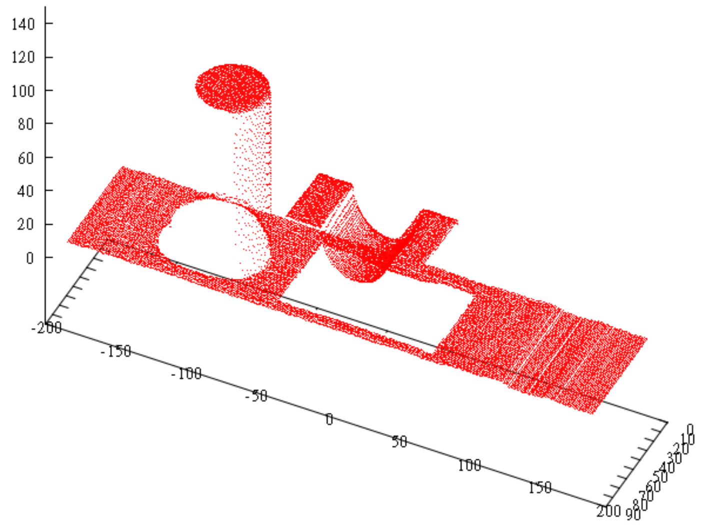


Figure 33 Dots image with GNU Plot

The results obtained with GNUPlot are very clear, but require a certain amount of commands to obtain images like the ones presented. The commands depend on the size of the image, the quantity of points from the input data and the output desired by the user.

MeshLab

MeshLab is an open source tool that allows creating 3D meshes with an input of points. It can also create Dots Images using the *.ply format. It requires a header on the file with information of the type of data (float, integer, etc.) and the quantity of points on the input. Tutorials on how to create *.ply files can be found at FSU PLY Files (n.d.), Paulbourke (n.d.) and FSU PLC Files (n.d.). Also an example on creating a mesh from point clouds can be found at Tenney (2012).

After creating a proper header, the following result was obtained:

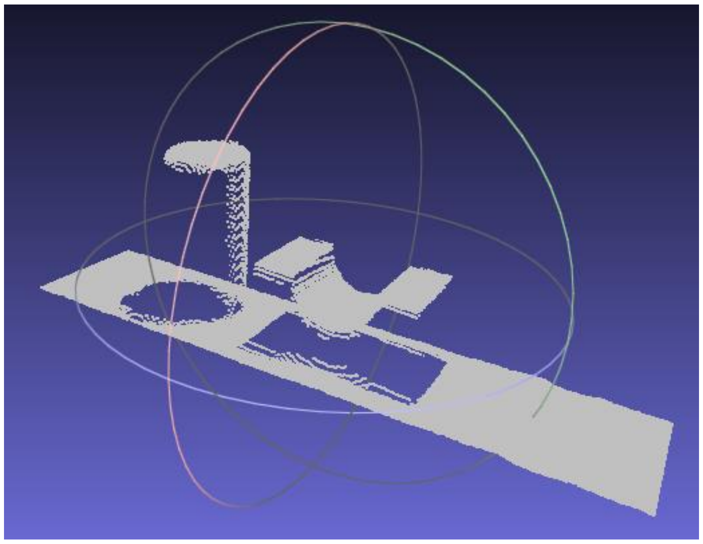
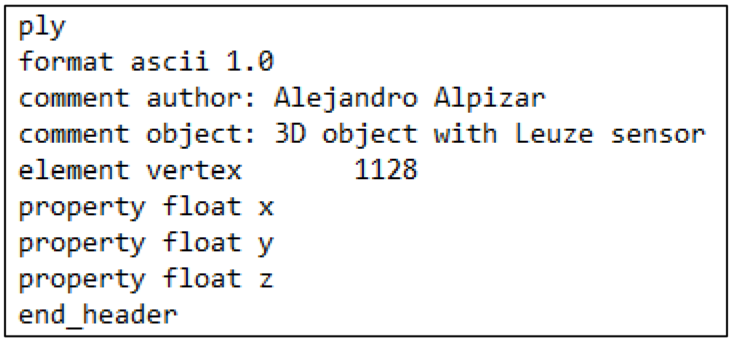


Figure 34 Dots image with MeshLab

MeshLab allows the user to obtain easy visualization of 3D data, and requires a small amount of parameters to do so. For this, the rest of scans will be analyzed with this software.

An example of the header for a *.ply file is as follows:



```
ply
format ascii 1.0
comment author: Alejandro Alpizar
comment object: 3D object with Leuze sensor
element vertex      1128
property float x
property float y
property float z
end_header
```

Figure 35 Ply header

The variable "vertex" indicates the total number of points with X, Y and Z coordinates. This number is obtained from the number of lectures that were made when scanning the surface. As every lecture returns 376 pairs of X and Z coordinates, the vertex can be calculated as:

Vertex= 376 * No. of lectures

Measuring process

The final measuring process flowchart can be seen in the next image:

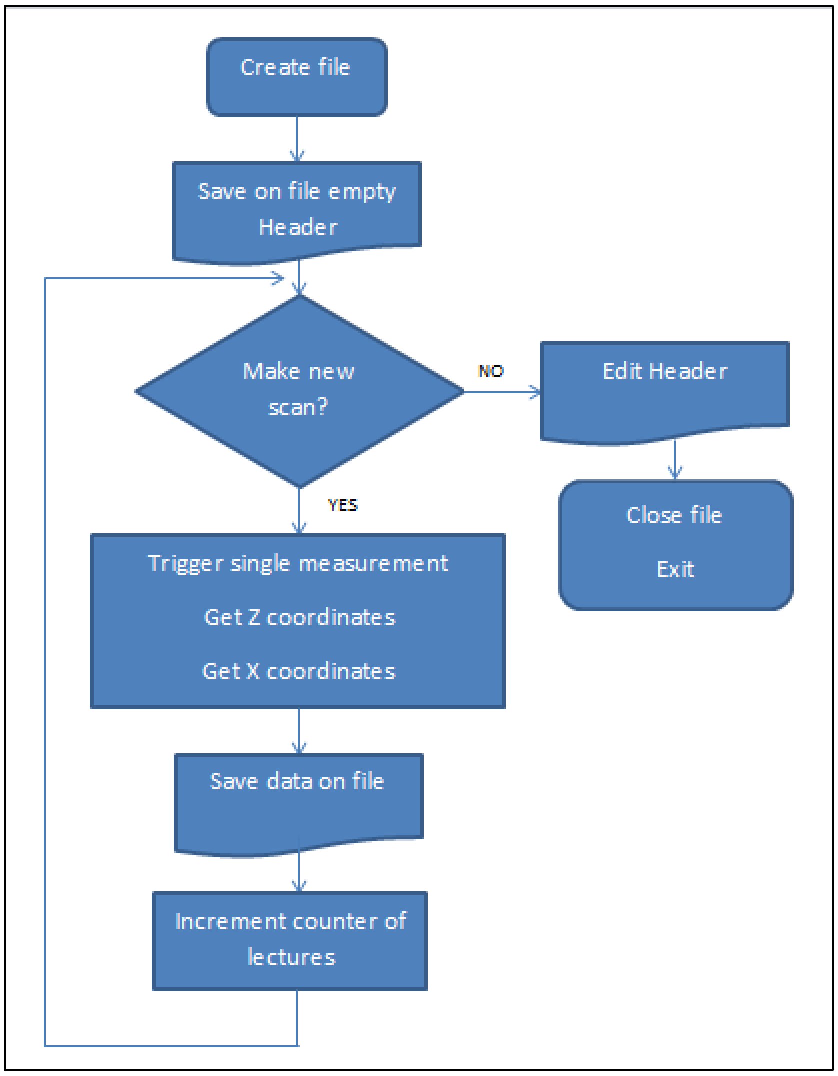


Figure 36 LPS measuring flowchart

An example of a file created with the Leuze program can be found on the Apendix F.

The program for measuring will be edited when the UR phase is working, as the user will have no need to indicate to make a new measure. Instead, the UR will send a command to the Leuze program requesting a new measure when it has reached a new position. The details of these will be explained in the next section.

v. Programming the Object Detection and Recognition Phase

Originally, the UR phase was planned to be developed before the Vision Phase. However, the Robotics and Automation Laboratory has certain restrictions when working on the Laboratory. One of these restrictions is that any user has to be accompanied by another professor or operator when working on the Laboratory. For this reason, it was decided to work first on the Vision Program with the data already acquired from the previous phase. It will most probably need adjustments when integrating the three phases, but it will allow a continuous work on the project.

1. Selecting the programming tool

Until this point, the programming has been developed in C language, assisted with MSYS for building the executables. For the Vision phase, there needs to be chosen a Vision specialized software that has the capacity for achieving the desired objectives of the project. Different software, such as MatLab, Octave and OpenCV have the utilities needed for complex Image Analysis. There are other libraries that can be used in C that will open more options for developing the project, as is Point Cloud Library (PCL).

One of the desired objectives is to have a low cost system developed. The use of open source software when possible would decrease enormously the cost on software of the overall project. With this in mind, MatLab would not be a desired software to use. Octave is open source, and has a wide variety of tools. OpenCV is an image and video dedicated library, and it is open source. There can be found plenty examples and tutorials on literature and on the web, that can improve the programming. For this reason, it was decided to use OpenCV for the Vision development of the project.

PCL is a powerful library; it allows the user to work with 2D and 3D point clouds in space, and carry out a series of algorithms such as object detection in 3D, Point Cloud Stitching (Fitting two or more Point clouds in one final mesh), and others. For this first project, it will not be implemented; however, its possibilities will be discussed on the Recommendations and Future work's section.

2. Setting up the programming environment

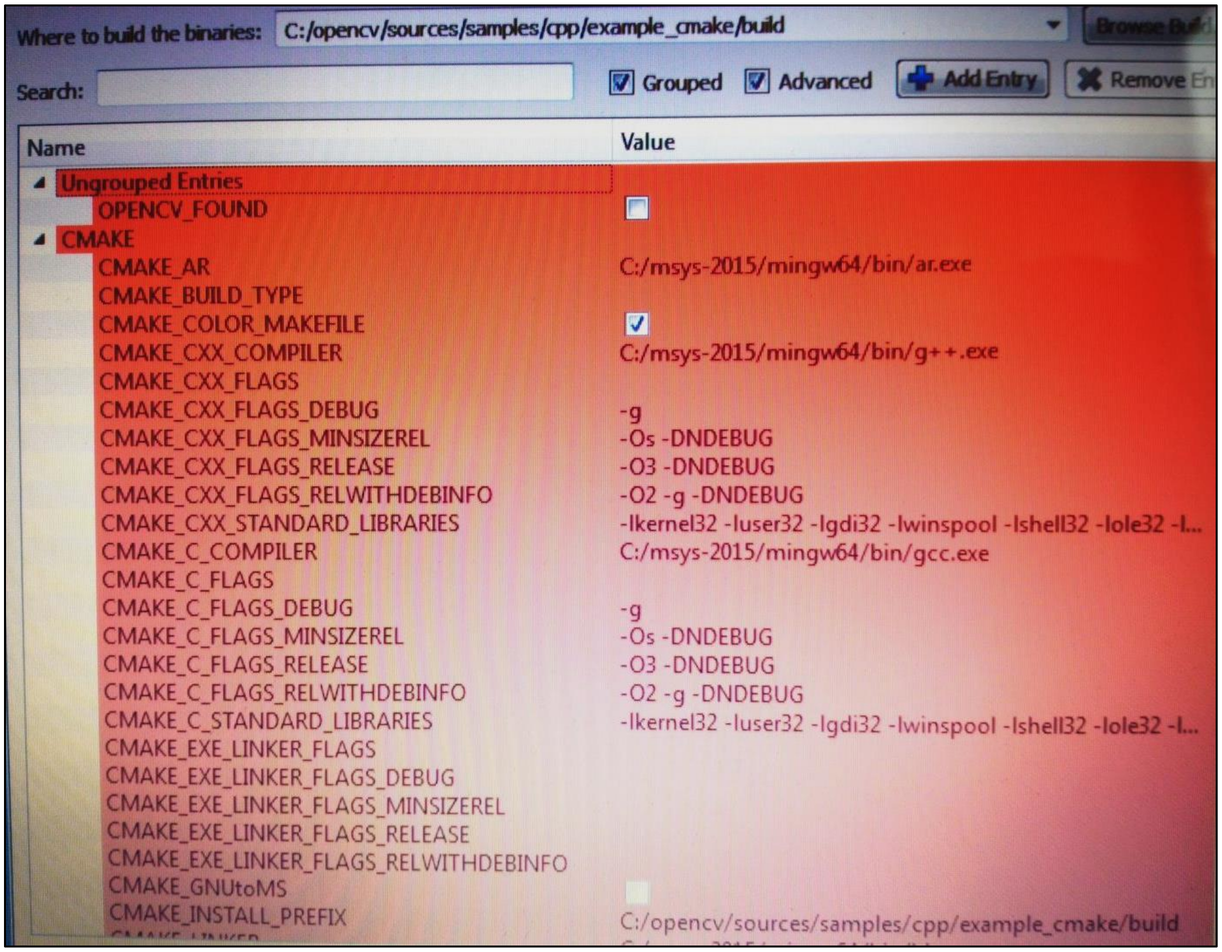
The setup of the environment for using OpenCV with C or C++ is a task that must be done carefully. Otherwise, the final result may not function properly or may even cause the installation to break. The installation depends on the OS, the version of the OpenCV to be used and the complements to be used. There are certain complements that can be installed together to expand the possibilities of use with the library. An example of a complement named Qt will be described in the Recommendations and Future work's section.

Different tutorials and forums that discuss the subject can be found for installing OpenCV in the PC and building it properly at M. Asad (2011), OoenCV.org(n.d.), Stackoverflow (n.d.), Sahid Hasan (n.d.), and others. One suggestion found is to use

Cmake for building OpenCV. Cmake is an open source tool used to control the software compilation process. It is basically a crossed-platform build environment that creates the necessary Makefiles for later use at any compiler of choice.

The steps for building OpenCV where followed:

1. Set the path variable
2. Select the location where the source code is.
3. Select an location where to build the binaries
4. Select MSYS Makefiles as the generator of the project
5. Generate with Cmake
6. Specify the location of the Compilers for C and C++.
7. After the red screen shown in the next picture, select the parameters wanted for the configuration and generate.
8. Close Cmake and on the Command Prompt go to the build folder and enter Make. After it is done, enter Make Install.



Where to build the binaries: C:/opencv/sources/samples/cpp/example_cmake/build

Name	Value
Ungrouped Entries	
OPENCV_FOUND	☐
CMAKE	
CMAKE_AR	C:/msys-2015/mingw64/bin/ar.exe
CMAKE_BUILD_TYPE	
CMAKE_COLOR_MAKEFILE	✓
CMAKE_CXX_COMPILER	C:/msys-2015/mingw64/bin/g++.exe
CMAKE_CXX_FLAGS	
CMAKE_CXX_FLAGS_DEBUG	-g
CMAKE_CXX_FLAGS_MINSIZEREL	-Os -DNDEBUG
CMAKE_CXX_FLAGS_RELEASE	-O3 -DNDEBUG
CMAKE_CXX_FLAGS_RELWITHDEBINFO	-O2 -g -DNDEBUG
CMAKE_CXX_STANDARD_LIBRARIES	-lkernel32 -luser32 -lgdi32 -lwinspool -lshell32 -lole32 -l...
CMAKE_C_COMPILER	C:/msys-2015/mingw64/bin/gcc.exe
CMAKE_C_FLAGS	
CMAKE_C_FLAGS_DEBUG	-g
CMAKE_C_FLAGS_MINSIZEREL	-Os -DNDEBUG
CMAKE_C_FLAGS_RELEASE	-O3 -DNDEBUG
CMAKE_C_FLAGS_RELWITHDEBINFO	-O2 -g -DNDEBUG
CMAKE_C_STANDARD_LIBRARIES	-lkernel32 -luser32 -lgdi32 -lwinspool -lshell32 -lole32 -l...
CMAKE_EXE_LINKER_FLAGS	
CMAKE_EXE_LINKER_FLAGS_DEBUG	
CMAKE_EXE_LINKER_FLAGS_MINSIZEREL	
CMAKE_EXE_LINKER_FLAGS_RELEASE	
CMAKE_EXE_LINKER_FLAGS_RELWITHDEBINFO	
CMAKE_GNUtoMS	☐
CMAKE_INSTALL_PREFIX	C:/opencv/sources/samples/cpp/example_cmake/build

Figure 37 Cmake configuration for building OpenCV

As it can be seen, the building process is not complicated, but it requires of caution. The main problem is to find a process that fits the needs of the user (OS, OpenCV version, compiler, building tool, and others).

Once the programming environment is ready, it is time to plan a solution to be developed.

3. Vision solution proposal

The advantage when working with Point Clouds is that the data is managed in space as it is obtained. Every point (xi, yi, zi) will have a specific coordinate with no possibility of occupying the same space than another point. Libraries like PCL have developed algorithms that handle these data, and allow to extract information and to transform the data (transforming the coordinate system or stitching with another point cloud for example). However, OpenCV cannot handle point clouds, which requires a different approach for a solution.

At this point of the project, there are files created with a list of points with x, y and z coordinates. A proposition for working with these data is to make a transformation from a 3D point cloud to a 2D grayscale image. The intensity of the gray represents in a 2D image the Z value of the 3D image, while the X and Y coordinates remain the same.

4. Developing the solution

Some adjustments need to be made for the transformation proposed, because the sensor range of height is from 0 to 8100 in 1/10 mm, and a grayscale image is usually within a range from 0 to 255.

For a better understanding of the transformation, the previously obtained profile will be used.

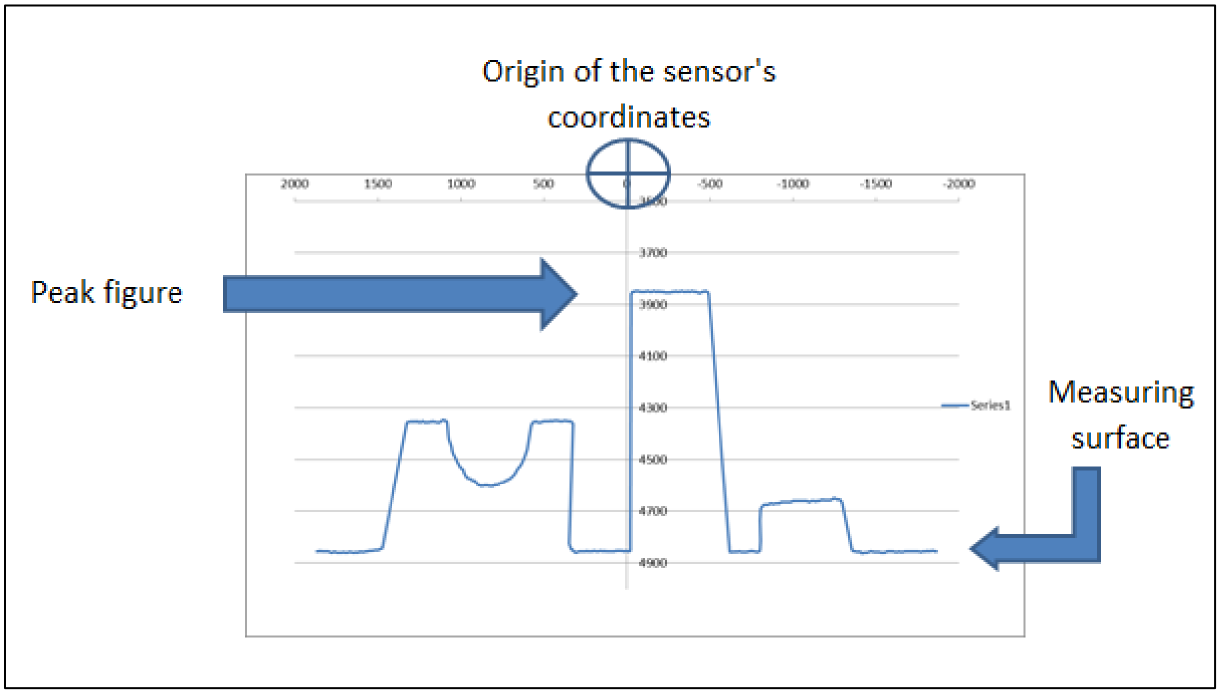


Figure 38 Explanation of transformation to grayscale image

The Z coordinates are 0 when closer to the sensor, and can go up to 8100. However, if a linear transformation was done from range [0 … 8100] to a range [0 … 255] the contrast of the image would be very bad, as all the figures would be concentrated in a small range of the possible values. The peak figure on the scan has a gradient of approximately 100mm out of the 810mm that can be represented. When converting the bottom value of 4900 and the top value of 3900 to grayscale, the following values are obtained:

Bottom value=154

Top value=123

This means that any other figure present on the measuring surface and smaller than the top figure will be within this range, and the rest will not be used. For this, it was defined that the transformation will be calculated between a range that goes from the highest level (the distance between the sensor and the measuring surface) and an established maximum point a little higher than the peak point, in this example could be 1500 in 1/10 mm above the measuring surface.

It is important to note that the distance between the sensor and the measuring surface is variable and should be set depending on the working area and the coordinates from where the sensor will be placed , and that the that the maximum point depends on the figures that will be handled. For the purpose of this project, no figures above 1500 in

1/10 of mm will be handled. The list of figures to be used can be found on the Appendix C.

Since the previous program saves a file with the data, this program will part from that point. It will be a separated program that opens the requested file and loads the data.

The following steps are taken:

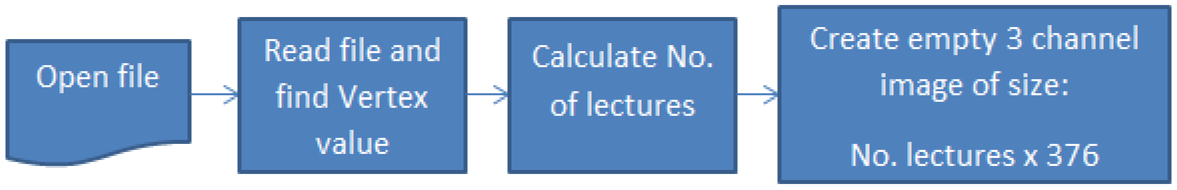

Open file
Read file and find Vertex value
Calculate No. of lectures
Create empty 3 channel image of size:
No. lectures x 376

Figure 39 Opening image from file

The resulting empty image is a 3 channel image that will store the file values as shown:

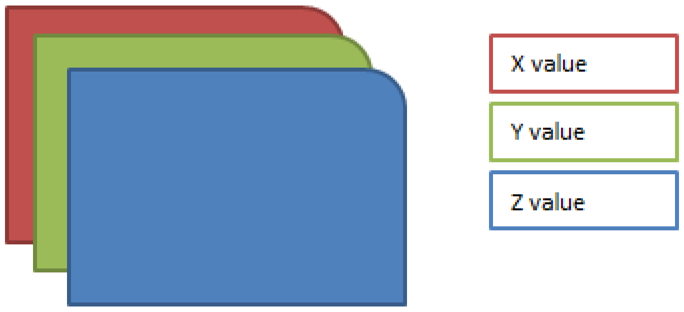


Figure 40 3-Channel XYZ image

The above image uses an analogy of the RGB format of images that is widely use on color images because they also have 3 Channels, one for each color (Red, Green and Blue). However, it does not represent a color image, as the values of each pixel represent the numeric value of each coordinate. To locate a point in space one would need to take the Ri, Gi and Bi values, and then represent them as an (x,y,z) coordinate. To avoid confusion, instead of referring the image as RGB, it will be referred as XYZ channels.

If the Z channel is output as a grayscale image, the above result is printed:

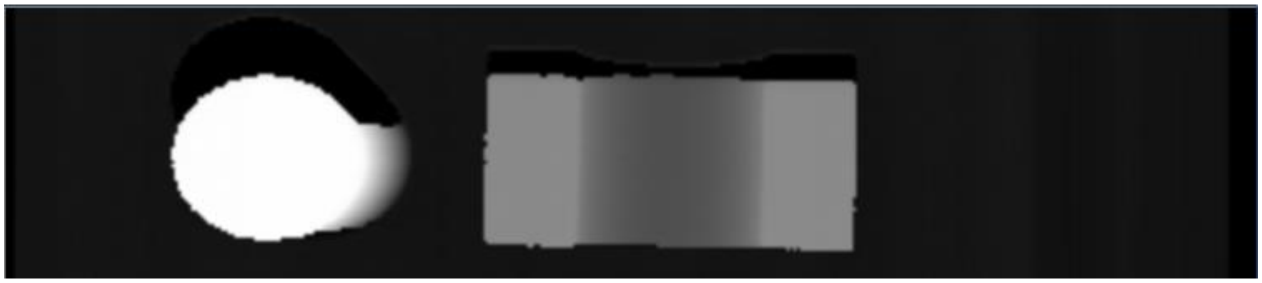


Figure 41 First grayscale image

As expected, the image is a grayscale representation of the 3D image that represents the height with a different intensity of gray. The darkest zones represent what is farther from the sensor, and the brightest zones represent what is closer to the LPS.

The image is a representation of the previously showed image:

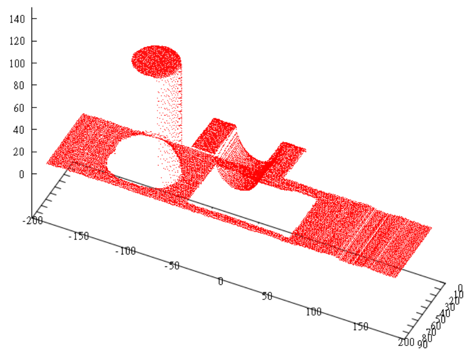


Figure 42 Original image in dots

From the grayscale image, there are two conclusions to make:

- The range transformation works properly for this image
- The occlusion zones are clearly observable in color black (They are represented as a value of 0).

The next step is to program a recognition algorithm to locate and identify objects in the image.

5. Edge detection algorithm

OpenCV offers a wide variety of functions that can help to identify, locate, classify and characterize objects on images. Usually, for doing so, the images need a conditioning step, that enhances the image for the desired purposes. The process usually needs feedback: enhance image, process and observe results. If results are not satisfactory, enhance image, process again and observe results until the final result is satisfactory. All functions from OpenCV can be found at OpenCV Documentation(n.d.).

First test will be made for finding contours on the image. After doing some research on the possible functions to obtain borders, the Canny function was implemented, because it gives equal priority for both X and Y directions. Other functions for finding borders are for example: Gaussian filter, Sobel filter, Scharr filter. The canny edge detector uses the Gaussian filter to filter out noise from the image, and then applies a similar to Sobel filter to find the gradients on the image. It is also known as Optimal Detector.

The canny edge detector returned the following result:

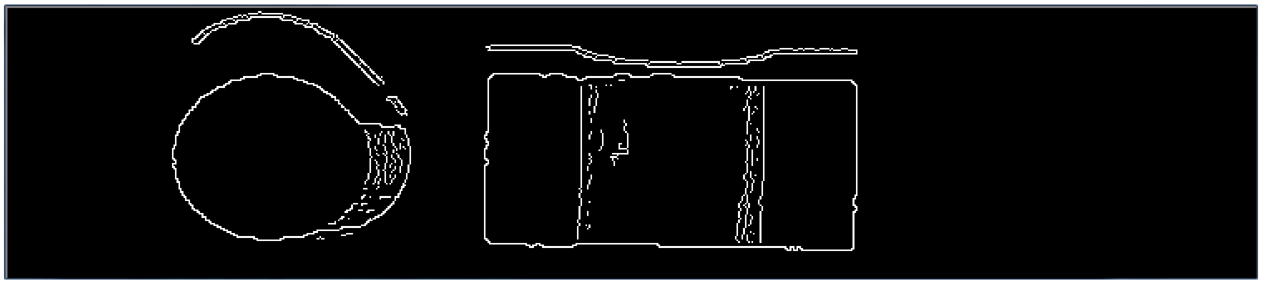


Figure 43 First edge detection

As it can be observed, the edges were found, together with some noise on the resulting image. The threshold parameters on the Canny function allow to make a more or less sensible edge detector depending on the needs of the user. When configuring the thresholds with different values, the following results were obtained:

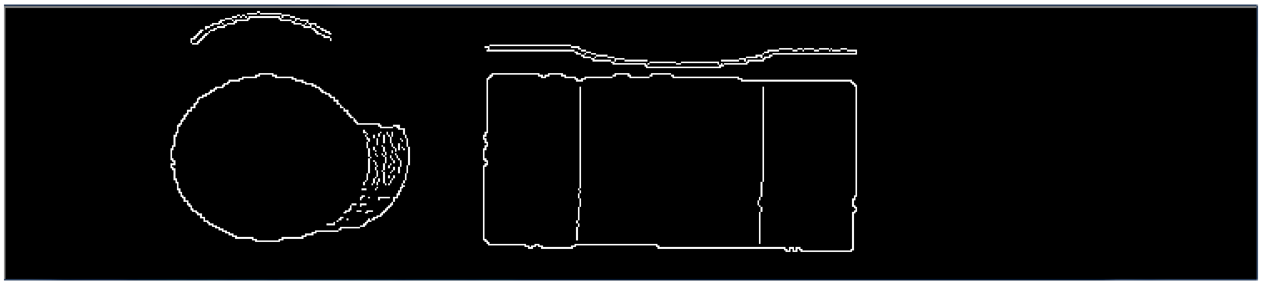


Figure 44 Canny with low threshold of 10

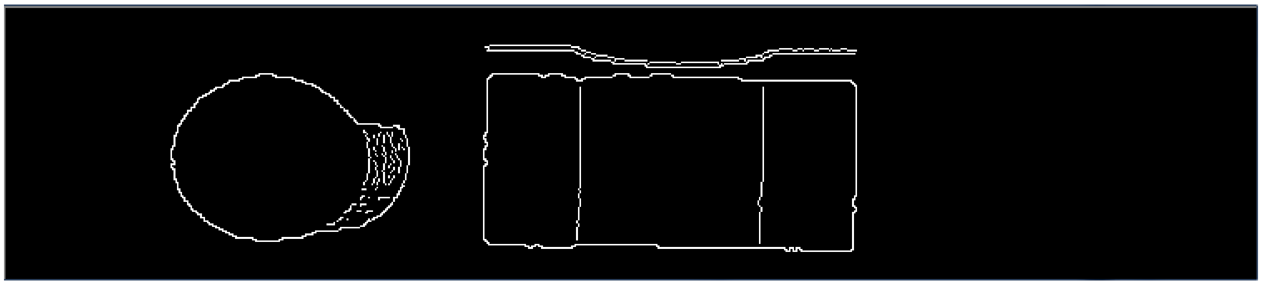


Figure 45 Canny with low threshold of 15

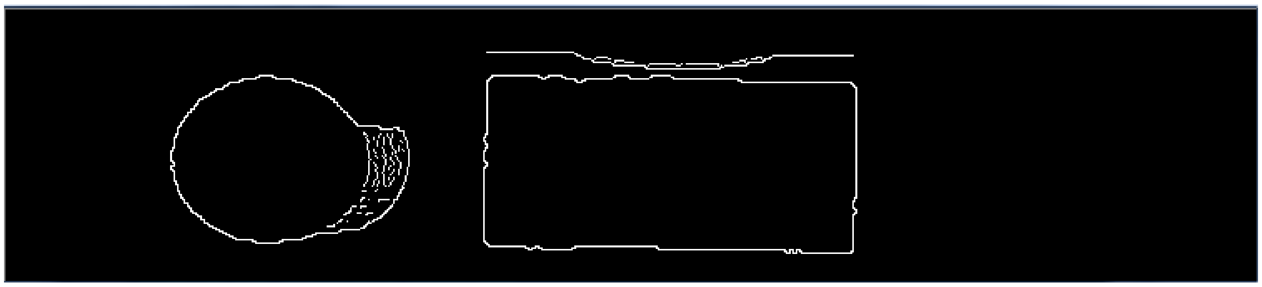


Figure 46 Canny with low threshold of 20

As it can be observed, when augmenting the threshold values, the noise is reduced; however, not completely eliminated. The above images are for a low threshold of 10, 15 and 20 respectively, and a high threshold of three times the low threshold.

Observing the resulting image, it was noticed that the image shows a perspective from the sensor, and it is not a flat Top View from the sensor. This is observed because of the section of wall of the cylinder that should not appear if it was a perspective-less Top View image. This means that all the values assigned for the wall of the cylinder should be projected to a single X and Y position, correcting the image to a proportional real life image.

For better results, new images need to be created, so it was decided to start the next phase of UR Programming, to enhance the process of data acquisition. The perspective problem will be continued on the integrating phase, after the UR Communication is programmed.

vi. Programming UR Communication

On this section, the process of programming the Universal Robot UR5 will be detailed. But before starting to program, the UR5 needs to be studied, its safety measures acknowledged and its commands understood.

1. Getting to know the UR5

The UR5 is a universal robot that has 6 joints; for this robot, this means 6 degrees of freedom. The first 3 joints of the robot (A, B and C) form the body of the robot, and the last 3 form the wrist (D, E and F). The robot is as shown below:

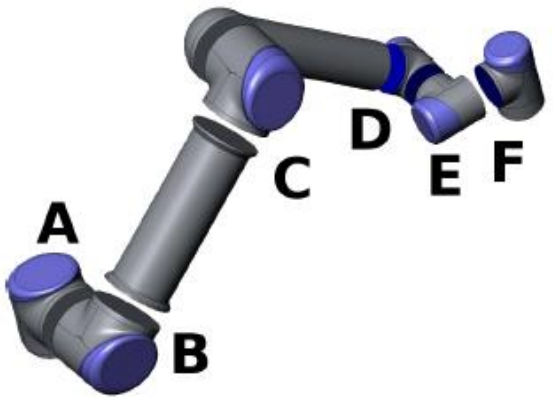


Figure 47 UR5 Joints

At the end of the wrist, different tools can be adapted depending on the application that the robot will be developing. The tool has a Tool Center Point (TCP) that will be the coordinate system of the tool. Each joint has its own coordinate system and with a combination of kinematics and electronics, the robot is able to move the TCP to a specific coordinate in space with a desired orientation. The only restrictions of movement of the tool are directly above and below the robot, and outside the workspace of the robot.

The UR5 has a reach of 850mm from the center of the base, which can be observed on the next image:

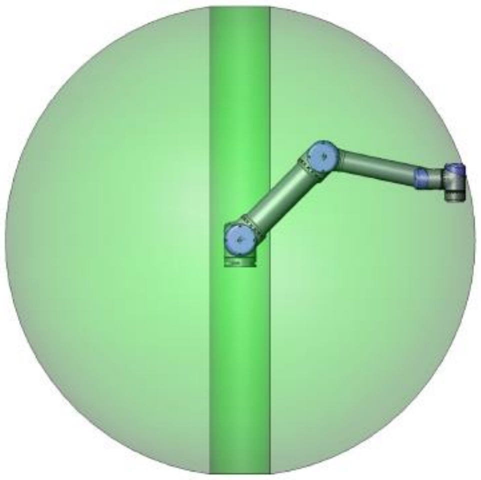


Figure 48 UR5 Workspace

The workspace is a sphere of approximately 170cm of diameter, excluding a cylinder volume that is formed above and under the robot base.

This workspace should be considered for defining the space where the robot will be working, and should also be considered as a security restricted area where no operator should remain, regardless of the programmed movements of the robot. An error on the programming or operation of the UR could severely injure a person. Also, the workspace is defined for a vertical floor mounting. The proper adjustments should be made when using a different mounting (Tilted mounting or roof mounting).

2. Safety of the UR

The UR5 has a safety interface integrated by two parts: the emergency stop and the safeguard stop. A summary of their differences is provided by Universal Robots (2012) and it's showed next:

	Emergency Stop	Safeguard Stop
Robot stops moving	Yes	Yes
Initiations	Manual	Manual or automatic
Program execution	Stops	Pauses
Brakes	Active	Not active
Motor power	Off	Limited
Reset	Manual	Automatic or manual
Use frequency	Infrequent	Every cycle to infrequent
Requires re-initialization	Brake release only	No
EN/IEC 60204 and NFPA 79	Stop category 1	Stop category 2
Performance level	ISO 13849-1 PLd	ISO 13849-1 PLd

Table 9 Safety interface

The Safeguard Stop can be activated by the UR when the motors of the joints are exceeded on a specific movement or torque, or when a risk of collision is detected. The Emergency Stop can be activated manually by the operator with the Teach Pendant's Emergency Stop button showed in the next image (The red button):

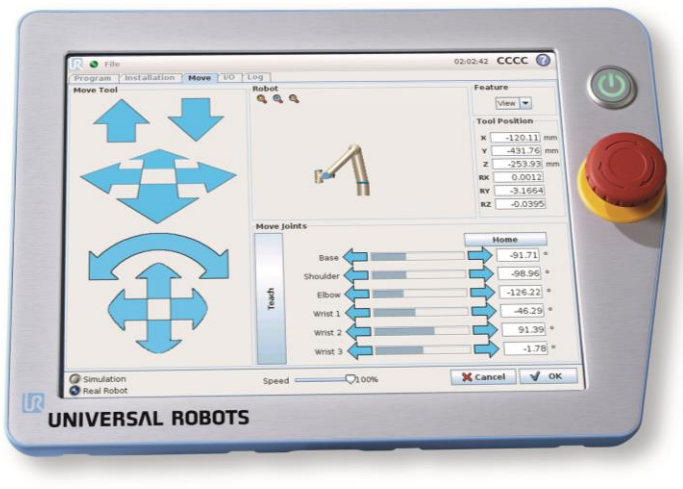


Figure 49 UR Teach Pendant and Emergency button

If it is required to move the robot joints on an emergency, the operator can press the back button on the teach pendant, which will activate a special mode of the robot. When presses, it releases the breaks on the joints and allows the user to manually move the joints.

It is highly important to train every operator on the safety use of the UR and how to proceed on an emergency.

3. Kinematics of the project

Kinematics is a branch of mechanics that involves movement of points, bodies and coordinate systems. It is widely used in the motion of robotic arms, as every joint works on its own coordinate system. To calculate the position of the TCP at a certain moment, a kinematic calculation from the base and all the way to the last joint should be performed.

For the project, not only the UR's coordinate systems are involved, but also the measuring surface's coordinates, the LPS's coordinates and the objects to be manipulated coordinates. Hence, a good managing of kinematics is required. The UR kinematics are performed by the UR's controller, and can be obtained with different functions that the UR's software provide.

Rotations can be calculated with the functions:

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Figure 50 Rotations' matrix

When using the Denavit-Hartenberg Notation, a rotation and translation from one coordinate system to another would be as follows (Sina, 2009):

$$T_{(i-1)i} = \begin{bmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & -\cos\theta_i\sin\alpha_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 51 Rotation and Translation matrix

Where:

- θ_i is the angle between axes x_{i-1} and x_i;
- α_i is the angle between axes z_{i-1} and z_i;
- d_i is the distance between the origin of the coordinate system and the common perpendicular of the next coordinate system;
- a_i is the distance between the two feet of the common perpendicular;

These values can be observed on the next image:

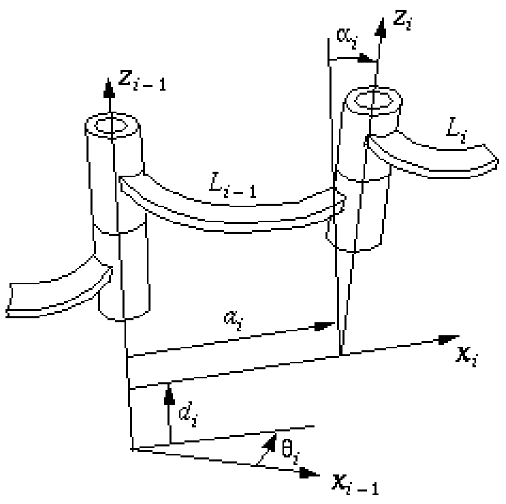
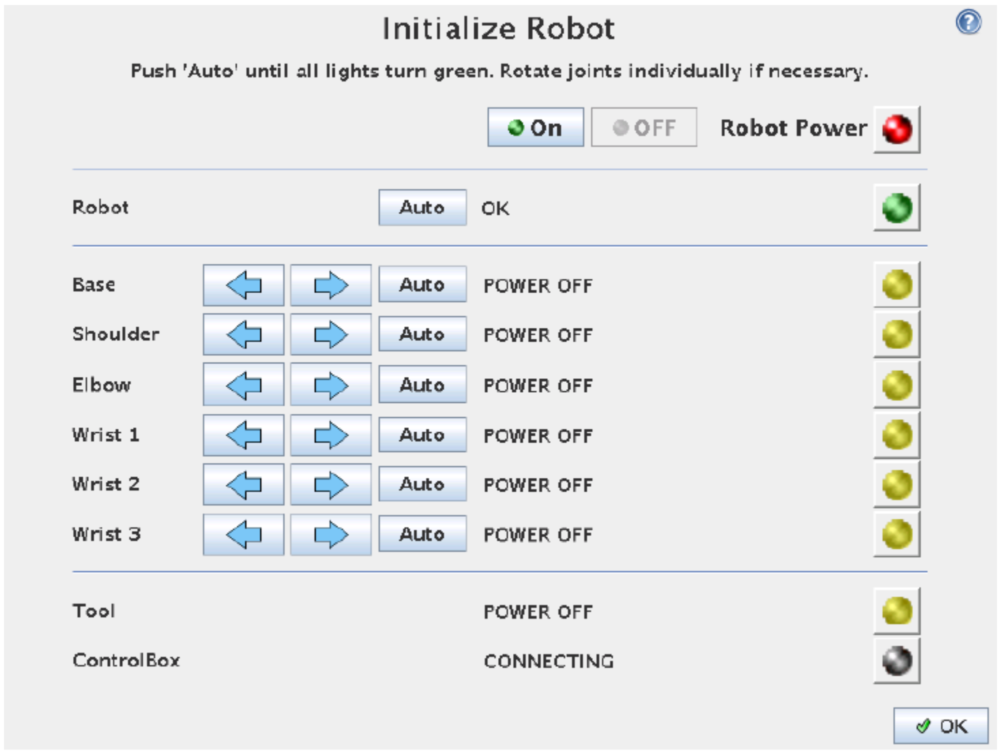


Figure 52 Joints parameters

The UR's controller realizes these calculations, and any external transformation should be made by the program.

4. Initialization of the UR5

When turning ON the UR5, a series of steps have to be taken, to initialize the robot, as the sensors on each joint need to be calibrated. When turning on the teach pendant, the following screen shows up:



Initialize Robot

Push 'Auto' until all lights turn green. Rotate joints individually if necessary.

On | OFF | Robot Power 🔴

Robot		Auto	OK	🟢
Base	⇦ ⇨	Auto	POWER OFF	🟡
Shoulder	⇦ ⇨	Auto	POWER OFF	🟡
Elbow	⇦ ⇨	Auto	POWER OFF	🟡
Wrist 1	⇦ ⇨	Auto	POWER OFF	🟡
Wrist 2	⇦ ⇨	Auto	POWER OFF	🟡
Wrist 3	⇦ ⇨	Auto	POWER OFF	🟡
Tool			POWER OFF	🟡
ControlBox			CONNECTING	⚫

✓ OK

Figure 53 UR initialization screen

On this screen, the steps to follow are:

- Power the Robot ON: The motors are powered, and the brakes released.
- Press Auto to calibrate the Joints: When pressing auto (For the robot or separately for each joint) the joints will move to some directions to calibrate and align the encoders of each joint. The user does not have control over these movements direction.
- Once aligned, the yellow indicators will turn green, and OK can be selected.

5. The tool

For this UR5, students from the area of Mechanics previously elaborated a Pneumatic gripper to be used with the robot. The gripper is the one showed on the next image:

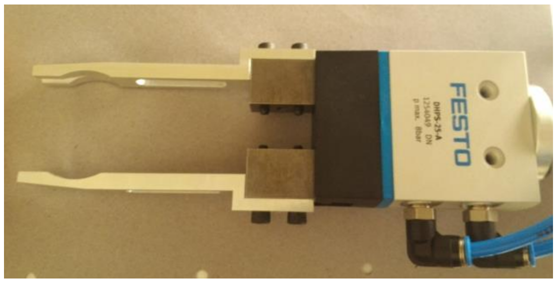


Figure 54 UR5 actual Gripper

As it can be observed, it uses a pneumatic Festo valve of double actuation, so the aperture and closure of the gripper can be controlled.

6. A first approach to handle LPS sensor

As it is wanted to continue with the Image recognition program, a first UR's program will be developed that enables the user to move the sensor with steps and control it manually. A further Ethernet communication will be developed in the Integration phase to make the process automatic.

To program the UR, the most common functions that the UR's software offers should be studied:

- MoveJ(), MoveP(), MoveL(): commands for moving the robot with certain characteristics, linear movement or fast route.
- pose_trans():Calculates the coordinate transformation between one coordinate system and another.
- pose_inv(): Calculates the inverse transformation between a point and the base coordinate system.
- get_actual_tcp_pose(): return the value of the actual TCP position from the base coordinate.
- Halt: Stops the program execution.
- Wait: Waits a determined time in seconds to continue executing the program.
- socket_open(): Opens a socket with specified port and the IP from the server (PC).
- Socket_send_string(): Send a string to the server.

There are other commands available; however, this are the ones the will be needed for the program purposes.

7. Define a plane

The UR's software allows defining features to make the programming easy. The user can define Points in space, Lines, and Planes. For this project, a plane is needed, which will be the table or measuring surface.

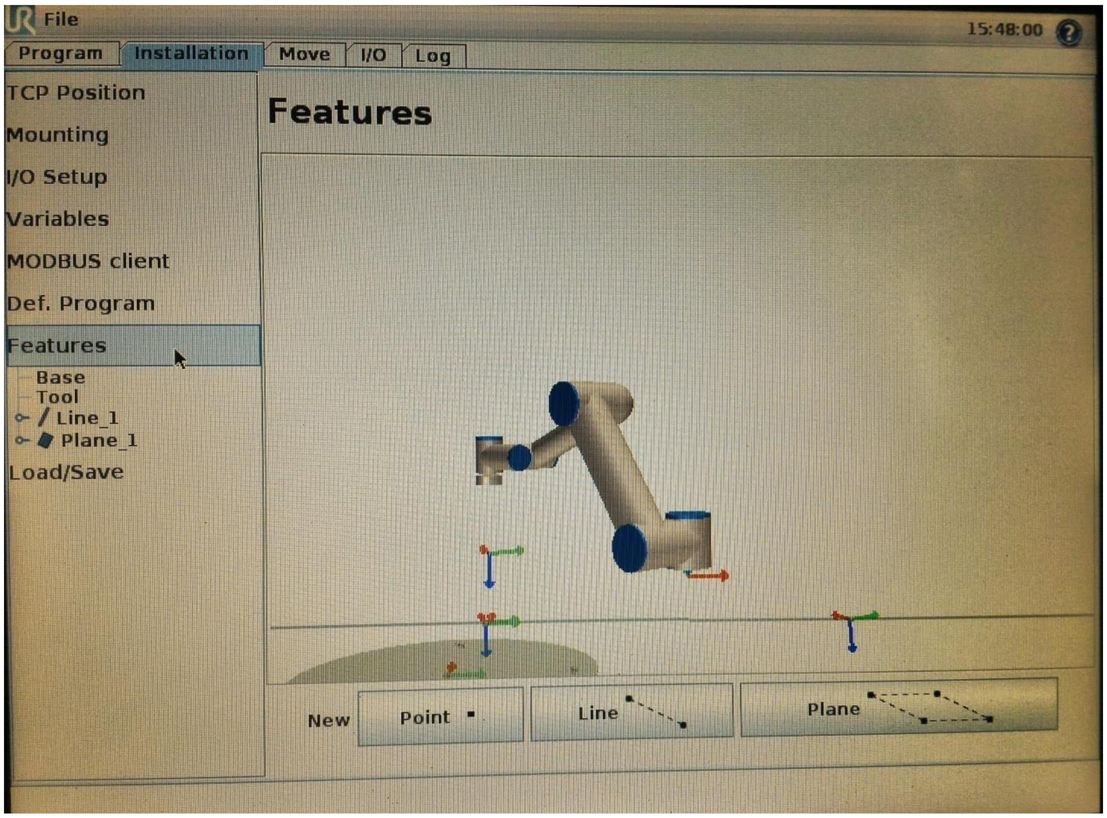


Figure 55 UR5 Features screen

The previous image shows the interface for defining a plane. Three points need to be defined, which can be taught to the UR by used the back button of the teach pendant, to enter Teach Mode. For this program, the points where selected in this order:

Point 1: The origin of the working plane.

Point 2: A point on the direction of the movement desired of the scan.

Point 3: Any point on the surface of the table.

The next image explains the disposition of points on the plane created:

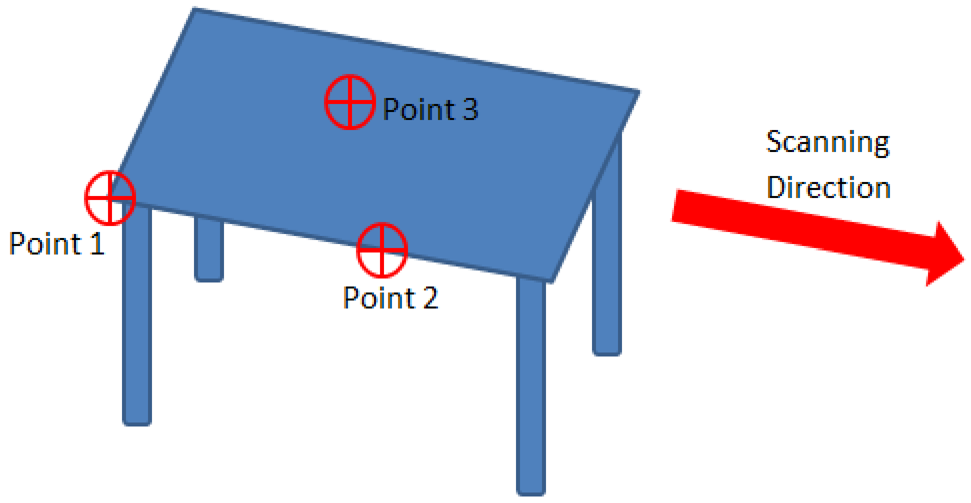


Figure 56 Defining the measuring plane

The coordinate system of the table will be located at Point 1.

8. Programming the robot

The next algorithm was developed to control the sensor:

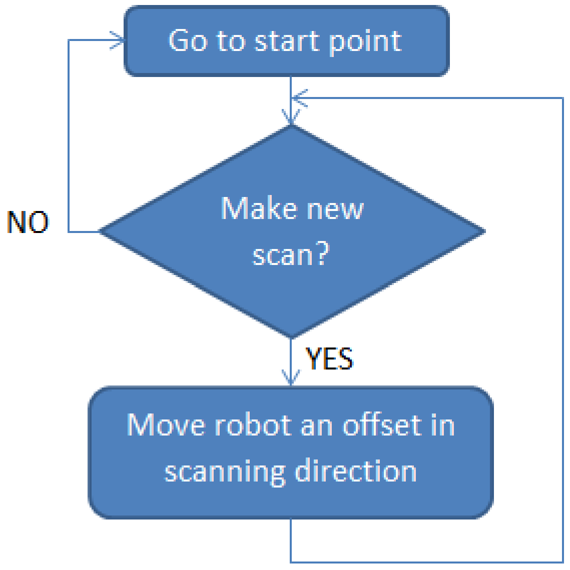


Figure 57 UR basic algorithm for a first scan

The algorithm is very simple, and allows the user to control with steps the movement of the sensor with a desired offset. In this case and offset of 1mm was established, to take images every 1mm and later build the 3D image.

The detail on this programming is that the robot moves in Base coordinates, so a transformation has to be done to move the robot linearly from the Table coordinates.

9. Results obtained

The next image was obtained on the first lecture:

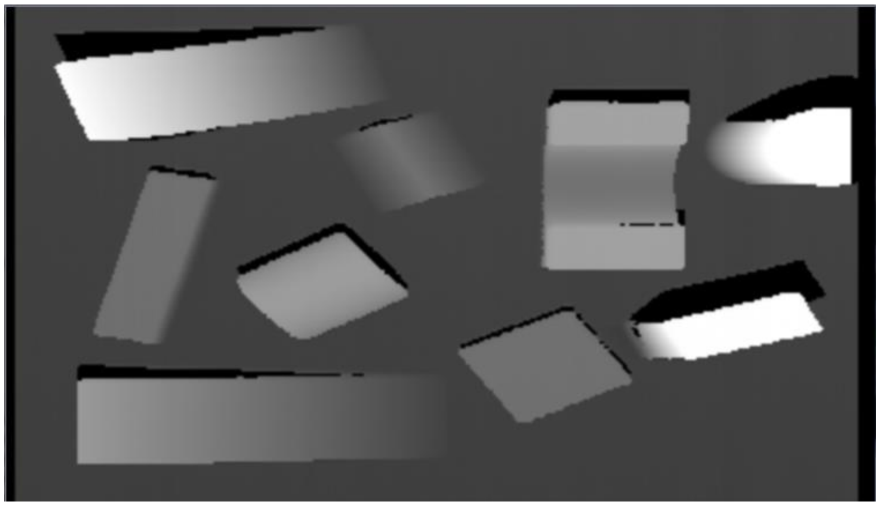


Figure 58 First scan obtained with UR

Using MeshLab tool, the same image can be observed in 3D:

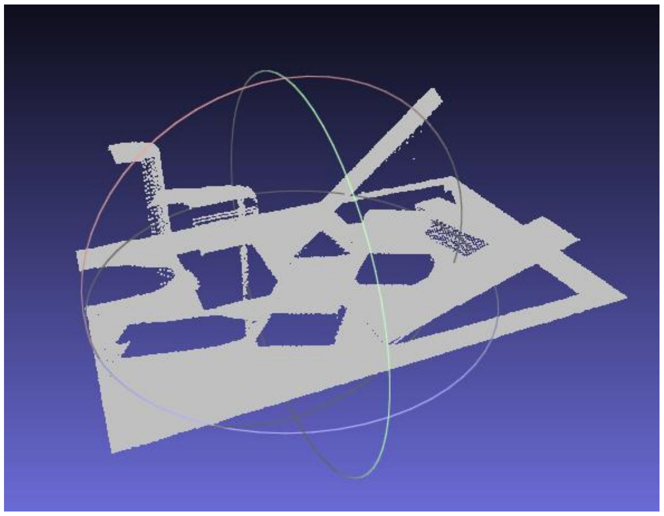


Figure 59 First scan obtained wit UR in MeshLab

10. Problem presented

When the sensor was installed on the UR, it was handled with no problem and the scan previously showed was obtained. However, after turning OFF and ON the UR, the Initialization of the Joints was done as it is normally done. The movements of the UR on this phase are not controlled by the user, and the cables were pulled off the sensor, as they were attached to the body of the robot. This resulted on the connectors of the sensor un-wielded from the sensor, and the sensor needed to be sent for repair back to Leuze Company.

The damage was documented on the following images:

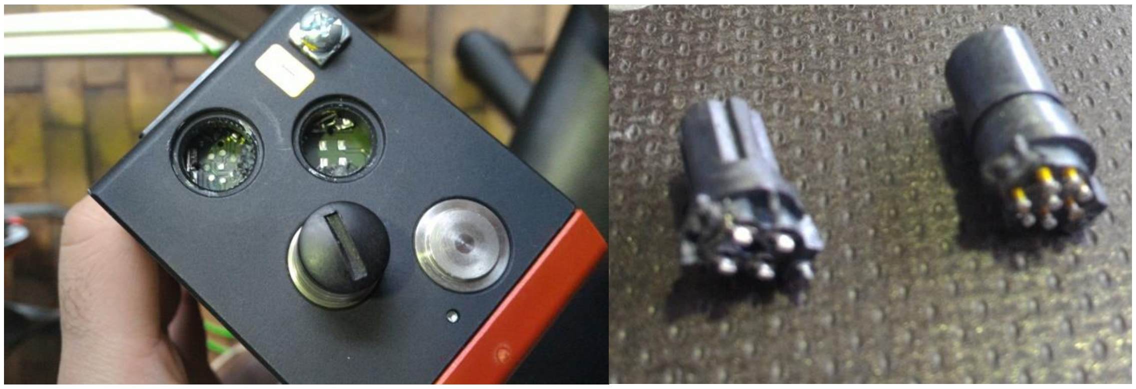


Figure 60 Damage to LPS's connectors

The process of repairing took from February 25, 2016 (Day of damage) to March 18, 2016, when the sensor was received back at the DHBW University. The receipt of repair can be found on the Appendix G.

11. Assessment of damage

The problem occurred because the cables that were previously installed on the UR for communication with the Leuze sensor were too tight. This caused that a different unexpected movement of the UR pulled the cables together with the connector, and tore them from the sensor. The integration of Leuze communication with the UR will be continued when the sensor is received back. For now, the Recognition program will be improved with the image that was acquired. Future enhancements will be made when more images can be obtained.

At this point, the UR algorithm is working properly, and it is time to continue improving the Image detection system. Also, working on parallel, the UR program will be modified to work by Ethernet and communicate with the PC. These next objectives will be explained on the Integration Phase.

vii. Integration phase

On the previous section, the following image was obtained:

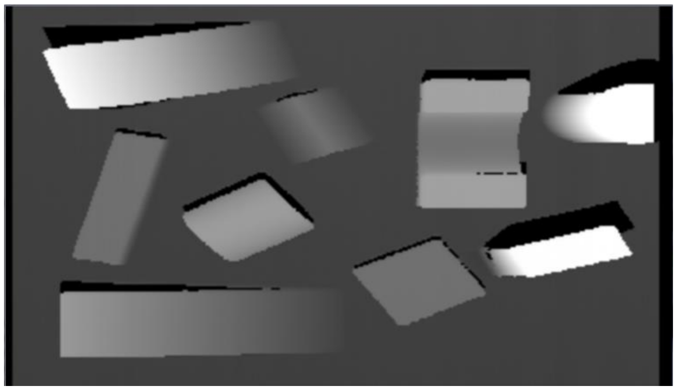


Figure 61 Image obtained with UR in grayscale

1. Flood fill

As it can be observed, the bottom of the image is not clearly distinguishable from some points of the objects. A flood fill algorithm was applied to identify the measuring surface, and separate it from the objects of interest. The next two images were obtained with different levels of flood:

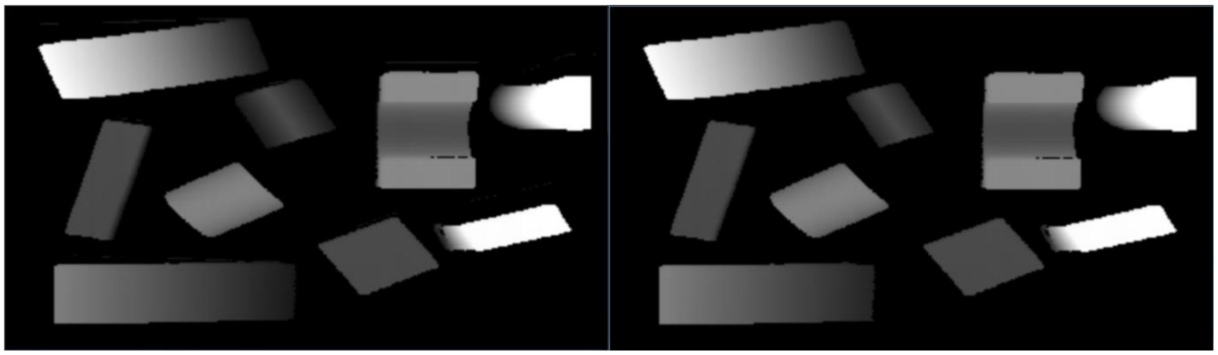


Figure 62 Scan with flood fill at different values

The second image uses a flood level of 15mm. It shows less noise on the image than the first; however, it is also appreciable the deformation of certain images, as is the case of the left bottom image, that corresponds to a ramp (tilted triangular prism). To avoid loss of information, the flood will be applied with a very low value.

Also, the deformation for perspective is very noticeable on the top right image, which corresponds to a cylinder. A solution for the perspective most be developed.

2. Perspective correction

To correct the perspective problem, a new image needs to be created, that has a different size. The size of the new image should be proportional to the area scanned, and then the Z values should be assigned to the proper X and Y coordinate.

To calculate the size of the new image, the old image needs to be evaluated, to find the maximum and minimum values for X and Y, and create a proportional but translated to the origin image. This may be easily to understand on the following image.

The image obtained is located as shown on a XY grid:

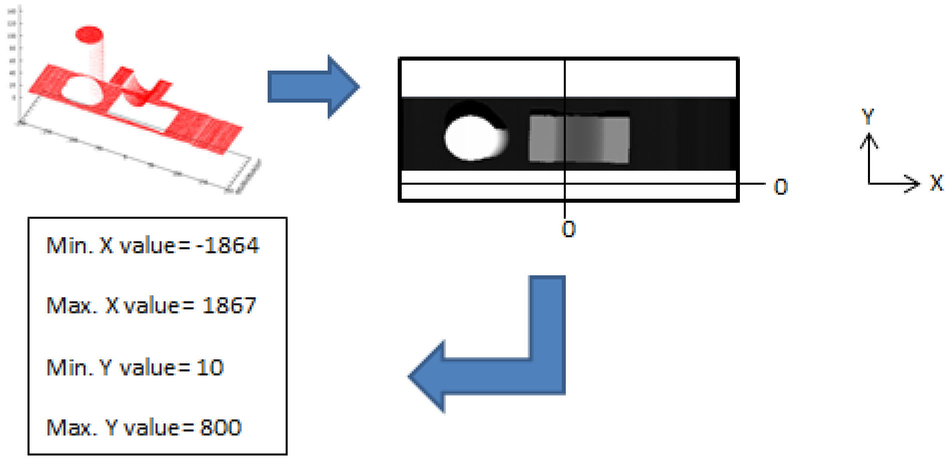


Figure 63 Original location of image in grid

And the new image should look like the following:

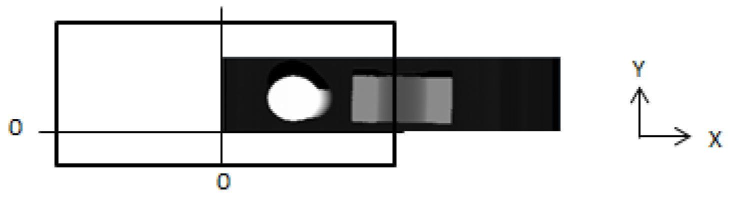


Figure 64 New location of image in grid

The image is translated to the origin, because images don't have negative pixels. Also, a rescaling of 10 was made. Originally, the units were 1/10 of mm; now, each pixel will represent 1 mm in real life. Multiple values of Z that correspond to the same X and Y coordinates will be overlapped, leaving the highest value of all, and eliminating the vertical values of the walls.

After all this transformations, the following image was obtained:

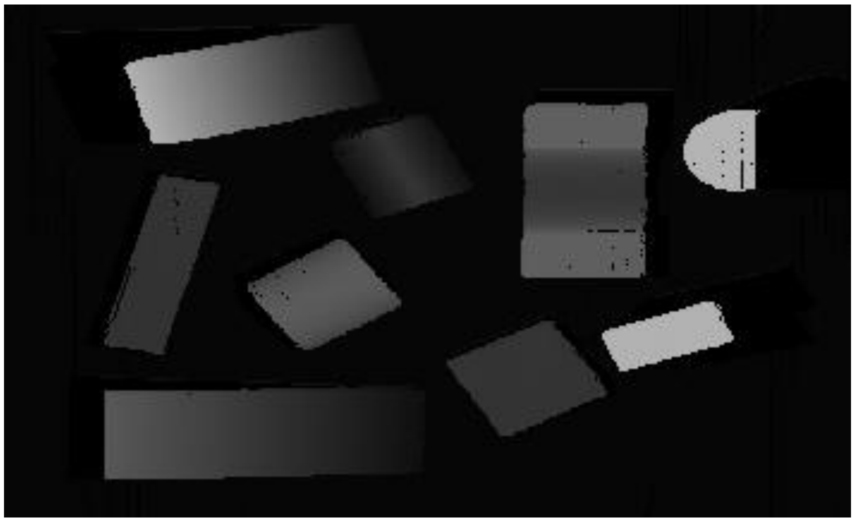


Figure 65 First proportional image obtained

As it can be appreciated, the perspective problem is not present on the image anymore, and the image is completely proportional to the real life objects. Applying the flood fill will obtain a better result as shown next:

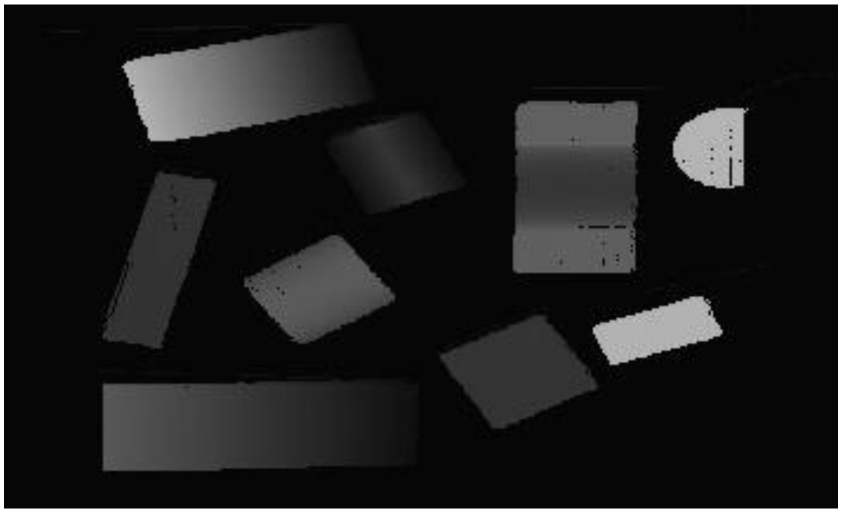


Figure 66 First proportional image obtained (with flood fill)

There is still some noise present on the image that will be treated later.

Applying a canny edge detector, the following results were obtained:

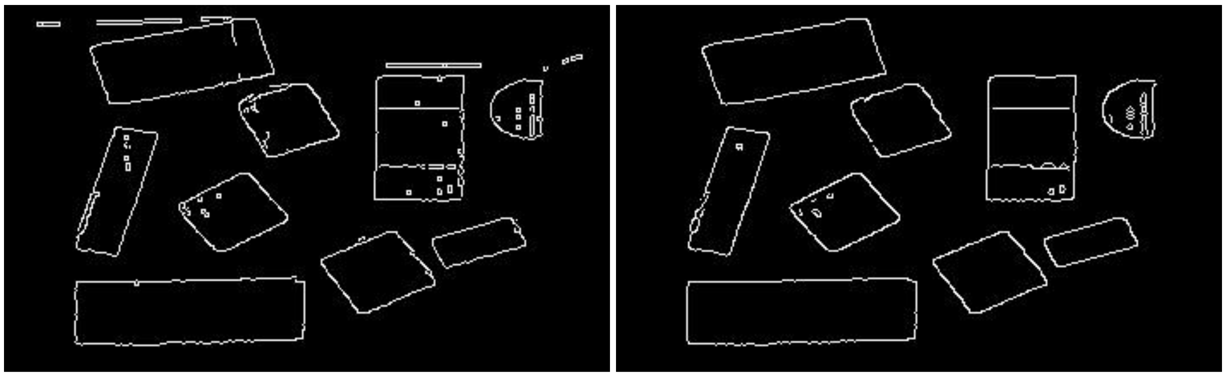


Figure 67 Image with Canny with two different values

The first image was a Canny applied to the obtained proportional image as it was. The second was obtained when applying the Canny algorithm after a Blur that was applied to the image. As it can be seen, it reduces the noise considerably. A blur is a mask that is applied to an image that softens the image by making an average on a pixel with the neighbors' pixel values. It is useful for reducing doted noise and to soften borders.

After applying the canny edge detector, a Contour detection algorithm has to be applied. This function evaluates the edges obtained and classifies them as independent contours when they are connected to each other or as separate contours if they do not belong to the same contour. The following image was the first attempt to identify contours:

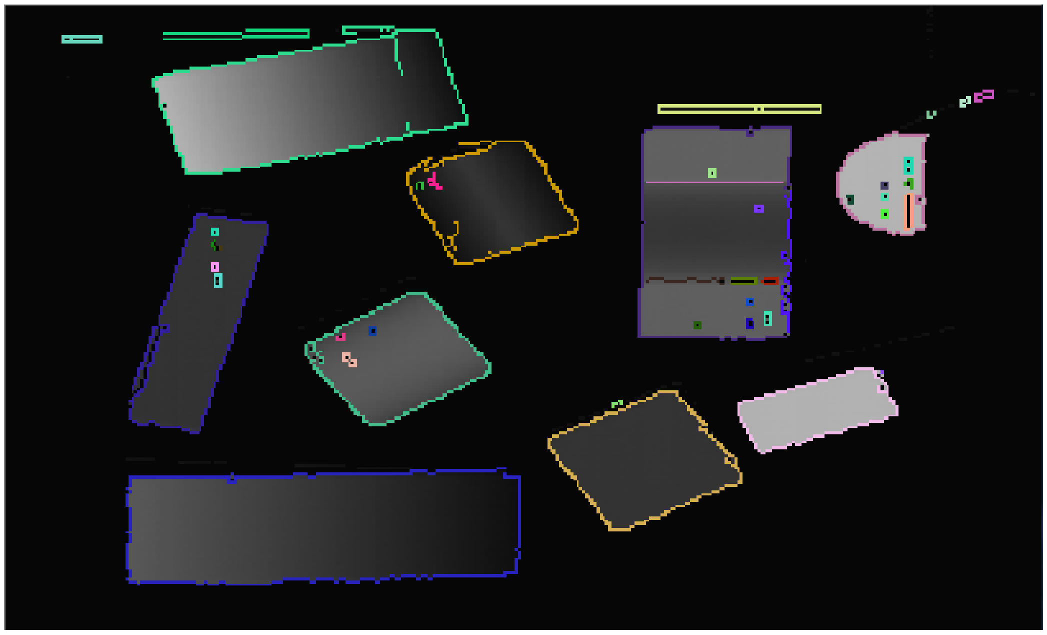


Figure 68 First Contour detector

Every color on the image represents a different independent contour. On this image, 85 contours were found. The objective is to reduce the amount of contours until every contour represents only the objects present on the image. To reduce this noise, a certain amount of filters has to be applied. The most common algorithms to reduce noise are blur, dilate and erode.

3. Filter algorithms

Blur: as explained before, it produces an average of the values per pixel with its neighbors, softening the image.

Example:

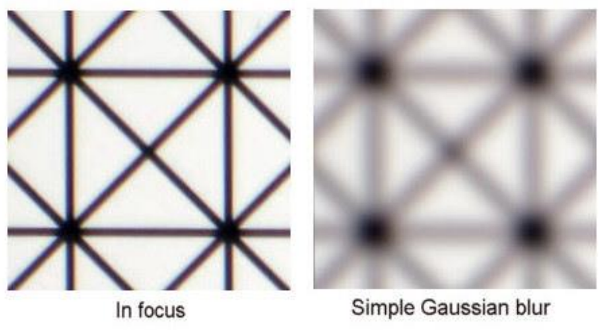


Figure 69 Blur example

Dilate: it increases the dark areas (or reduces the white areas) by applying a structuring element to the borders.

Example:

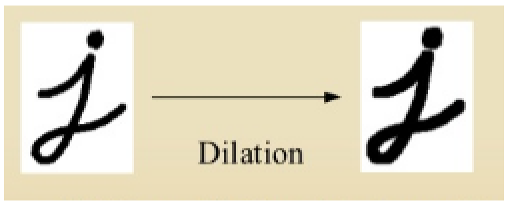


Figure 70 Dilate example

Erode: it increases the white areas (or reduces the dark areas) by applying a structuring element to the borders.

Example:

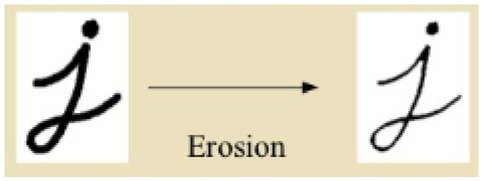
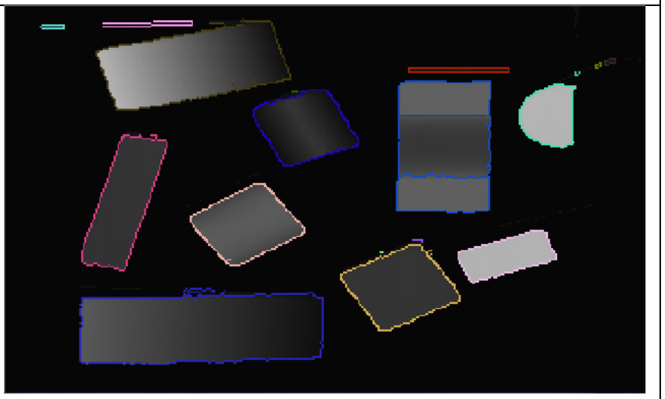
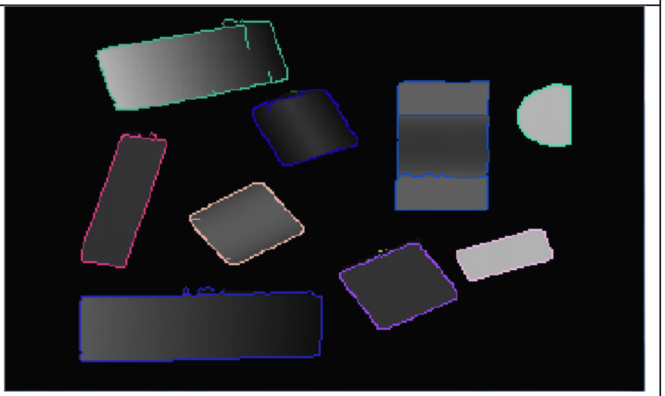


Figure 71 Erode example

It is not only important to understand what each filter algorithm does, but also in which order to apply them, as it will generate different results. The working image presents holes inside the objects that can be filtered applying dilation, but it also presents white noise on the outside of the objects that can be reduce applying erosion.

Different combinations of filters were applied, to obtain the best results.

Combination applied	Image obtained
Dilate-erode	
Dilate-erode-erode-dilate	

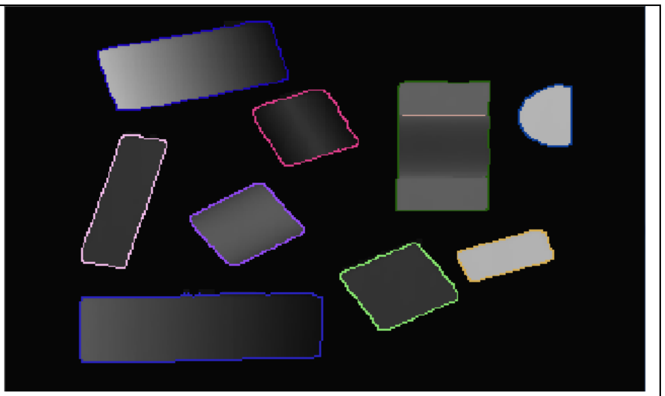
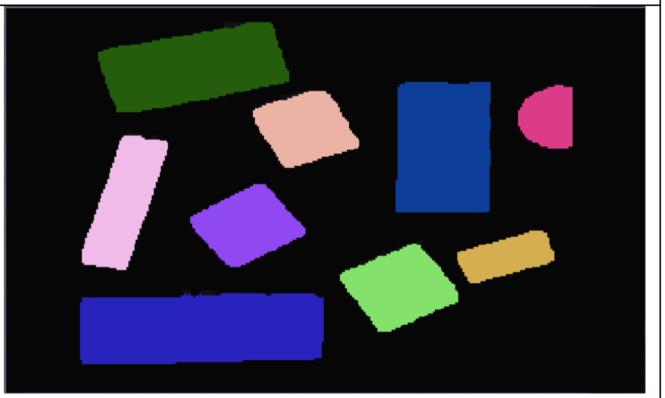
Dilate-erode-erode-dilate-blur	
Dilate-erode-erode-dilate-blur-fill	

Table 10 Different results obtained with filters

As it can be seen, very different results can be obtained when changing the combination of filters. The last image is obtained from the previous one, but represents the contours with a fill parameter. With this, the different objects where finally separated as different contours, as it ignores the contours inside other contours.

4. Poly Approximation

The Poly_approx() function was applied to improve the shape of the rectangle shaped objects. It calculates the best fit of a rectangle that contains the object. The following result was obtained:

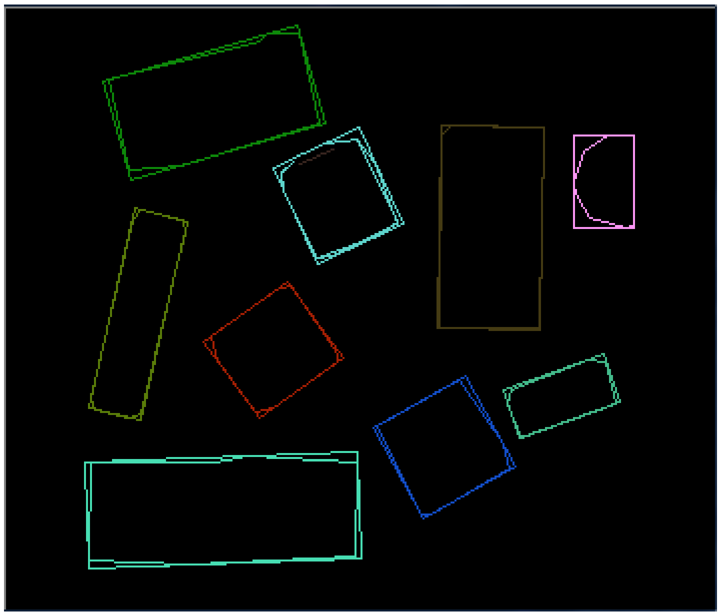


Figure 72 Approximation with Poly Approx

The new profiles found have a better rectangle shape; however; some of the resulting contours seem to be rotated a little, which is not desirable, since the final objective is to locate the objects with their X and Y coordinates and rotation. With this information, the UR should be able to handle the objects. It was decided not to use the Poly Approximation algorithm.

5. Best fit (circle and rectangle)

An algorithm for the best circle fit and best rectangle fit was applied to all the contours. The idea with this is to find the best fit, compare the areas of the obtained figures with the area of the original figures, and identify the shape of the objects.

The following image was obtained:

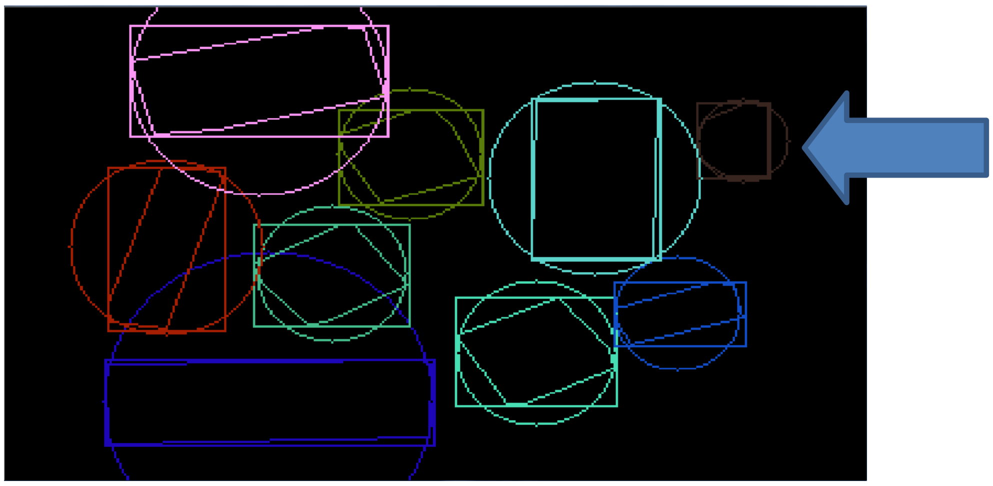


Figure 73 Best fit circles and rectangles

On the previous image, the top right figure is a cylinder (which was trimmed when measuring). The best circle fit shows that it located a circle where the figure was originally. On this image, the only circle-profile object was this cylinder, and it is trimmed because it was out of the sensors range when the scan was made. New images are necessary at this point to continue improving the classification algorithm.

6. Improvement of UR program

When the sensor was finally received back at the university, the work for integrating the phases was continued. The UR program was modified to establish a TCP connection and connect with the PC.

A sequence was defined for the UR and the LPS sensor. The algorithm was as showed next:

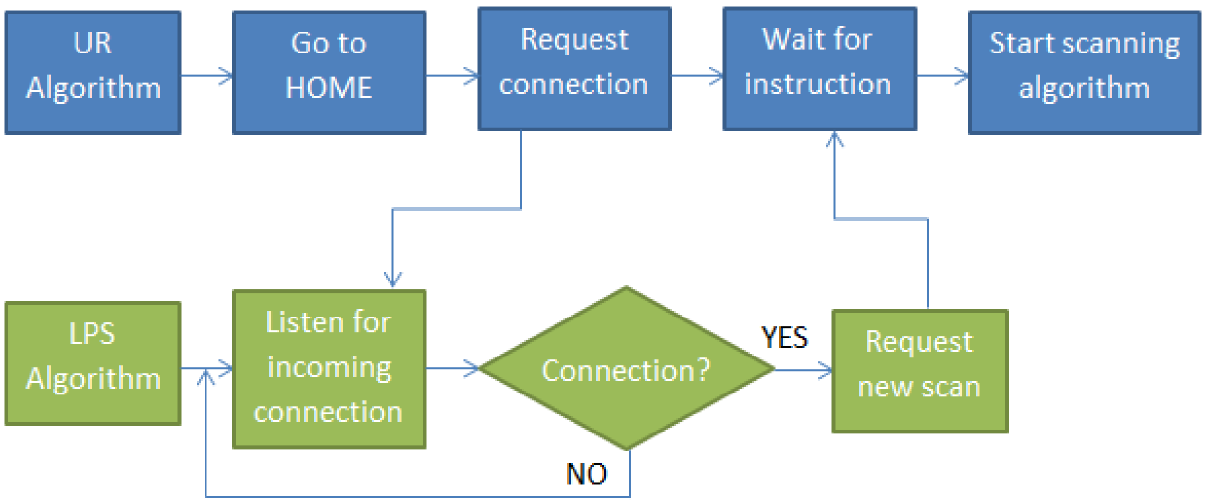


Figure 74 LPS and UR algorithm. Part 1

On this first part of the algorithm, the communication between LPS and UR are established. Once the connection is established, the LPS program requests to make a new scan, and the UR starts the sequence for this, that can be shown on the next part of the algorithm:

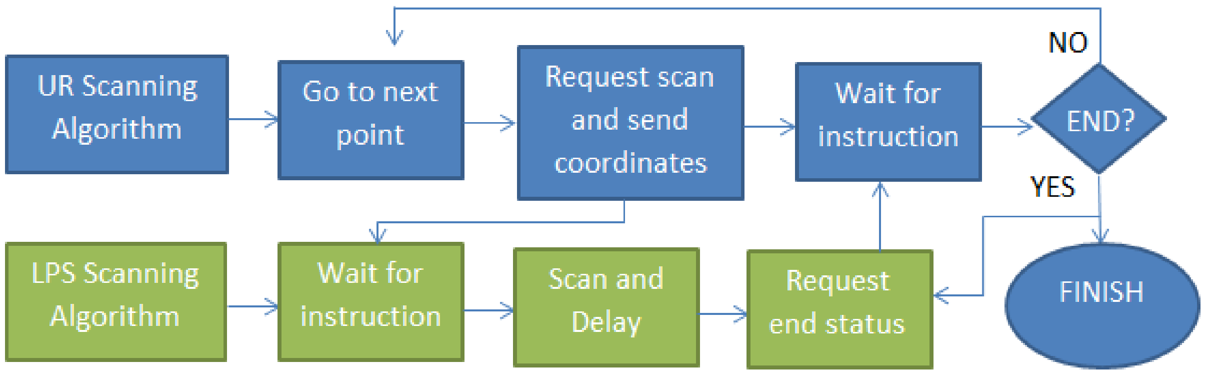


Figure 75 LPS and UR algorithm. Part 2

The algorithm might seem confusing at first sight. This is because two different programs are running simultaneously. But it's actually pretty simple to understand the flow of the programs. A brief verbal explanation is presented next:

Once the UR starts the scanning process, it moves the sensor to the starting point. When it reaches the desired position it sends a request for scanning to the LPS. The LPS was waiting for instructions, and once it receives the command it makes the new scan. Also, the UR sent the actual TCP coordinates, which will be used to locate the objects on the table. A delay is necessary in this case, because the sensor needs time to perform the scan before the UR starts moving again. Once the delay is over, the sensor requests the status of the scan. A total scanning distance was previously defined to the UR that the user defines at the beginning of the configuration. If the total distance is already scanned, the program is terminated. If it is not finished, the UR moves to the new position with the offset defined (until now is 1mm) and requests a new scan.

7. Speed improvement

The complete algorithm was taking about 2 minutes to perform a complete scan for a length of 60cm (600 lectures). To improve the speed, an offset of 2mm was tested. This means that the sensor will make a scan every 2mm. The image obtained is as follows:

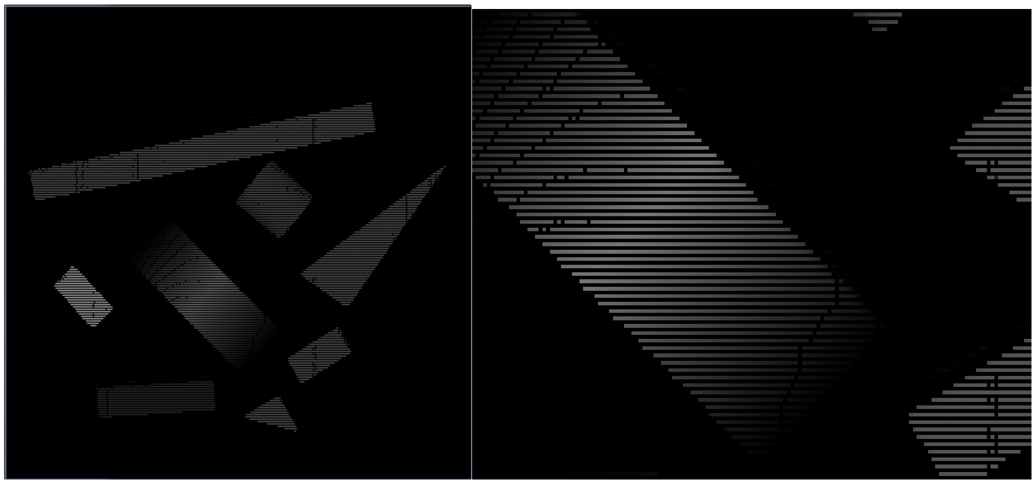


Figure 76 Scan 1 every 2mm

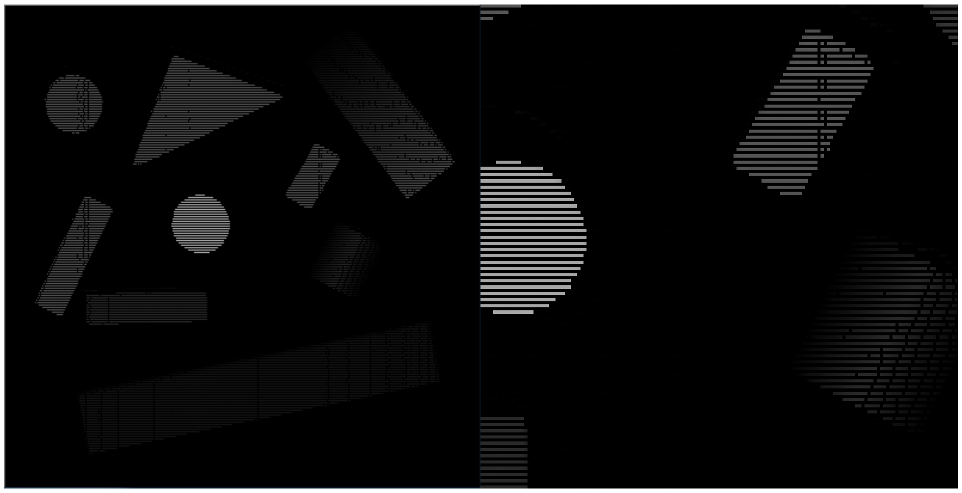


Figure 77 Scan 2 every 2mm

The first and third images are the images obtained every 2mm. The second and last images are a close up of the originals. As expected, the objects are now made by lines instead of been solid objects. This will affect the previous established filtering stage but with some adjustments it can be corrected.

After a Canny filter, followed by a dilate-erode stage, the following images were obtained:

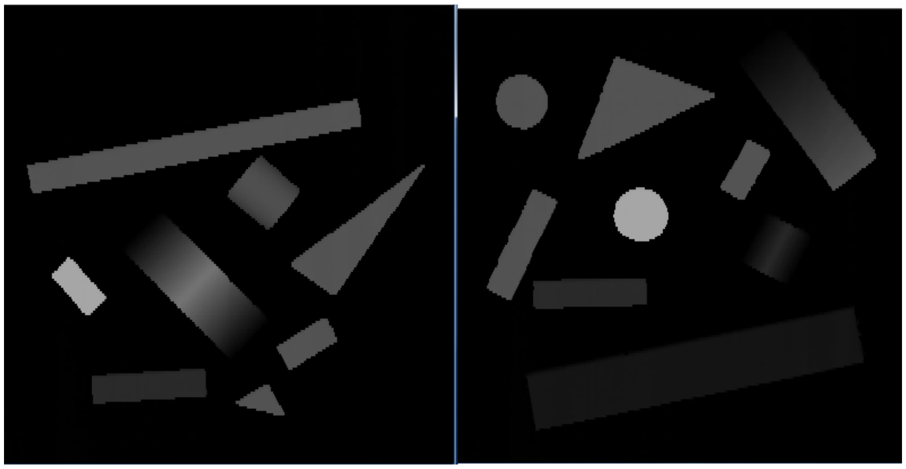


Figure 78 Every 2mm image filtered

The images are now solid images, and the borders were changed from the first to the second image shown below:

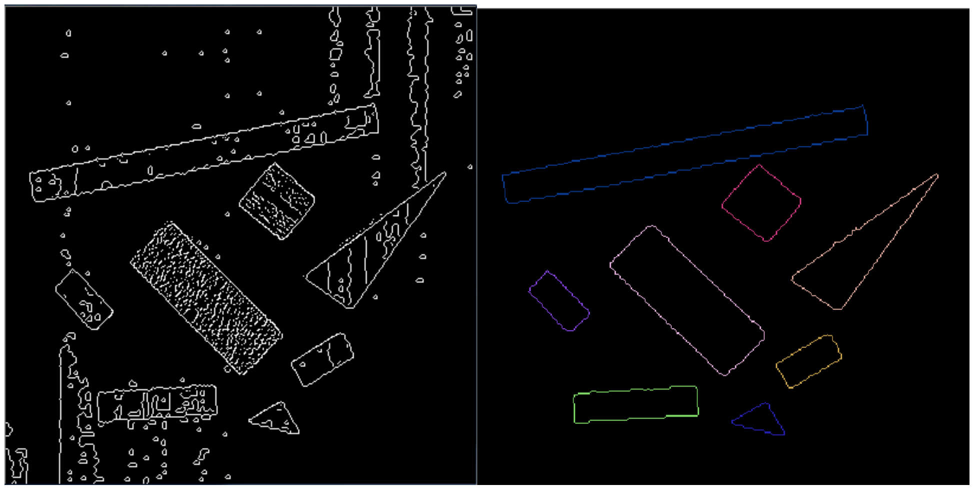


Figure 79 Canny and edge detector for every 2mm image 1

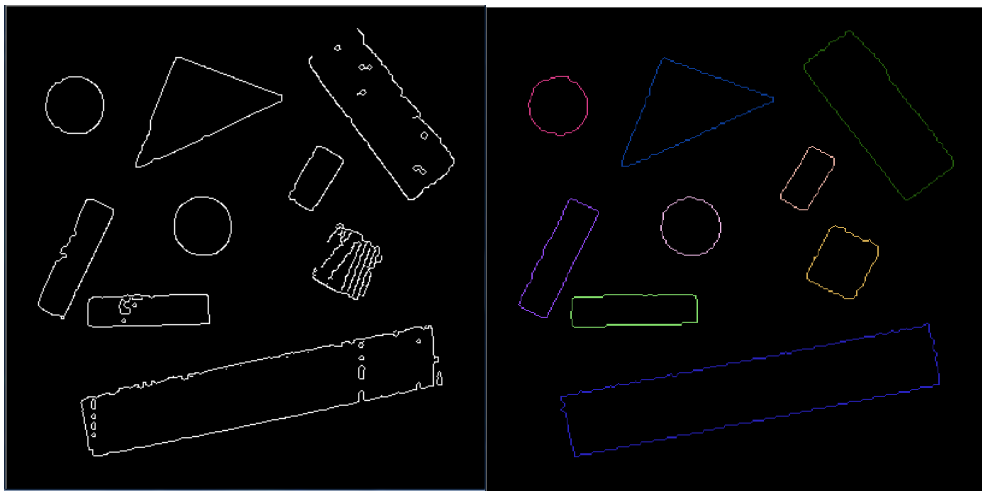


Figure 80 Canny and edge detector for every 2mm image 2

Note: More than one image is used to improve the algorithm as the algorithms may behave different on different images.

8. Best fit for rectangles, circles and triangles

As the objects that are handled have mainly rectangle, circle and triangle profiles, and algorithm for detecting the best fit on this figures is developed. The following results are obtained:

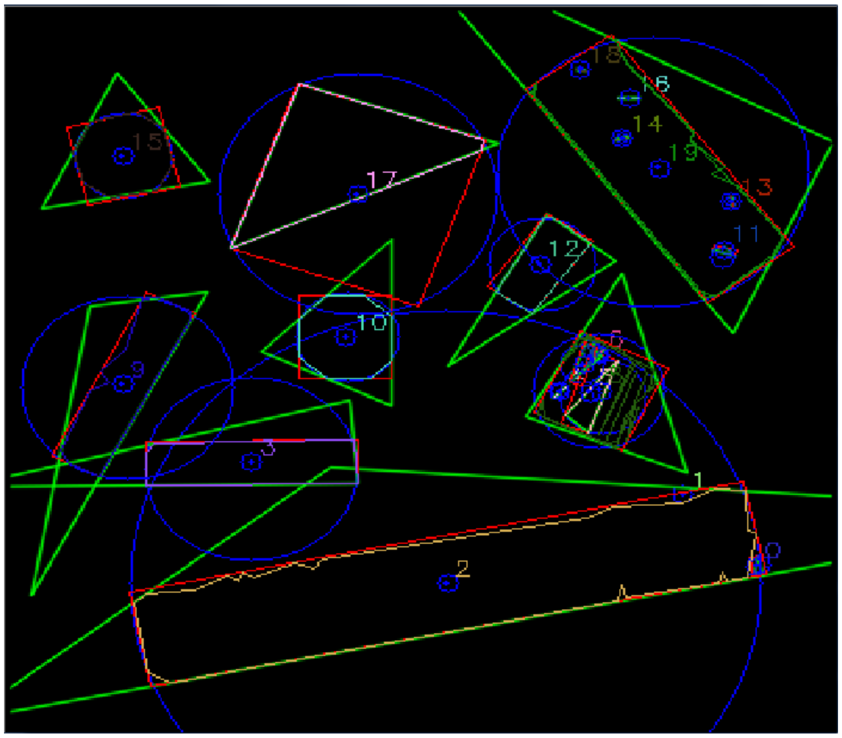


Figure 81 Best fit for rectangles, circles and triangles

On every object has now been drawn the best fit for circle, triangle and rectangle. This will be used to characterize each object.

9. Comparing areas

The idea of finding the best for circle, rectangle and circle is to compare the area of all these best fit figures, and compare it with the area of the figure. All the areas were calculated, and compared, and it was found that when the error was below 20% of the original contour area, the figure coincided with the one evaluated. An example is explained next.

The following image has several objects, some of them rectangles and a circle:

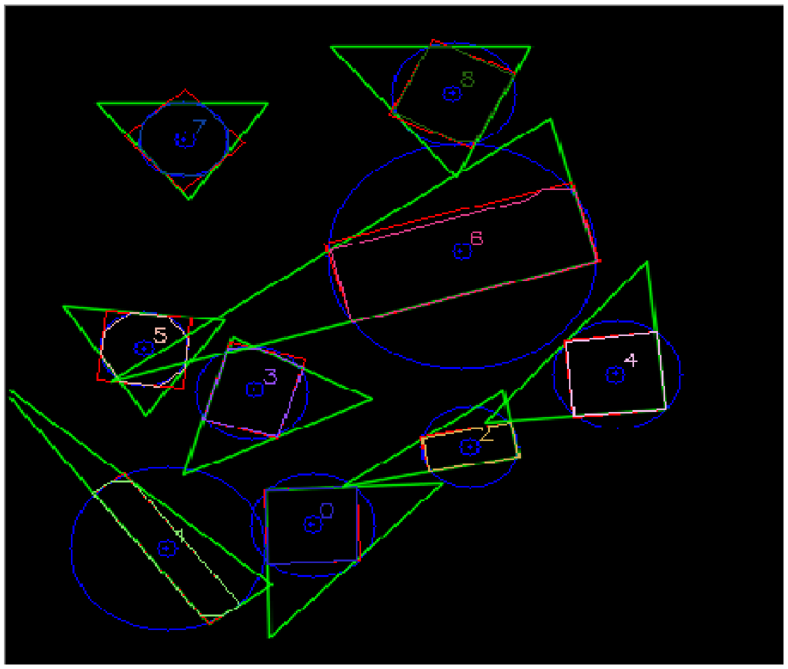


Figure 82 Numbered objects

The objects are numbered from 0 to 8 to differentiate them.

The following results from calculating areas were obtained for figures 0 and 7:

Area	Figure 0	% Error	Figure 7	% Error
Original Figure	2048.5		1563	
Best fit Rectangle	2115	3.25	1893.13	21.12
Best fit Triangle	4050.09	97.7	2524.71	61.53
Best fit circle	3325.39	62.3	1777.12	13.7

Table 11 Areas comparison

The results of the error percentage on each figure show that the lowest error is the one that corresponds to the best fit. Also, after several objects analyzed, it was conclusive that in most cases the error percentage was not higher than 20%. This is useful because if the 3 errors are higher than 20% error, it can be conclusive that the figure is neither rectangle, triangle nor circle.

In the examples mentioned before, Figure 0 is effectively a Rectangle, while Figure 7 is a circle.

10. Analyzing gradients on each object

Now that the objects are classified between circles, triangles and rectangles, it only remains to sub-classify the rectangles. The following figures can present a rectangle profile when observed from a top view:

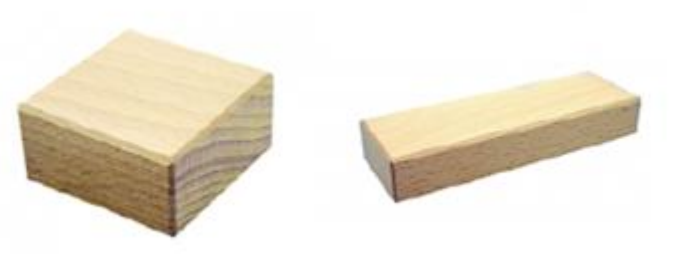
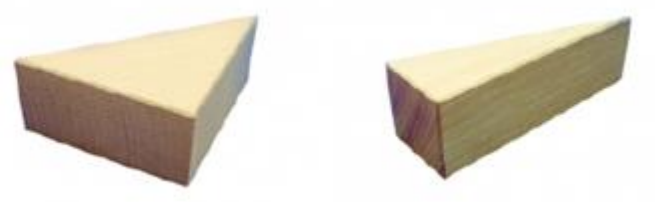
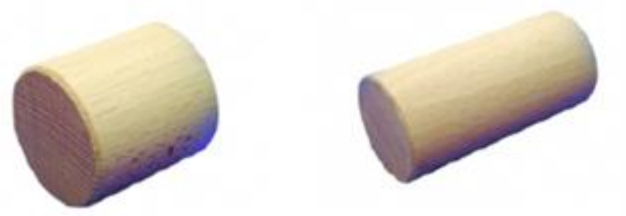
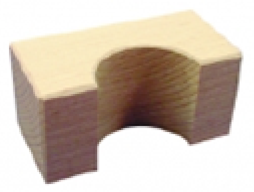
Any prism laying down on any face	
Pyramids when they are not laying down on their triangular faces	
Cylinders that are not laying on their bottom	
Different pieces that have a rectangular top view	

Table 12 Possible rectangle shaped profiles

This analysis is based on the pieces acquired from the university to work on this and other projects; however, different profiles may be found in real life that would require a different approach for identifying.

To distinguish from all these rectangular profiles, a gradient analysis needs to be made.

Pyramids, for example, will have a constantly growing gradient as shown in the next images. The first one is like a simple ramp; the second one is a double ramp.

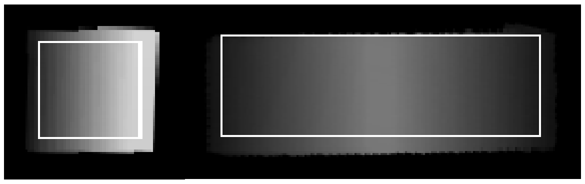


Figure 83 Pyramids

Prisms will have a constant value, with a null gradient as in the following image:

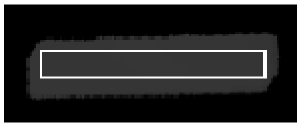


Figure 84 Prism

Cylinders will present a growing and decreasing not constant gradient:

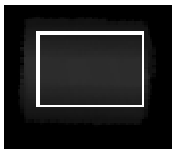


Figure 85 Cylinder

For making the gradient analysis, first of all each figure was isolated and rotated to end up with a straight single figure. After this, a gradient analysis was made in both X and Y direction, to determine what figure it is, and which orientation it has.

The process is as follows:

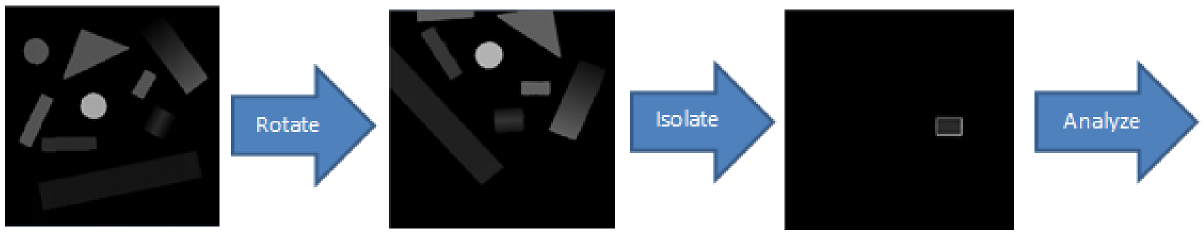


Figure 86 Gradient analyzing procedure

After obtaining all gradients, the program is able to differentiate objects. An extra feature was added to the program that allows to click on a figure and obtain information as showed on the next image:

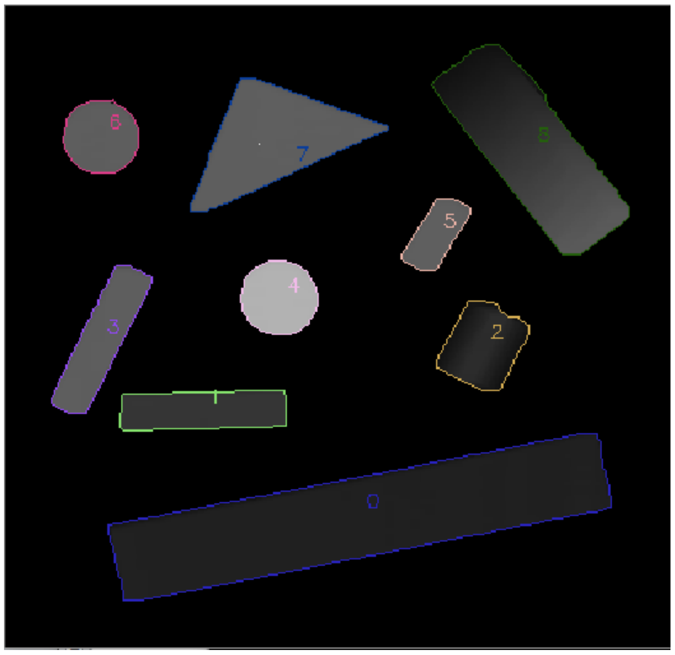
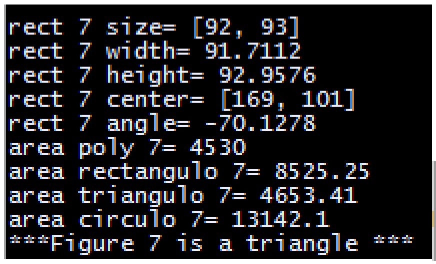
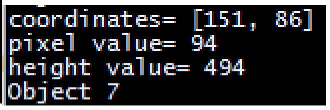


Figure 87 Final image for user interface

This is the image that the user handles, and when making click to an object, for example Object 7, the following information is printed on screen:



```
rect 7 size= [92, 93]
rect 7 width= 91.7112
rect 7 height= 92.9576
rect 7 center= [169, 101]
rect 7 angle= -70.1278
area poly 7= 4530
area rectangulo 7= 8525.25
area triangulo 7= 4653.41
area circulo 7= 13142.1
***Figure 7 is a triangle ***
```



```
coordinates= [151, 86]
pixel value= 94
height value= 494
Object 7
```

Figure 88 Information printed to user

The coordinates are showed to the user, and the real value of the height in mm.

If the user wants to take an object, the coordinates have to be transformed first to sensor coordinates (Inverse process as the one explained in the "Perspective Correction" section). Once that the coordinates are known in the sensor coordinate system, another transformation needs to be done, because the sensors coordinate system is different than the one of the TCP. For the case of the gripper used, it needs to be added an offset of 60mm in X and 210mm in Y; however, this value depends on the gripper used, and needs to be calculated by the user when installing the LPS.

XI. Solution's Analysis

The solution developed is very wide, and has many details. As it was proposed at the beginning, it is divided in 3 main phases:

1. Leuze sensor programming
2. UR Programming
3. Image recognition programming

i. Leuze sensor programming

The LPS program is in charge of making the scan of the surface to be analyzed. It also manages part of the communication between the sensor and the Universal Robot. From the Leuze interface, it should be possible to configure certain parameters of the sensor that give the user a widespread amount of opportunities on different scenarios.

Some advantages of using laser depth sensors instead of cameras for object detection and identification are that the ambient light is not as problematic as with cameras, hence the working area doesn't need a controlled light ambient. Also, the possibility to vary the laser exposition allows the user to quickly adapt the sensor for working with different materials on the objects to be detected, from dark objects, to metallic bright objects.

The interface for communicating the sensor with the UR is satisfactory, and the sensor is able to scan surfaces in a proper mode. The result of the scanning process can be evaluated when observing the images obtained. The time of scanning however is high and should be improved in a next level of the project. The sensor uses a delay when making each scan and this makes the code slow. A proposition for solving this problem will be proposed in the Recommendations and Future work's section. The interface for the program is console based, and could be improved to make it more user-friendly. An idea for the interface will be proposed at the Recommendations and Future work's section.

ii. UR Programming

The UR program is in charge of moving the sensor over the scanning area. It needs to communicate with the LPS sensor to coordinate scanning and positioning, and to exchange information such as actual position of the TCP, or the LPS sensor if the conversion is made.

The Universal Robot software is very flexible for programming. The Ethernet connection is easy to program, and does not need to take the programming to a low level to achieve most of the work. If special programming is needed, UR offers a variant of the programming by inserting Script code. A complete manual on Script programming can be found at Universal Robots (2013) and Zacobria Lars (n.d.). The Ethernet communication on this project was done using script code. The rest of the programming was done by using the functions that the UR offer from the teach pendant.

The resultant program is effective and does what it is supposed to do. The same issue as for the Leuze program is the time it takes to make a full scan. They are both dependent and to improve one, the other should be improved as well. The actual code moves the UR step by step, and this makes it slow, and also creates a small vibration on the robot, which could introduce some noise on the scan. A solution will be proposed together with the Leuze's solution on the Recommendations and Future work's section.

iii. Image recognition programming

The image recognition phase could probably be the most important of all the three as it is the one in charge of interpreting the data acquired from the Leuze sensor. The First two phases are in charge together of acquiring the data, but this last phase is the one that defines if the project fulfills its objectives. The main objective is both acquiring the 3D images, and interpreting them for future manipulation.

The transformation from a 3D Point Cloud to a 2D Grayscale image is very useful to analyze data with image recognition software. The possibilities with open-source powerful software as is OpenCV are very wide, and the amount of information, forums and tutorials that can be found on the web is very extensive. The solution developed covers the objectives desired: to detect objects, identify and characterize them and finally locate them in space. The project was based on a variety of objects that can be found on the appendix C, and similar objects will be also identified by the program. However, if the intention is to identify any object at all, a different approach needs to be made. An alternative solution will be proposed at the Recommendations and Future work's section that uses the Point Cloud Library (PCL) together with OpenCV.

Some errors were detected when two objects from similar height were very close to each other. The program would detect both objects as one, because there is no gradient variation between each. This can be observed in the Appendix E. A possible solution would be to make the UR move the container of the objects, to re-arrange them and make a new scan.

iv. Integration phase

The three phases that make the project (Leuze program, UR program and Vision program) were integrated to function together, having in consideration the needs of the other phases. The results were good but they can improve, mainly in time.

As an additional but optional objective, it was proposed to make a final program with the UR receiving the coordinates of the objects, and picking them in their place. The programming and knowledge behind this is already implicit in the other programs, as the coordinates just need to be sent to the UR (data exchange has already been done), and the UR needs to position on a specific coordinate (this was done in the scanning phase every time the robot places on the next scanning position). The plan was to develop this program, but with the time lost on the repairing of the Leuze sensor, the time left was just enough to finish the other objectives, hence it was not developed.

v. Final cost of the project

The cost of the project was initially calculated with more elements than needed. After developing the solution, a more accurate calculation can be done.

Item	Price (€)
UR10	23870.00
UR5	17870.00
Gripper	1378.00
3D Image Recognition	9714.00
Fittings, small parts	2000.00
Total	54832.00

Table 13 Real budget of the project

This total is calculated including both UR5 and UR10 robots; however, if a more low cost solution was required, using only the UR5 would give a total budget of €30962. The main inversion is in the Universal Robot and the Leuze LPS sensor, but no big investment was made for license on image processing software.

XII. Conclusions

The use of light section sensors is optimal for acquiring 3D images on scanning surfaces, as it is very flexible to program, and offers the user a wide range of possibilities of configuration for obtaining the desired results.

The use of an assisting robot, in this case the UR5, to assist a depth sensor on realizing a scan over a measuring surface is very useful. It allows the user to make complicated scans with a high precision when a transporting band is not an option to move objects.

Vision systems are very powerful for analyzing data, and the extent of their scope is tied to the user's imagination and programming skills. An open-source tool such as OpenCV gives the user more than enough options for developing object identification, characterization and image treatment, but a background on image processing is required.

The integration of different areas such as Sensor programming, Robotics programming and kinematics, Image processing and others are a very useful quality for mechatronics, as they allow the user to build a wide variety of solutions that can be applied on the industry with success and for a large amount of applications.

The use of open-source software is growing every day, and the possibilities with these tools are getting bigger. This type of software generally gets support from the users, and the cost benefits are admirable.

XIII. Recommendations and Future work

As the solution developed is for a university for research purposes, the DHBW wants to continue doing research and improving the results obtained from the realization of this project. Hence, a series of recommendations will be listed as ideas to be developed and enhancements that can be made to the solution obtained.

i. Improvement of time

As it was mentioned previously, the time for scanning is not very good, as it takes approximately 1 minute to scan a surface of 50cm. This is because of the logic that was used all along the programming of the algorithms.

A proposed solution is to make the Leuze sensor scan continuously and the UR also move continuously from the start point to the end point. An algorithm for managing the data sent from the sensor to the PC needs to be programmed, probably using a buffer that stores all the data and then stores it properly. With no delays on the scanning process, the scanning time should be reduced drastically, probably to less than a 10% of the actual scanning time.

Another proposition for improving time is to make a coarse first scan that will get the location of objects on the beginning, and after locating the present objects over the scanning surface, a smoother scan can be made locally to obtain more clear scans of each object. This will also reduce the time of scanning, as it uses most of the time only where there are objects present, and very little time where no objects are found.

ii. Occlusion solving

There is no need to have multiple sensors to avoid the occlusion phenomenon because the sensor is not fixed to a static surface. The advantage of having the Leuze sensor fixed to a Universal Robot is that it can change position and orientation easily. If a complete 3D image wants to be acquired, it's only a matter of making a scan on one direction, rotating 180°, and making one second scan. The sensor can even be rotated to obtain a different degree of detail. It will just need a kinematic transformation of coordinates on each position and orientation of the sensor to fit all the images taken.

iii. Use of Point Cloud Library

PCL is a powerful tool that will allow treating the data as it is taken by the Leuze sensor. It only requires point coordinates on a 3D space to analyze data. PCL has algorithms for stitching more than one point cloud, which would allow making scans to an object from different directions, and putting them together in only one image. With this, it is even possible to create complete 3D images of objects that can later be used to reproduce the objects with 3D printers or CNC machines.

iv. Use of Qt to create an IDE

The result produced is mainly for a user that knows of programming, as it is console based. However, using a tool as Qt enables the user to create IDE's that can realize the same functions as the program, but with a friendly-user interface. The configuring of the

Leuze sensor can be handled from this interface, and it should be possible also to display the results in 3D to the user. With this, the operator would not need to worry on entering commands, but would only see a final graphic interface program.

XIV. Bibliography

Henry, P., Krainin, M., Herbst, E., Ren, X., y Fox, D. (2010). RGB-D Mapping: Using Depth Cameras for Dense 3D Modeling of Indoor Environments. Experimental Robotics, 79, 477-491. DOI: 10.1007/978-3-642-28572-1_33

Schwarz, B. (2010). LIDAR: Mapping the world in 3D. Nature photonics, 4, 429-430. DOI: 10.1038/nphoton.2010.148

Valencia, R., Teniente, E., Trulls, E., and Andrade-Cetto, J. (2009). 3D mapping for urban service robots. In Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on (pp. 3076-3081). IEEE.

Gokturk, S. B., Yalcin, H., & Bamji, C. (2004, June). A time-of-flight depth sensor-system description, issues and solutions. In Computer Vision and Pattern Recognition Workshop, 2004. CVPRW'04. Conference on (pp. 35-35). IEEE.

Hornung, A., Wurm, K., Bennewitz, M., Stachniss, C., and Burgard, W. (2013). OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octree. Autonomous Robots. 34 (3), pp(189-206). DOI 10.1007/s10514-012-9321-0

Weise, T., Wismer, T., Leibe, B., and Van Gool, L. (2009). Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on. pp (1630 - 1637). IEEE. DOI: 10.1109/ICCVW.2009.5457479

Krainin, M., Henry, P., Ren, X., and Fox D. (2011). The International Journal of Robotics. 30 (11), pp (1311-1327). DOI: 10.1177/0278364911403178

Collet, A., Berenson, D., Srinivasa, S., and Ferguson, D. (2009). Object Recognition and Full Pose Registration from a Single Image for Robotic Manipulation. IEEE International Conference on Robotics and Automation (ICRA '09), May, 2009. Source: https://www.ri.cmu.edu/

Lai, K., and Fox, D. (2009). 3D Laser Scan Classification Using Web Data and Domain Adaptation. Proc. of Robotics: Science and Systems, 2009. pp (1-8)

Ebrahim, M. (2014). 3D laser scanners: History, Applications, and Future. Research Gate. DOI: 10.13140/2.1.3331.3284

GNU. Makefile manual. (n.d.). Retrieved 10-15-2015.
http://www.gnu.org/software/make/manual/html_node/index.html#SEC_Contents

jonY. (01-18-2008). MSYS. Retrieved 10-15-2015. http://www.mingw.org/wiki/msys

FSU.PLY Files: an ASCII Polygon Format. (n.d.) Retrieved 10-24-2015.
http://people.sc.fsu.edu/~jburkardt/data/ply/ply.html

Paulbourke. PLY-Polygon File Format. (n.d.) Retrieved 10-24-2015. http://paulbourke.net/dataformats/ply/

Tenney, Matthew. 2012. Point Clouds to Mesh in “MeshLab”.CAST Technical Publications Series. Number 10062. http://gmv.cast.uark.edu/scanning/point-clouds-to-mesh-in-meshlab/. [Retrieved : 10-20-2015]. [Last Updated: 29 November 2012]. Disclaimer: All logos and trademarks remain the property of their respective owners.

FSU.PLC Files. (n.d.) Retrieved 10-24-2015. http://people.sc.fsu.edu/~jburkardt/data/plc/plc.html

Zacobria Lars. UR Script Programming. (n.d.) Retrieved 11-10-2015. http://www.zacobria.com/universal-robots-zacobria-forum-hints-tips-how-to/universal-robots-ur-script/

M. Asad. (12-09-2011). Linux Like Installation of OpenCV 2.3.0 on Windows. Retrieved 11-05-2015. http://seevisionc.blogspot.co.uk/2011/09/linux-like-installation-of-opencv-230.html

opencv.org. Installation in Windows (n.d.). Retrieved 11-05-2015. http://docs.opencv.org/2.4/doc/tutorials/introduction/windows_install/windows_install.html#windows-installation

Stackoverflow. Compiling MinGW libs for OpenCV under Windows. (n.d.) Retrieved 11-05-2015. http://stackoverflow.com/questions/26397657/compiling-mingw-libs-for-opencv-under-windows

Zahid Hasan. How to install OpenCV on Windows (64bit) using MinGW (64) and Codeblocks. (n.d.) Retrieved 11-05-2015. https://zahidhasan.wordpress.com/2013/02/16/how-to-install-opencv-on-windows-7-64bit-using-mingw-64-and-codeblocks/

OpenCV. OpenCV Documentation. (n.d.) Retrieved 11-05-2015.

http://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html?highlight=gaussianblur#void medianBlur(InputArray src, OutputArray dst, int ksize)

Silver Moon. (12-26-2011). Winsock tutorial-Socket programming in C on windows. Retrieved 10-12-2015. http://www.binarytides.com/winsock-socket-programming-tutorial/

Silver Moon (08-30-2012). UDP socket programming in winsock. Retrieved 10-12-2015. http://www.binarytides.com/udp-socket-programming-in-winsock/

MSDN Microsoft. Windows Sockets Error Codes. (n.d.) Retrieved 10-13-2015. https://msdn.microsoft.com/en-us/library/windows/desktop/ms740668(v=vs.85).aspx

Walrand. Transport Protocols: UDP, TCP. (n.d.). Retrieved 10-13-2015. http://robotics.eecs.berkeley.edu/~wlr/12203/transport-slides.pdf

Chunyan Fu. TCP/UDP Basics. (n.d.) Retrieved 10-14-2015. http://users.encs.concordia.ca/~glitho/F09_TCP_UDP.pdf

Sina. (01-05-2009). Basic Kinematics of Constrained Rigid Bodies. Retrieved 11-20-2015. http://blog.sina.com.cn/s/blog_48571ec80100brx6.html

Leuze Electronic. (2014). Technical Description LPS. http://www.leuze.com

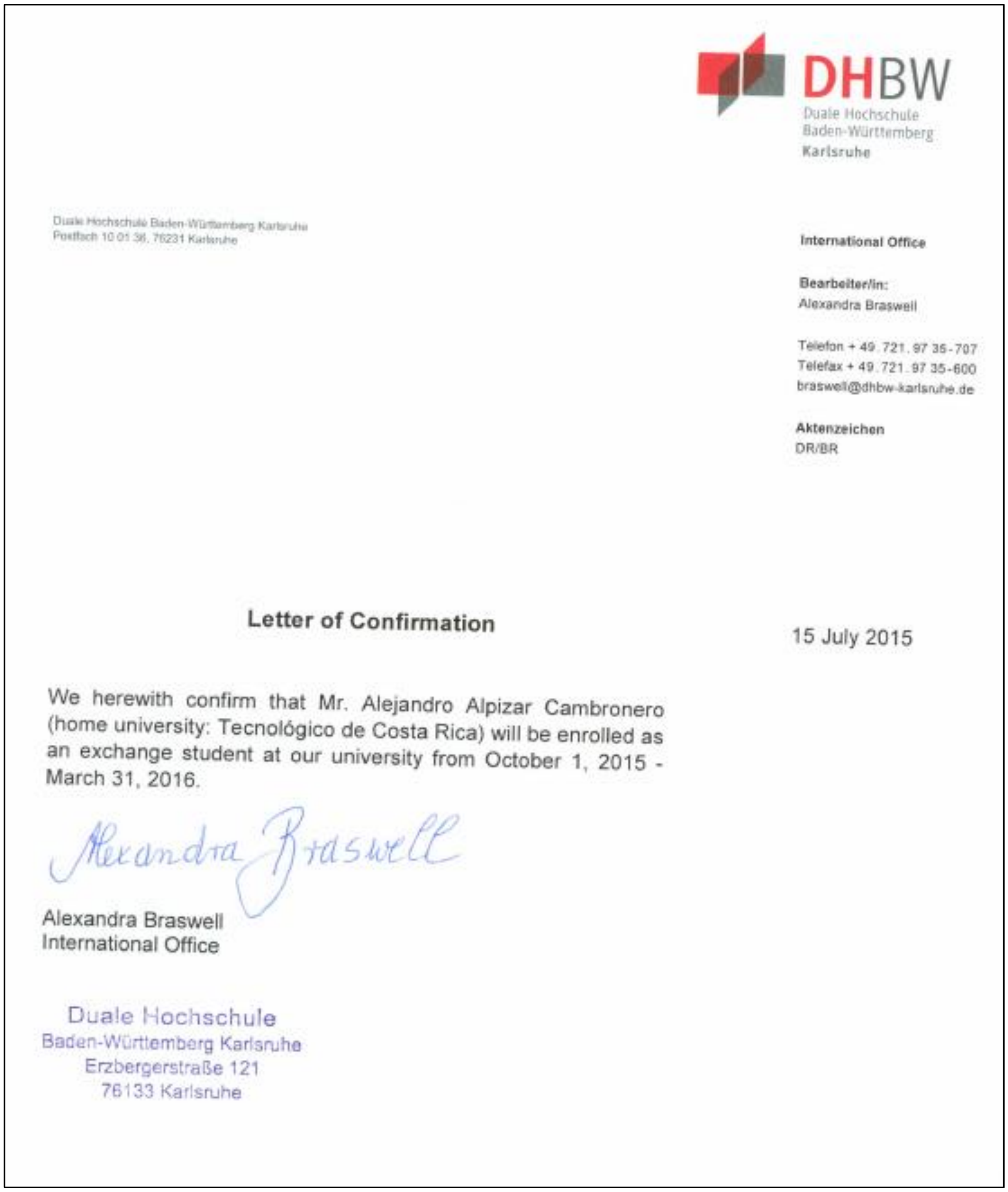
Universal Robots. (January, 2012). User manual. Version 1.5.

Universal Robots. (August, 2013). The URScript Programming language. Version 1.8.

XV. Appendix section

i. Appendix A: Acceptance letter from DHBW

Acceptance letter for developing the Graduation Project as an exchange student in the DHBW, in the period from October 2015 to March 2016.



DHBW
Duale Hochschule
Baden-Württemberg
Karlsruhe

Duale Hochschule Baden-Württemberg Karlsruhe
Postfach 10 01 36, 76231 Karlsruhe

International Office

Bearbeiter/in:
Alexandra Braswell

Telefon + 49 . 721 . 97 35-707
Telefax + 49 . 721 . 97 35-600
braswell@dhbw-karlsruhe.de

Aktenzeichen
DR/BR

Letter of Confirmation

15 July 2015

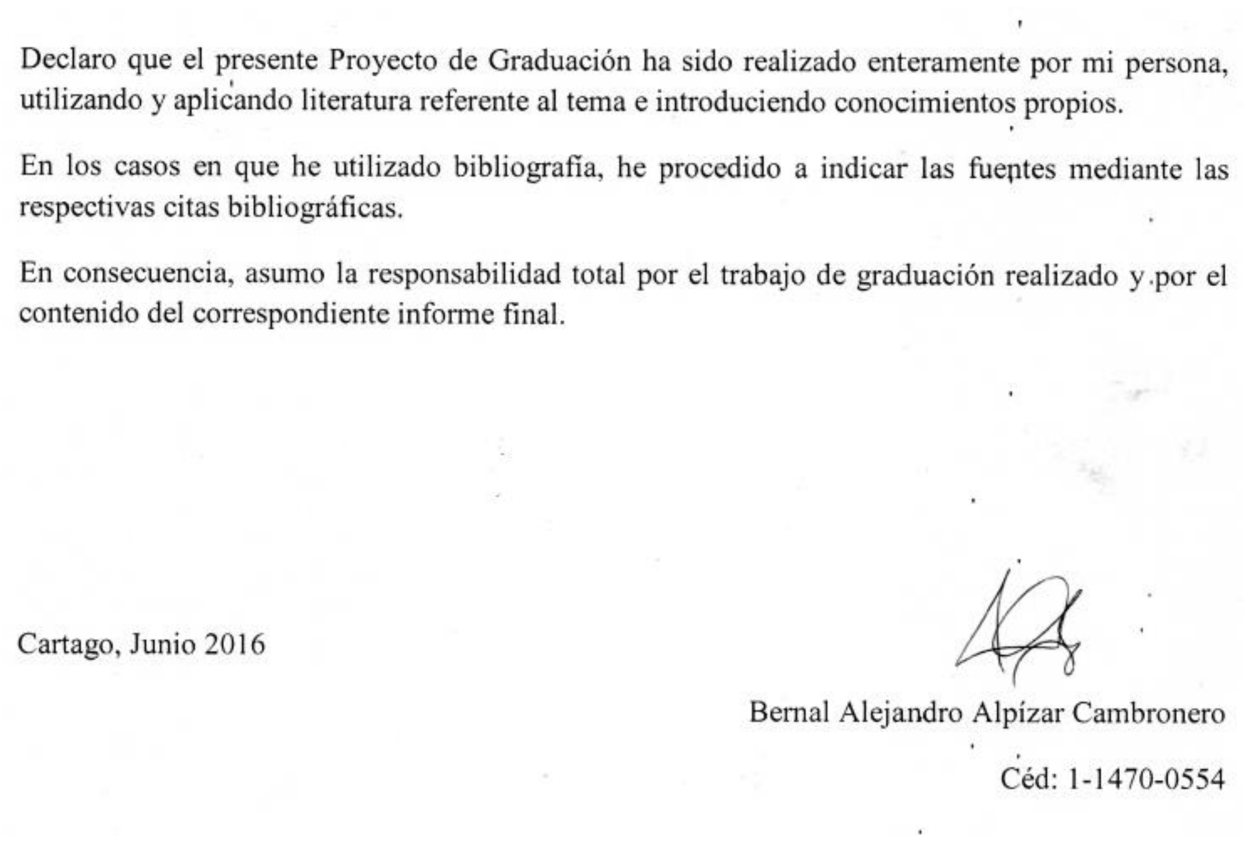
We herewith confirm that Mr. Alejandro Alpizar Cambronero (home university: Tecnológico de Costa Rica) will be enrolled as an exchange student at our university from October 1, 2015 - March 31, 2016.

Alexandra Braswell

Alexandra Braswell
International Office

Duale Hochschule
Baden-Württemberg Karlsruhe
Erzbergerstraße 121
76133 Karlsruhe

ii. Appendix B: Autenticity Declaration



Declaro que el presente Proyecto de Graduación ha sido realizado enteramente por mi persona, utilizando y aplicando literatura referente al tema e introduciendo conocimientos propios.

En los casos en que he utilizado bibliografía, he procedido a indicar las fuentes mediante las respectivas citas bibliográficas.

En consecuencia, asumo la responsabilidad total por el trabajo de graduación realizado y por el contenido del correspondiente informe final.

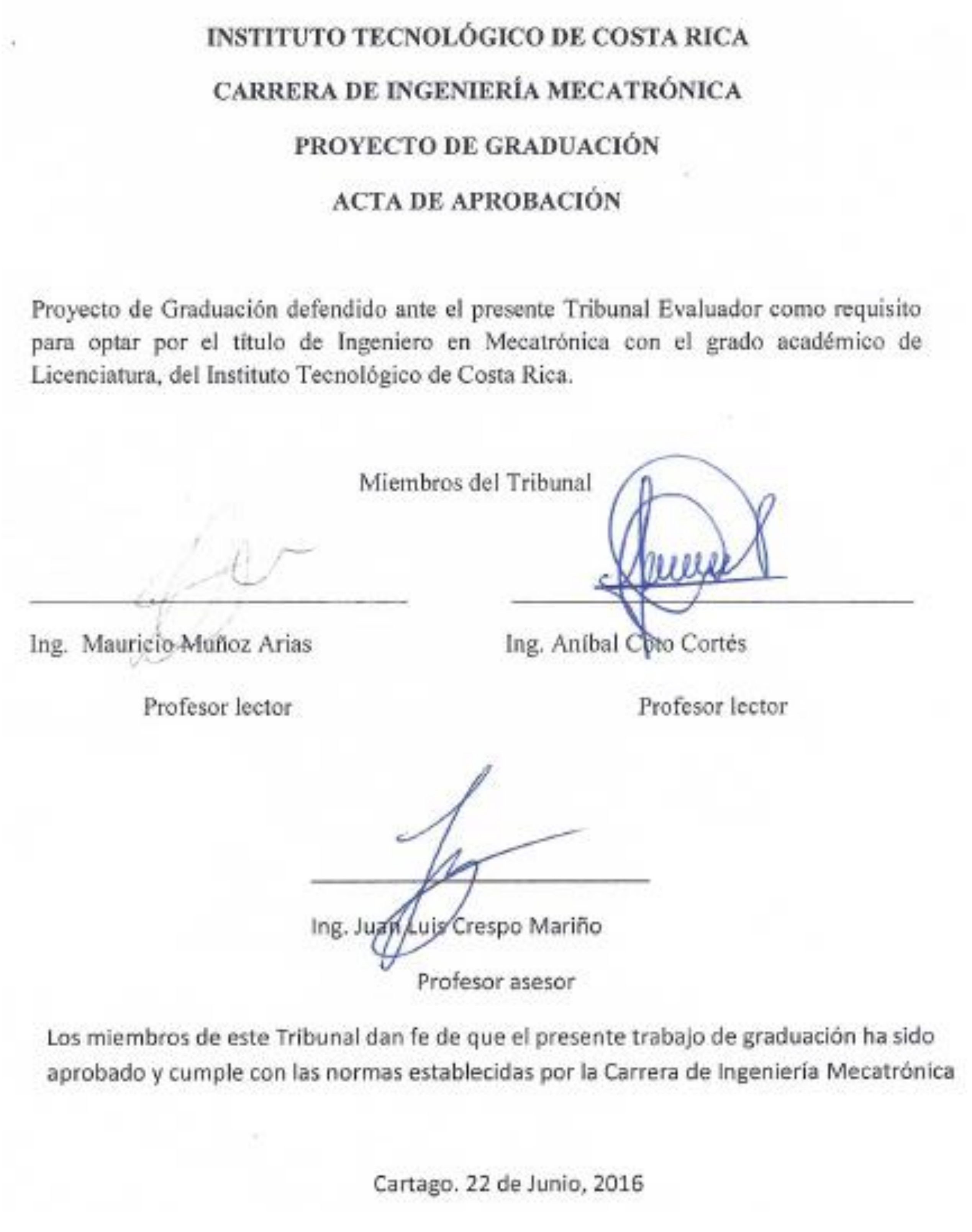
Cartago, Junio 2016

[illegible]

Bernal Alejandro Alpízar Cambronero

Céd: 1-1470-0554

iii. Appendix C: Final Evaluation



INSTITUTO TECNOLÓGICO DE COSTA RICA

CARRERA DE INGENIERÍA MECATRÓNICA

PROYECTO DE GRADUACIÓN

ACTA DE APROBACIÓN

Proyecto de Graduación defendido ante el presente Tribunal Evaluador como requisito para optar por el título de Ingeniero en Mecatrónica con el grado académico de Licenciatura, del Instituto Tecnológico de Costa Rica.

Miembros del Tribunal

[illegible]

Ing. Mauricio Muñoz Arias

Profesor lector

[illegible]

Ing. Aníbal Coto Cortés

Profesor lector

[illegible]

Ing. Juan Luis Crespo Mariño

Profesor asesor

Los miembros de este Tribunal dan fe de que el presente trabajo de graduación ha sido aprobado y cumple con las normas establecidas por la Carrera de Ingeniería Mecatrónica

Cartago. 22 de Junio, 2016

iv. Appendix D: Contact information from professor at DHBW

Contact for the developing of the project:

Prof. Dr.-Ing. Clemens Reitze

c/o Duale Hochschule Baden-Württemberg Karlsruhe
Baden-Wuerttemberg Cooperative State University
Fakultät für Technik, Maschinenbau

Erzbergerstr. 121
D-76133 Karlsruhe

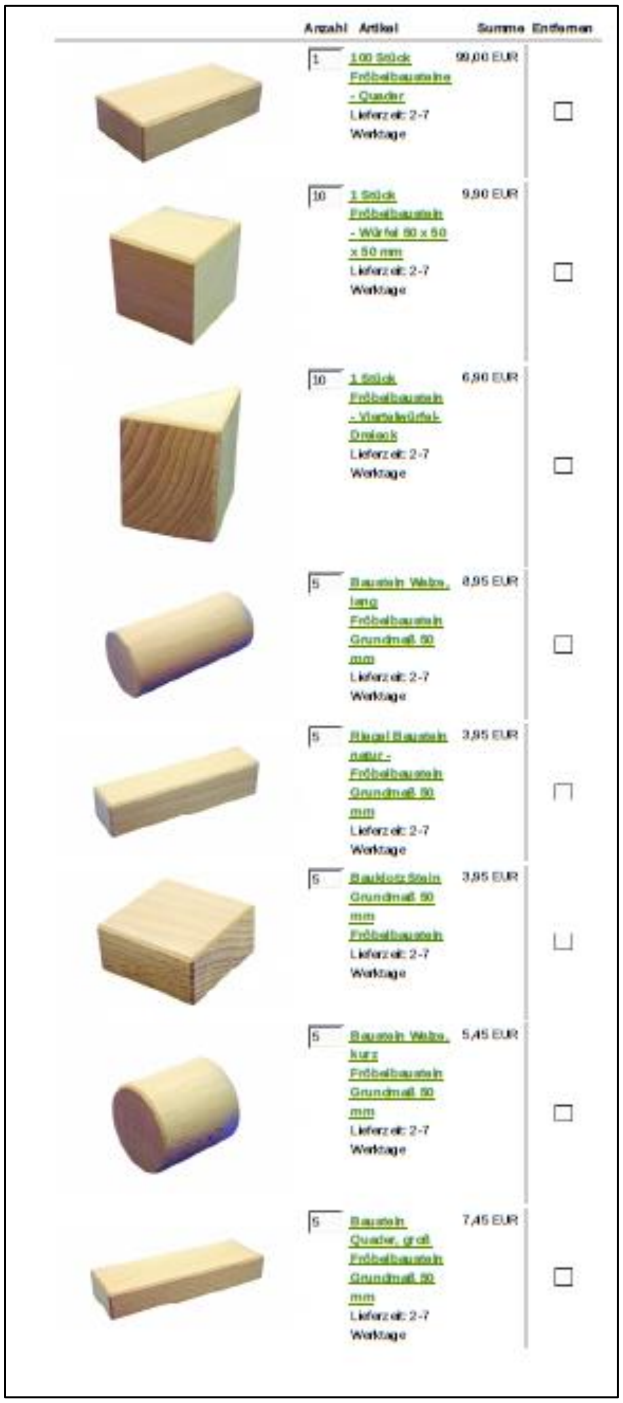
Postfach 100136
D-76231 Karlsruhe

GERMANY

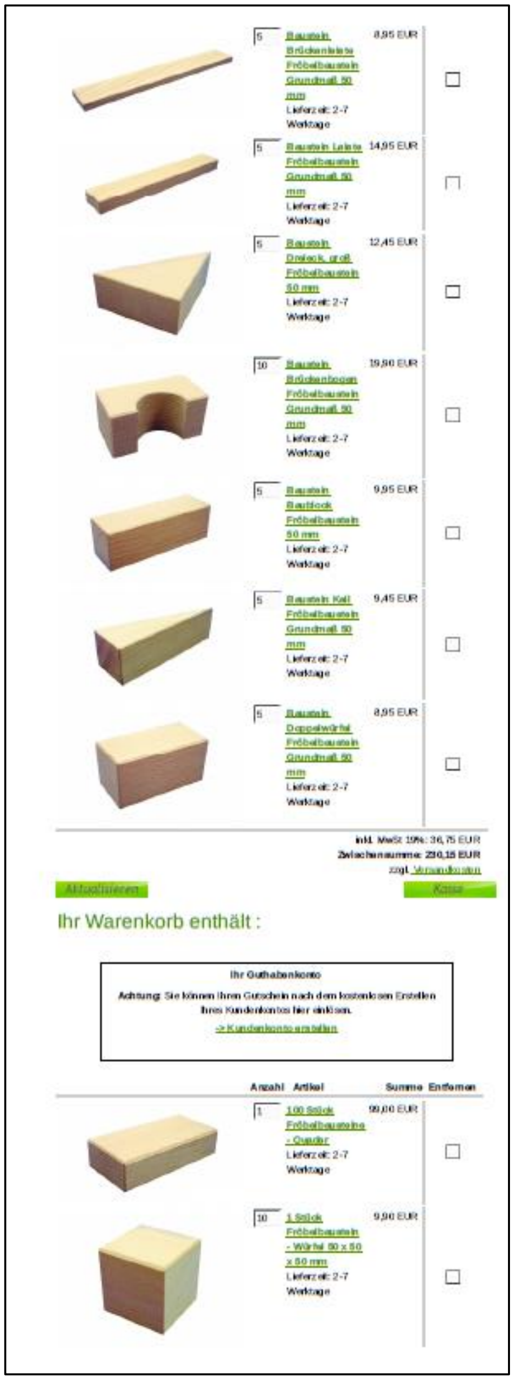
Fon +49 721 9735 829
Fax +49 721 9735 77 829
Mobile +49 176 80140133
eMail clemens.reitze@dhbw-karlsruhe.de

http://www.dhbw-karlsruhe.de

v. Appendix E: Objects available for research



Anzahl	Artikel	Summe	Entfernen
1	100 Stück Fröbelbausteine - Quader Lieferzeit: 2-7 Werktage	99,00 EUR	☐
10	1 Stück Fröbelbaustein - Würfel 50 x 50 x 50 mm Lieferzeit: 2-7 Werktage	9,90 EUR	☐
10	1 Stück Fröbelbaustein - Viertelwürfel-Dreieck Lieferzeit: 2-7 Werktage	6,90 EUR	☐
5	Baustein Walze, lang Fröbelbaustein Grundmaß 50 mm Lieferzeit: 2-7 Werktage	8,95 EUR	☐
5	Riegel Baustein natur - Fröbelbaustein Grundmaß 50 mm Lieferzeit: 2-7 Werktage	3,95 EUR	☐
5	Bauklotz Stein Grundmaß 50 mm Fröbelbaustein Lieferzeit: 2-7 Werktage	3,95 EUR	☐
5	Baustein Walze, kurz Fröbelbaustein Grundmaß 50 mm Lieferzeit: 2-7 Werktage	5,45 EUR	☐
5	Baustein Quader, groß Fröbelbaustein Grundmaß 50 mm Lieferzeit: 2-7 Werktage	7,45 EUR	☐



5	Baustein Brückenleiste Fröbelbaustein Grundmaß 50 mm Lieferzeit: 2-7 Werktage	8,95 EUR	☐
5	Baustein Leiste Fröbelbaustein Grundmaß 50 mm Lieferzeit: 2-7 Werktage	14,95 EUR	☐
5	Baustein Dreieck, groß Fröbelbaustein 50 mm Lieferzeit: 2-7 Werktage	12,45 EUR	☐
10	Baustein Brückenbogen Fröbelbaustein Grundmaß 50 mm Lieferzeit: 2-7 Werktage	19,90 EUR	☐
5	Baustein Baublock Fröbelbaustein 50 mm Lieferzeit: 2-7 Werktage	9,95 EUR	☐
5	Baustein Keil Fröbelbaustein Grundmaß 50 mm Lieferzeit: 2-7 Werktage	9,45 EUR	☐
5	Baustein Doppelwürfel Fröbelbaustein Grundmaß 50 mm Lieferzeit: 2-7 Werktage	8,95 EUR	☐

inkl. MwSt 19%: 36,75 EUR
Zwischensumme: 230,15 EUR
zzgl. Versandkosten

Ihr Warenkorb enthält :

Ihr Guthabenkonto

Achtung: Sie können Ihren Gutschein nach dem kostenlosen Erstellen Ihres Kundenkontos hier einlösen.

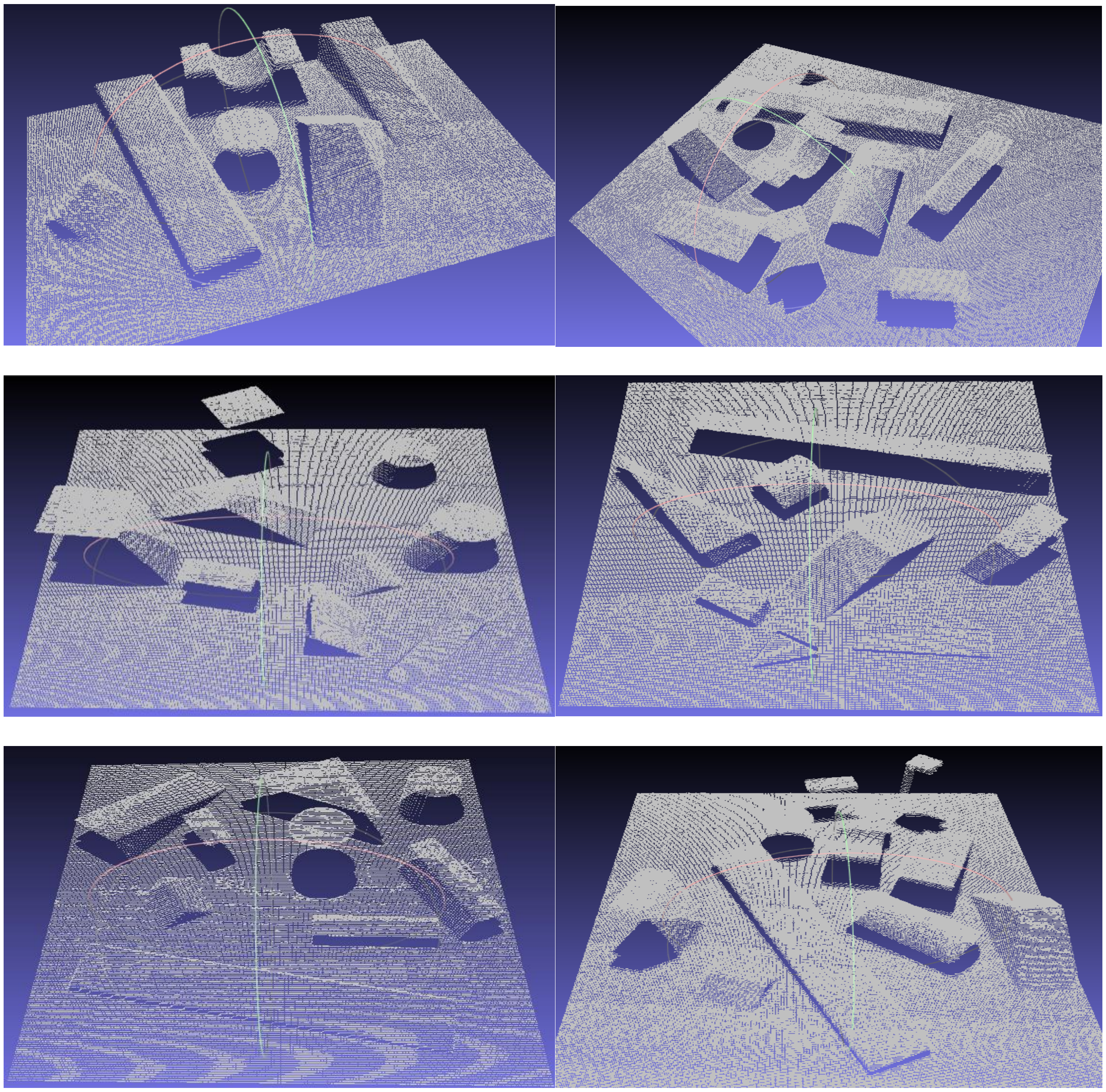
-> Kundenkonto erstellen

Anzahl	Artikel	Summe	Entfernen
1	100 Stück Fröbelbausteine - Quader Lieferzeit: 2-7 Werktage	99,00 EUR	☐
10	1 Stück Fröbelbaustein - Würfel 50 x 50 x 50 mm Lieferzeit: 2-7 Werktage	9,90 EUR	☐

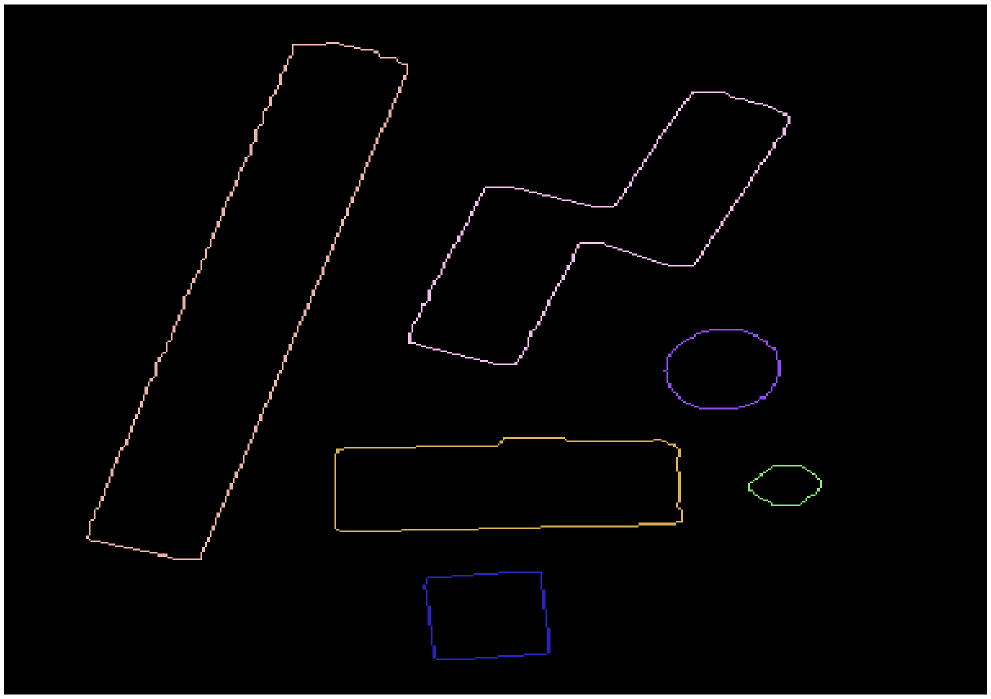
vi. Appendix F: Makefile used for the Leuze program

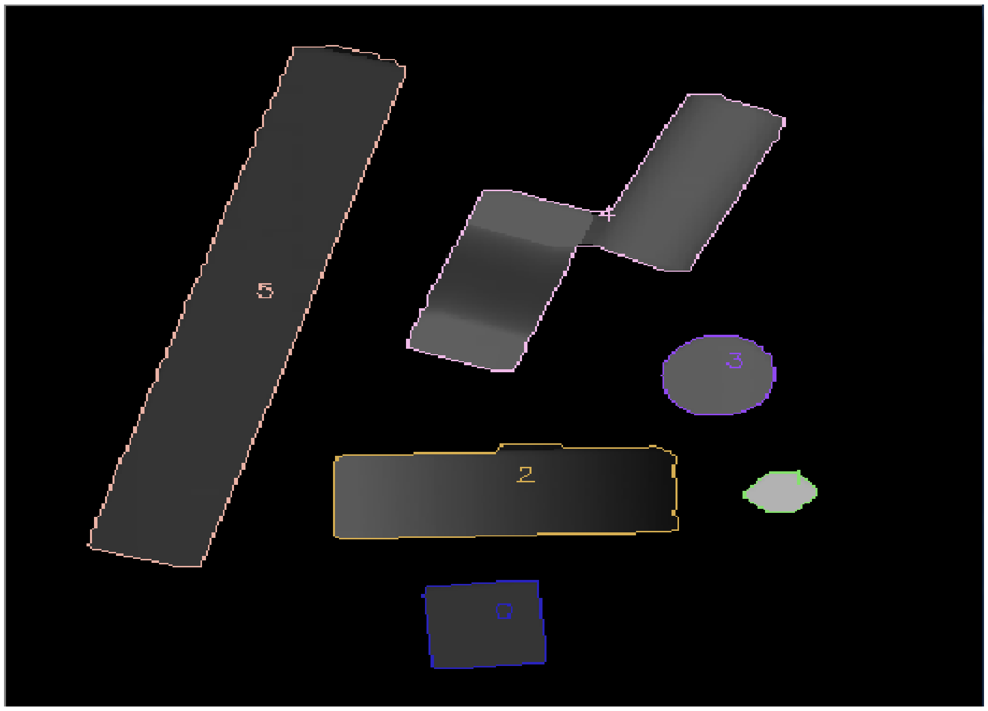
```
# file: Makefile

CC = gcc

OSTYPE := $(shell uname -s)
ifeq ($(OSTYPE), Linux)
  OSTYPE := linux
endif

CFLAGS = -g -O0 -Wall
LDFLAGS =
LD_LIBS =

ifeq ($(OSTYPE),linux)
CFLAGS += -DLINUX
else
CFLAGS += -DWIN64
LD_LIBS +=  -lWs2_32
endif

default:  sensor_client

leuze-lps.o:     leuze-lps.c leuze-lps.h Makefile
sensor_client.o: sensor_client.c leuze-lps.h  Makefile
ptg.o: ptg.c  Makefile
ur5_test1.o: ur5_test1.c  Makefile
ur5_server.o: ur5_server.c  Makefile
ur5_client.o: ur5_client.c  Makefile
client.o: client.c Makefile

sensor_client:  sensor_client.o leuze-lps.o
	$(CC) $(CFLAGS) $(LDFLAGS) -o $@ $^ $(LD_LIBS)
server:  server.o
	$(CC) $(CFLAGS) $(LDFLAGS) -o $@ $^ $(LD_LIBS)
client:  client.o
	$(CC) $(CFLAGS) $(LDFLAGS) -o $@ $^ $(LD_LIBS)
sensor_server:  sensor_server.o
	$(CC) $(CFLAGS) $(LDFLAGS) -o $@ $^ $(LD_LIBS)
ptg:  ptg.o
	$(CC) $(CFLAGS) $(LDFLAGS) -o $@ $^ $(LD_LIBS)
udpclient:  udpclient.o
	$(CC) $(CFLAGS) $(LDFLAGS) -o $@ $^ $(LD_LIBS)
ur5_test1:  ur5_test1.o
	$(CC) $(CFLAGS) $(LDFLAGS) -o $@ $^ $(LD_LIBS)
ur5_client:  ur5_client.o
	$(CC) $(CFLAGS) $(LDFLAGS) -o $@ $^ $(LD_LIBS)
ur5_server:  ur5_server.o
	$(CC) $(CFLAGS) $(LDFLAGS) -o $@ $^ $(LD_LIBS)

clean:
	-rm -f *.o *~
```

vii. Appendix G: Different images acquired with LPS



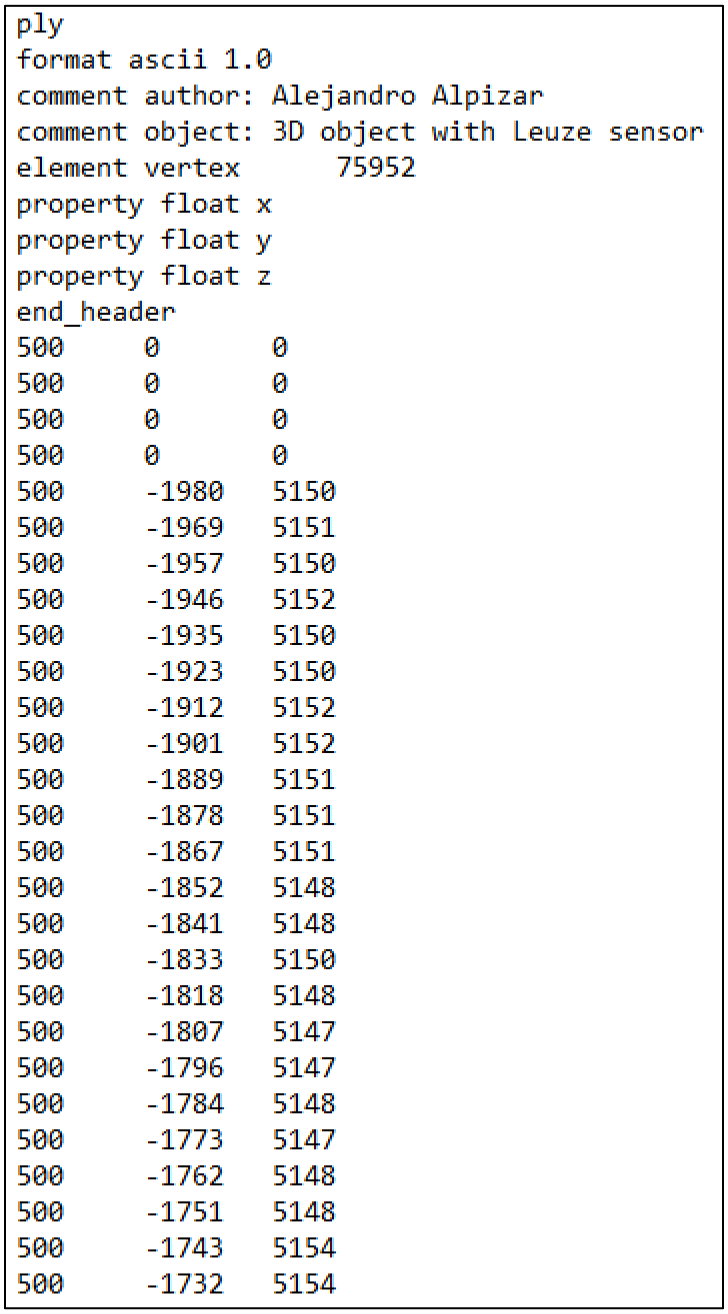
viii. Appendix H: Error detected when scanning



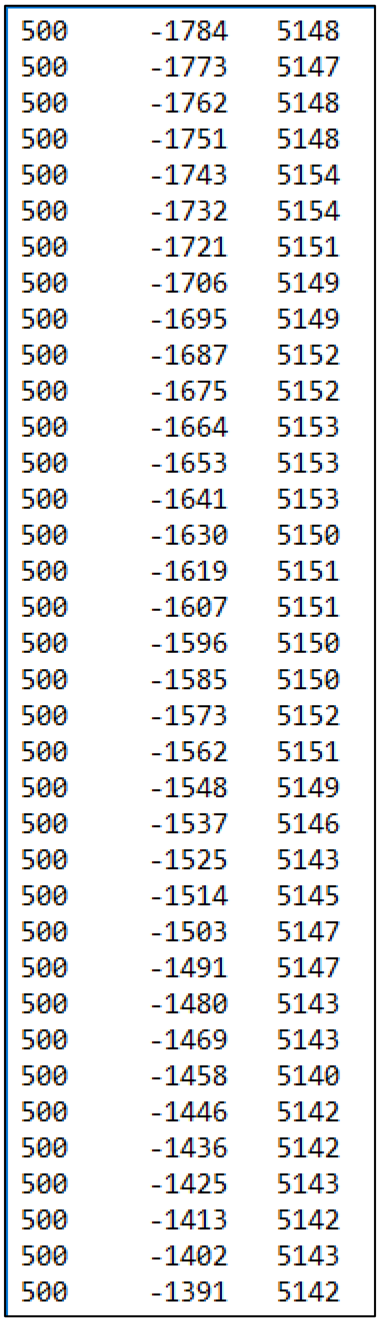


ix. Appendix I: Sample of data file with coordinates

The tabled values are Y axis, X axis and Z axis.



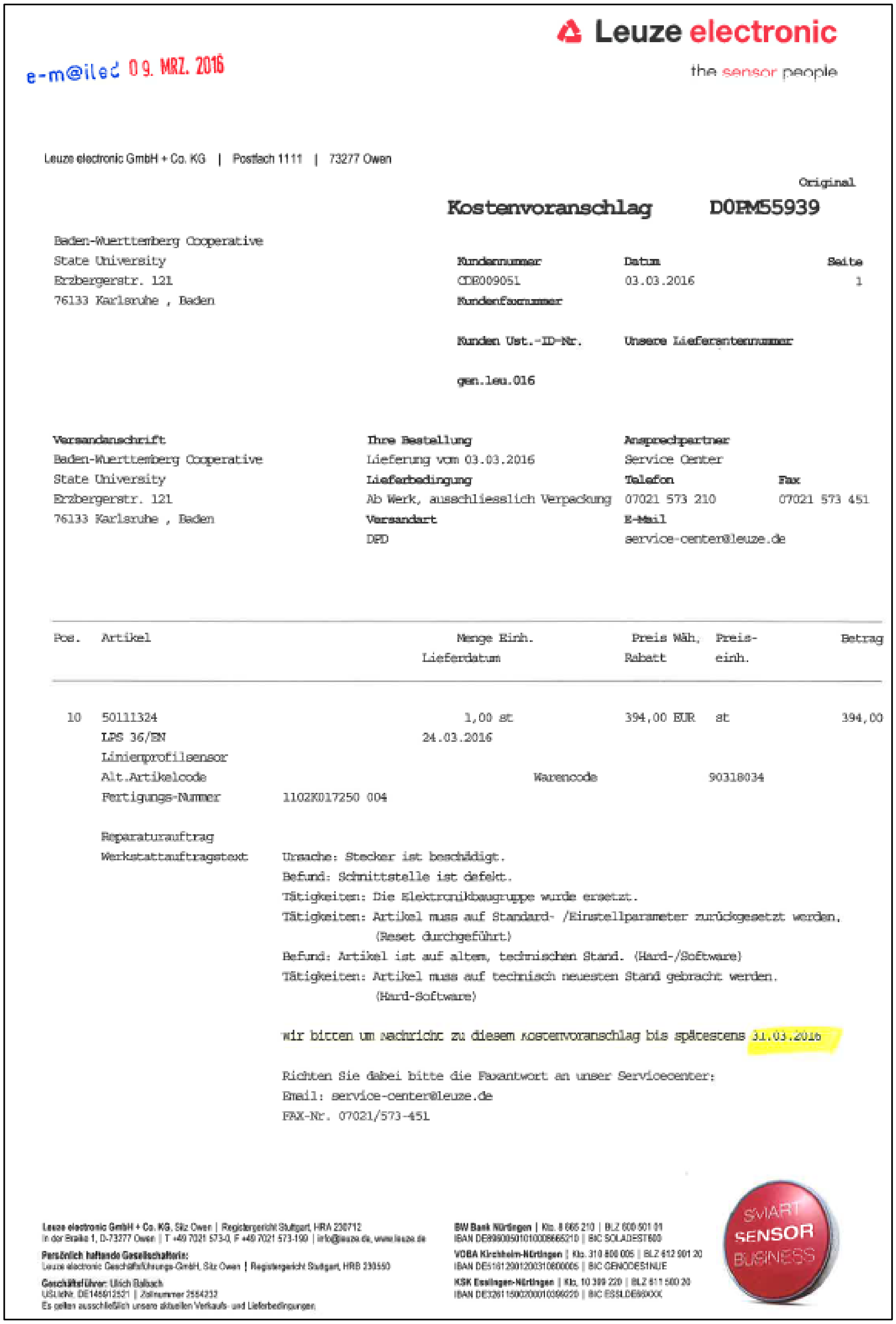
```
ply
format ascii 1.0
comment author: Alejandro Alpizar
comment object: 3D object with Leuze sensor
element vertex      75952
property float x
property float y
property float z
end_header
500     0       0
500     0       0
500     0       0
500     0       0
500     -1980   5150
500     -1969   5151
500     -1957   5150
500     -1946   5152
500     -1935   5150
500     -1923   5150
500     -1912   5152
500     -1901   5152
500     -1889   5151
500     -1878   5151
500     -1867   5151
500     -1852   5148
500     -1841   5148
500     -1833   5150
500     -1818   5148
500     -1807   5147
500     -1796   5147
500     -1784   5148
500     -1773   5147
500     -1762   5148
500     -1751   5148
500     -1743   5154
500     -1732   5154
```



```
500     -1784   5148
500     -1773   5147
500     -1762   5148
500     -1751   5148
500     -1743   5154
500     -1732   5154
500     -1721   5151
500     -1706   5149
500     -1695   5149
500     -1687   5152
500     -1675   5152
500     -1664   5153
500     -1653   5153
500     -1641   5153
500     -1630   5150
500     -1619   5151
500     -1607   5151
500     -1596   5150
500     -1585   5150
500     -1573   5152
500     -1562   5151
500     -1548   5149
500     -1537   5146
500     -1525   5143
500     -1514   5145
500     -1503   5147
500     -1491   5147
500     -1480   5143
500     -1469   5143
500     -1458   5140
500     -1446   5142
500     -1436   5142
500     -1425   5143
500     -1413   5142
500     -1402   5143
500     -1391   5142
```

…

x. Appendix J: Receipt of repair of LPS sensor



Leuze electronic

the sensor people

e-m@iled 09. MRZ. 2016

Leuze electronic GmbH + Co. KG | Postfach 1111 | 73277 Owen

Original

Kostenvoranschlag D0PM55939

Baden-Wuerttemberg Cooperative
State University
Erzbergerstr. 121
76133 Karlsruhe , Baden

Kundennummer	Datum	Seite
CDE009051	03.03.2016	1

Kundenfaxnummer

Kunden Ust.-ID-Nr.	Unsere Lieferantennummer

gen.leu.016

Versandanschrift	Ihre Bestellung	Ansprechpartner	
Baden-Wuerttemberg Cooperative	Lieferung vom 03.03.2016	Service Center	
State University	**Lieferbedingung**	**Telefon**	**Fax**
Erzbergerstr. 121	Ab Werk, ausschliesslich Verpackung	07021 573 210	07021 573 451
76133 Karlsruhe , Baden	**Versandart**	**E-Mail**	
	DPD	service-center@leuze.de	

Pos.	Artikel	Menge Einh. Lieferdatum	Preis Wäh. Rabatt	Preis-einh.	Betrag
10	50111324	1,00 st	394,00 EUR	st	394,00
	LPS 36/EN	24.03.2016			
	Linienprofilsensor				
	Alt.Artikelcode		Warencode	90318034	
	Fertigungs-Nummer 1102K017250 004				

Reparaturauftrag

Werkstattauftragstext

Ursache: Stecker ist beschädigt.
Befund: Schnittstelle ist defekt.
Tätigkeiten: Die Elektronikbaugruppe wurde ersetzt.
Tätigkeiten: Artikel muss auf Standard- /Einstellparameter zurückgesetzt werden.
(Reset durchgeführt)
Befund: Artikel ist auf altem, technischen Stand. (Hard-/Software)
Tätigkeiten: Artikel muss auf technisch neuesten Stand gebracht werden.
(Hard-Software)

Wir bitten um Nachricht zu diesem Kostenvoranschlag bis spätestens 31.03.2016

Richten Sie dabei bitte die Faxantwort an unser Servicecenter:
Email: service-center@leuze.de
FAX-Nr. 07021/573-451

Leuze electronic GmbH + Co. KG, Sitz Owen | Registergericht Stuttgart, HRA 230712
In der Braike 1, D-73277 Owen | T +49 7021 573-0, F +49 7021 573-199 | info@leuze.de, www.leuze.de
Persönlich haftende Gesellschafterin:
Leuze electronic Geschäftsführungs-GmbH, Sitz Owen | Registergericht Stuttgart, HRB 230550
Geschäftsführer: Ulrich Balbach
USt.IdNr. DE145912521 | Zollnummer 2554232
Es gelten ausschließlich unsere aktuellen Verkaufs- und Lieferbedingungen.

BW Bank Nürtingen | Kto. 8 665 210 | BLZ 600 501 01
IBAN DE89600501010008665210 | BIC SOLADEST600
VOBA Kirchheim-Nürtingen | Kto. 310 800 005 | BLZ 612 901 20
IBAN DE51612901200310800005 | BIC GENODES1NUE
KSK Esslingen-Nürtingen | Kto. 10 399 220 | BLZ 611 500 20
IBAN DE32611500200010399220 | BIC ESSLDE66XXX

SMART SENSOR BUSINESS

xi. Appendix K: Codes developed for final solution

Header file for LPS

```
/*
******************************************************************************
**  Leuze LPS Interface
**
**
**
** - UDP/IP communication
**   - port 9008
**   - port 5634
**   - ICMP echo messages
** -  byte order: BIG ENDIAN, high byte first, followed by low byte
**    (Intel x86 CPUs, Siemens S7, ... uses little endian format,
**    low byte first, followed by high byte)
**
** - measure mode:
**   - free running measurement operating mode:
**     - continuously, f_max = 100Hz
**     - no measurement request needed
**   - trigger mode:
**     - after a trigger signal
**     - after ethernet trigger command
** - command mode:
**   - reaction to triggers
**
**
** ----------------------------------------------------------------------
**
**
**
******************************************************************************
*/


#ifndef __LEUZE_LPS_H__
#define __LEUZE_LPS_H__

#ifdef __cplusplus
extern "C" {
#endif



   enum {
         LPS_Status_MeasureMode,
         LPS_Status_MenuMode,
         LPS_Status_CommandMode,
    LPS_Status_ErrorMode,

    LPS_Status_Activated,         // 0|1 by activation function
    LPS_Status_Warning,
    LPS_Status_TriggeredMeasureMode,  // 0: free running, 1: triggered
    LPS_Status_ErrorDetected
};

enum {
    LPS_Cmd_Ok =                  0x4141, // Acknowledge, confirmed
    LPS_Cmd_Error =      0x414E, // not Acknowledge, Error, not executed

    LPS_Cmd_ConnectToSensor =                         0x434E,

    LPS_Cmd_DisconnectFromSensor =                            0x4443,

    // 0: Standard-Connect, 2 separate data packets for Z and X, 782 bytes
    // 1: Hi-Connect, only LPS36HI/EN, 1 common data packet for Z and X, 990 bytes

    LPS_Cmd_CmdModeEnter =                            0x3132,
    LPS_Cmd_CmdModeExit =                  0x3133,

    // in command mode:
    LPS_Cmd_SetLaserGate =                  0x0001,
    LPS_Cmd_TriggerSingleMeasurement =      0x0003,

    LPS_Cmd_GetXCoordinates =               0x0011,
    LPS_Cmd_GetXCoordinates_Ans =                     0x0012,
    // - signed values, unit: 1/100mm, 376 values

    LPS_Cmd_GetZCoordinates =               0x0013,
    LPS_Cmd_GetZCoordinates_Ans =                     0x0014,
    // - signed values, unit: 1/100mm, 376 values

    LPS_Cmd_GetZXCoordinats =               0x005F,
    LPS_Cmd_GetZXCoordinats_Ans =                     0x0060,
    // - signed values, unit: 1/100mm, 480 values

    LPS_Cmd_SetEncoderValues =              0x0029,

    LPS_Cmd_GetActualInspectionTask =       0x0049,
    LPS_Cmd_GetActualInspectionTask_Ans =             0x004A,
    // user data: task number 0=Task0 ... 15=Task15


    /*** Single Inspection Task Parameters */

    LPS_Cmd_SetActualInspectionTask =       0x004B,
    LPS_Cmd_SetScanNumber =                 0x0053,
    LPS_Cmd_SetSingleUserParam =            0x0059,
	LPS_Cmd_GetSingleUserParam =		0x005B,
	LPS_Cmd_GetSingleUserParam_Ans =	0x005C, // Parameter is output

	// Parameters:
	LPS_ParamID_NumberOfInspectTask =			0x0BB8,
	LPS_ParamID_NameOfInspectTask =			0x0BB9,
	LPS_ParamID_OperatingMode =				0x0BBA,
	LPS_ParamID_EnablingActivation =			0x0BBB,
	LPS_ParamID_EnablingCascadingOutput =			0x0BBC,
	LPS_ParamID_LaserExposureDuration =			0x0BBD,
	LPS_ParamID_LaserExposureDurationManualAdjust =	0x0BBE,

	LPS_ParamID_DetectRangeX =		0x0BBF,
	// 2 values: min, max  (-3000..+3000 LPS36, -700..+700 LPS36HI/EN)
	// unit: 1/10mm

	LPS_ParamID_DetectRangeZ =		0x0BC0,
	// 2 values: min, max  (+1900..+8100 LPS36, +1959..+6100 LPS36HI/EN)
	// unit: 1/10mm


	/*** Single User Parameters */


	/*** misc */

	LPS_Cmd_EthernetTrigger =				0x4554,

	LPS_Cmd_EthernetTrigger_Ans1 =			0x5A5A,
	// with activated output of X coordinates:
	//   - 1st of 2 packtes
	//   - Z coordinates, 376 values
	// with deactivated output of X coordinates:
	//   - X coordinates, 376 values

	LPS_Cmd_EthernetTrigger_Ans2 =			0x5858, // X
	// with activated output of X coordinates:
	//   - 2nd of 2 packtes
	//   - X coordinates, 376 values
	// with HI-Connect activated (LPS36HI):
	//   - Z and X coordinates


	LPS_Cmd_EthernetActivation =			0x4541,
	// - connection to the sensor must exist before!

};



struct tLeuzeLPS_Header {
        unsigned short   Startseq0;
        unsigned short   Startseq1;
        unsigned char    FillChar0[2];
        unsigned short  CmdNo;
        unsigned char    FillChar1[2];
        unsigned short  PacketNo;
        unsigned char    FillChar2[2];

        unsigned short  TransactionNo;       /* measure mode:
                                           *   always 0x0000
                                           * command mode:
                                           *   cmd number of the command
                                           *   that is answered
                                           */
        unsigned short  Status;
        unsigned short  EncoderH;    // [0..1] H, [2..3] L,
        unsigned short  EncoderL;    // [0..1] H, [2..3] L,
        unsigned char    FillChar3[2];
        unsigned short  ScanNo;              /* 0x0000 .. 0xFFFF, incremented by 1
                                           */
        unsigned short  UserDataType;        /* 0x0010 := 16 bit data
                                           *
                                           */
        unsigned short  UserDataN;  /* number of user data elements
                                           * of UserDataType
                                           * possible values:
                                           *   0x0000,
                                           *   0x0001, 0x0002, 0x0003,
                                           *   0x0178
                                           *   0, 1, 2, 3, 376 or 480 data words
                                           *   0, 2, 4, 6, 752 or 960 bytes
                                           */
        //unsigned char          UserData[2];
};

struct tLeuzeLPS_Msg {
        struct tLeuzeLPS_Header Header;
        unsigned char          UserData[2];
};


typedef struct tLeuzeLPS_Header  tLeuzeLPS_Header;
typedef struct tLeuzeLPS_Msg     tLeuzeLPS_Msg;


int LeuzeLPS_InitHeader  (struct tLeuzeLPS_Header *h);
int LeuzeLPS_PrintMsg (struct tLeuzeLPS_Msg *h);
int LeuzeLPS_MsgToN   (struct tLeuzeLPS_Msg *h);
int LeuzeLPS_TwoComp   (struct tLeuzeLPS_Msg *h);
//int LeuzeLPS_PrintXZ(struct tLeuzeLPS_Msg h,struct tLeuzeLPS_Msg j);
```

```
#ifdef __cplusplus
}
#endif

#endif // of #ifndef __LEUZE_LPS_H__
```

Code for LPS Scanning with UR

```
/*
****************************************************************************
** Program for scanning with LPS sensor and saving data in desired file
**
** Author: Alejandro Alpizar
**
** ----------------------------------------------------------------------
**
**
**
****************************************************************************
*/

#if defined(LINUX)
# include <sys/types.h>
# include <sys/socket.h>
# include <netinet/in.h>
# include <netdb.h>
# include <arpa/inet.h>

#elif defined(WIN32)
# include<winsock2.h>
# include<windows.h>
# include<fcntl.h>
# include<sys/stat.h>

#endif

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <errno.h>
#include "leuze-lps.h"



/*** Byte Order stuff ********************************************************/

#if 0
```

```
static const int bsti = 1;  // Byte swap test integer
#define is_bigendian() ( (*(char*)&bsti) == 0 )


// swap to network byte order (big endian)

static void *
SwapBytes (int n, void *p)
{
   int i;
   static unsigned char buf[16];        // n < 16 !!
   unsigned char *t = (unsigned char *)&buf[0];
   unsigned char *s = (unsigned char *)p;

   for (i=0; i < n; i++) {
         t[i] = s[n-i-1];
   }
   return (void*)t;
}


#if defined(LINUX) || defined(WIN32)

# define S2N_USHORT(v)        (unsigned short*) SwapBytes(2, v)
# define S2N_SHORT(v)                  (short*) SwapBytes(2, v)
# define AS2N_USHORT(v) (v) = (unsigned short*) SwapBytes(2, v)
# define AS2N_SHORT(v)  (v) =          (short*) SwapBytes(2, v)

# define S2N_UINT(v)          (unsigned  int *) SwapBytes(4, v)
# define S2N_INT(v)                     (int *) SwapBytes(4, v)
# define AS2N_UINT(v)   (v) = (unsigned  int *) SwapBytes(4, v)
# define AS2N_INT(v)    (v) =           (int *) SwapBytes(4, v)


# define S2N_FLOAT(v)         (float*)  SwapBytes(4, v)
# define AS2N_FLOAT(v)  (v) = (float*)  SwapBytes(4, v)

# define S2N_DOUBLE(v)        (double*) SwapBytes(8, v)
# define AS2N_DOUBLE(v) (v) = (double*) SwapBytes(8, v)


#elif defined(PPC)

# define S2N_USHORT(v)
# define S2N_SHORT(v)
# define AS2N_USHORT(v) (v)
# define AS2N_SHORT(v)  (v)

# define S2N_UINT(v)
# define S2N_INT(v)
# define AS2N_UINT(v)   (v)
# define AS2N_INT(v)    (v)


# define S2N_FLOAT(v)
# define AS2N_FLOAT(v)  (v)

# define S2N_DOUBLE(v)
# define AS2N_DOUBLE(v) (v)

#else
#  error ERROR Byte order not yet handled for swapping!
#endif


#endif




void
DumpMsg (const unsigned char *msg, int len)
{
    int i;
    const int n_wrap = 6;

    for (i=0; i < len; i++) {
            if (i%n_wrap == 0) {
                const char *fmt = (i>0 ? "\n%03d: " : "%03d: ");
                //printf (fmt, i);
            }
            //printf (" %02x", msg[i]);
    }
    if (i > 0)
            printf ("*******Msg dumped*******\n");
        fflush(stdout);
}



int
main(int argc, char *argv[])
{
    int rv = 0, n = 0,i = 0;

    struct sockaddr_in srv_tx;
    struct sockaddr_in srv_rx;
    struct sockaddr_in srv_other;
    int srv_size = sizeof(srv_tx);
    int sock_tx = 0;
    int sock_rx = 0;
    int waiting = 0;
    int yvec=0;
    int lecture=1;
    long offset;

    SOCKET s;
    struct sockaddr_in server, si_other;
    int slen , recv_len;
    int client_conection;
    int finish=0;
    char read;
    slen = sizeof(si_other);

    #define BUFLEN  1024//512      //512
    #define PORT 30002  //The port on which to listen for incoming data
    #define MAXPENDING 5    /* Maximum outstanding connection requests */

    char *srv_ip;
    int srv_port_tx = 0;
    int srv_port_rx = 0;

#define BUFLEN  1024
    char buf[BUFLEN];
    unsigned char *msg;
    unsigned int msg_size;
    char file_name[25];
    const char *ply_header1,*ply_header2;

    short xvec[480];
    unsigned short zvec[480];
    int   nxvec, nzvec, ycoord;

    struct tLeuzeLPS_Msg lpsdata, *plpsdata;

    memset (&buf[0],  0, sizeof(buf));
    memset (&lpsdata, 0, sizeof(lpsdata));


    srv_ip =       "192.168.28.50";
    srv_port_tx =   9008;
    srv_port_rx =   5634;

#if defined (WIN32)
    WSADATA wsa;

    //Initialise winsock
    printf("\nInitialising Winsock...");
      fflush(stdout);
    if (WSAStartup(MAKEWORD(2,2),&wsa) != 0)
    {
        printf("Failed. Error Code : %d",WSAGetLastError());
        exit(EXIT_FAILURE);
    }
    printf("Initialised.\n");
    fflush(stdout);
#endif

    // Create UDP client
    sock_tx = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
    if (sock_tx < 0) {
        printf("ERROR: opening tx socket, could not create socket, %s, returns %d\n",
                   strerror(errno), sock_tx);
                   fflush(stdout);
        return 1;
    }

    memset(&srv_tx, 0, srv_size);
    srv_tx.sin_family = AF_INET;
    srv_tx.sin_port =    htons(srv_port_tx);

    // convert IPv4 and IPv6 addresses from text to binary form
#if 0
    if ((rv=inet_aton(AF_INET, srv_ip, &srv_tx.sin_addr)) <= 0) {
        printf ("ERROR: inet_pton error occured, returns %d, %s\n",
                      rv, strerror(errno));
        return 1;
    }
#else
    srv_tx.sin_addr.S_un.S_addr = inet_addr(srv_ip);
#endif


    // Create UDP server
    sock_rx = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
    if (sock_rx < 0) {
        printf("ERROR: opening rx socket, could not create socket, %s\n",
                   strerror(errno));
                   fflush(stdout);
        return 1;
    }

    memset(&srv_rx, 0, srv_size);
    srv_rx.sin_family =      AF_INET;
    srv_rx.sin_port =        htons(srv_port_rx);
    srv_rx.sin_addr.s_addr = htonl(INADDR_ANY);

    // bind socket to port
    rv = bind(sock_rx, (struct sockaddr*)&srv_rx, srv_size);
    if (rv  == -1) {
        printf ("ERROR: binding rx socket, %s\n",
                      strerror(errno));
                      fflush(stdout);
        return 1;
```

```
   }
   printf ("bind rx returns %d\n", rv);
   fflush(stdout);

//*********************************************UR socket conection
   //Create a socket for incoming connections of the UR
   if((s = socket(PF_INET /*AF_INET */, SOCK_STREAM , IPPROTO_TCP)) ==
INVALID_SOCKET)
   {
      printf("Could not create UR socket : %d" , WSAGetLastError());
      return -1;
   }
   printf("UR Socket created.\n");

   //Prepare the sockaddr_in structure
   server.sin_family = AF_INET;
   server.sin_addr.s_addr = htonl(INADDR_ANY);
   server.sin_port = htons( PORT );

   //Bind
   if( bind(s ,(struct sockaddr *)&server , sizeof(server)) == SOCKET_ERROR)
   {
      printf("Bind UR failed with error code : %d" , WSAGetLastError());
      closesocket(s);
      WSACleanup();
      exit(EXIT_FAILURE);
   }
   puts("Bind UR done");

   if( listen(s,MAXPENDING) == SOCKET_ERROR) //MAXPENDING  //1
   {
      printf("Listen UR failed with error code : %d" , WSAGetLastError());
      closesocket(s);
      WSACleanup();
      exit(EXIT_FAILURE);
   }
   puts("Listening for UR...");



   //keep listening for data


//*********************************************UR socket conection END


//***************************************************************************
   // prepare message Connect To Sensor

   LeuzeLPS_InitHeader (&lpsdata.Header);
   lpsdata.Header.UserDataType = 0x0010;//htons(0x0010);
   lpsdata.Header.CmdNo =
LPS_Cmd_ConnectToSensor;//htons(LPS_Cmd_ConnectToSensor);
   lpsdata.Header.UserDataN =         0x0000;
   lpsdata.UserData[0]=0;
   msg_size = sizeof(lpsdata.Header)+lpsdata.Header.UserDataN ;

   msg = (unsigned char*)&lpsdata;

   //LeuzeLPS_PrintMsg (&lpsdata);
   printf ("\n");

while(1){
   // send the message
         printf ("Connecting..... \n");
         fflush(stdout);
   rv = sendto(sock_tx, (const char*)msg, msg_size, 0,
                  (struct sockaddr *) &srv_tx, srv_size);
   if (rv < 0) {
         printf ("ERROR: failed to send a message, %s\n", strerror(errno));
         fflush(stdout);
         return 1;
   } else {
         printf ("sendto: %d bytes\n", rv);
         fflush(stdout);
   }


   memset (buf, 0, sizeof(buf));

   n = recvfrom(sock_rx, (char*)buf, BUFLEN, 0,
                   (struct sockaddr *) &srv_other, &srv_size);

         printf ("ran recvfrom\n");
         fflush(stdout);

   if (n <= -1) {
         printf ("ERROR: recvfrom error, %s\n", strerror(errno));
         fflush(stdout);
   } else {
         printf ("\nrecvfrom: %s:%d  %d bytes\n",
                  inet_ntoa(srv_other.sin_addr), ntohs(srv_other.sin_port), n);
                  fflush(stdout);
         DumpMsg (buf, n);
         plpsdata = (struct tLeuzeLPS_Msg*)buf;
         LeuzeLPS_MsgToN (plpsdata);

         printf ("\n");
         LeuzeLPS_PrintMsg (plpsdata);
   if (plpsdata->Header.CmdNo==LPS_Cmd_Ok){
      printf ("\nConnected. \n");
      fflush(stdout);
```

```
        break;
    } else {
        printf ("\nError: NOT Connected. \n");
        fflush(stdout);
//*********************
    while(1){
      printf("\nAttempt to reconnect? (y or n) ");
      fflush(stdout);
      scanf(" %c", &read);
      if (read == 'y' || read == 'Y'){
        printf("\nRe");
        fflush(stdout);
        break;
      }
      else if (read == 'n' || read == 'N'){
        printf("\nTerminating program...\n\n");
          //**********
          // prepare message Disconnect From Sensor

    LeuzeLPS_InitHeader (&lpsdata.Header);
    lpsdata.Header.UserDataType = 0x0010;//htons(0x0010);
    lpsdata.Header.CmdNo = 
LPS_Cmd_DisconnectFromSensor;//htons(LPS_Cmd_ConnectToSensor);
    lpsdata.Header.UserDataN =          0x0000;
    msg_size = sizeof(lpsdata.Header);

    msg = (unsigned char*)&lpsdata;

    printf ("\nDisconnecting..... \n");

    //LeuzeLPS_PrintMsg (&lpsdata);
    // send the message
    rv = sendto(sock_tx, (const char*)msg, msg_size, 0,
                    (struct sockaddr *) &srv_tx, srv_size);
    if (rv < 0) {
            printf ("ERROR: failed to send a message, %s\n", strerror(errno));
            return 1;
    } else {
            printf ("sendto: %d bytes\n", rv);
    }


    memset (buf, 0, sizeof(buf));
    n = recvfrom(sock_rx, (char*)buf, BUFLEN, 0,
                     (struct sockaddr *) &srv_other, &srv_size);
    if (n <= -1) {
            printf ("ERROR: recvfrom error, %s\n", strerror(errno));
    } else {
            printf ("\nrecvfrom: %s:%d  %d bytes\n",
                    inet_ntoa(srv_other.sin_addr), ntohs(srv_other.sin_port), n);
            DumpMsg (buf, n);
            }

         //*********


       return -1;
       break;
     }
     else {
        printf("\nPlease type a correct command\n");
        fflush(stdout);
     }
     }
//*********************
   }


         //puts (buf);
   }
   }
    printf ("\n");
//***************************************************************************
  // prepare message Enter Command Mode

   LeuzeLPS_InitHeader (&lpsdata.Header);
   lpsdata.Header.UserDataType = 0x0010;//htons(0x0010);
   lpsdata.Header.CmdNo =   LPS_Cmd_CmdModeEnter;
   lpsdata.Header.UserDataN =        0x0000;
   lpsdata.UserData[0]=0;
   msg_size = sizeof(lpsdata.Header)+lpsdata.Header.UserDataN;

   msg = (unsigned char*)&lpsdata;

   //LeuzeLPS_PrintMsg (&lpsdata);
   printf ("\n");



   // send the message
         printf ("Entering command mode..... \n");
   rv = sendto(sock_tx, (const char*)msg, msg_size, 0,
                  (struct sockaddr *) &srv_tx, srv_size);
   if (rv < 0) {
         printf ("ERROR: failed to send command, %s\n", strerror(errno));
         return 1;
   } else {
          printf ("sendto: %d bytes\n", rv);
          waiting = 1;

      //*a*a*a*a*a*a*a*a     Reading Loop
      while (waiting){
        memset (buf, 0, sizeof(buf));
        n = recvfrom(sock_rx, (char*)buf, BUFLEN, 0,
                (struct sockaddr *) &srv_other, &srv_size);
        if (n <= -1) {
          printf ("ERROR: recvfrom error, %s\n", strerror(errno));
        } else {
            printf ("\nrecvfrom: %s:%d  %d bytes\n",
          inet_ntoa(srv_other.sin_addr), ntohs(srv_other.sin_port), n);
          DumpMsg (buf, n);
            plpsdata = (struct tLeuzeLPS_Msg*)buf;

          if (plpsdata->Header.TransactionNo!=lpsdata.Header.CmdNo){
            printf (".");
          } else {
            waiting=0;
            LeuzeLPS_MsgToN (plpsdata);
              printf ("\n");
              //LeuzeLPS_PrintMsg (plpsdata);
            if ((plpsdata->Header.CmdNo==LPS_Cmd_Ok)||( ((plpsdata-
>Header.Status)&0x00F0)==0x0040)){
              printf ("\nCommand mode entered. \n");
              fflush(stdout);
            } else {
              printf ("\nCommand mode NOT entered. Status: %x\n",((plpsdata-
>Header.Status)&0x00F0));
              fflush(stdout);
            }
          }
        }
      }
  //*a*a*a*a*a*a*a*a
   }
//*****************************************************************************
  #if 0
  // prepare message Laser Toggle
  for (i=0; i < 10; i++) {

   LeuzeLPS_InitHeader (&lpsdata.Header);
   lpsdata.Header.UserDataType = 0x0010;//htons(0x0010);
   lpsdata.Header.CmdNo =   LPS_Cmd_SetLaserGate;
   lpsdata.Header.UserDataN =        0x0001;

   if (i%2 == 0) {
   lpsdata.UserData[0]=0x0000;
   }
   else{
   lpsdata.UserData[0]=0x0001;
   }
   msg_size = sizeof(lpsdata.Header)+lpsdata.Header.UserDataN;

   msg = (unsigned char*)&lpsdata;
```

```
//LeuzeLPS_PrintMsg (&lpsdata);
printf ("\n");


// send the message
        printf ("Toggling laser..... \n");
        fflush(stdout);
rv = sendto(sock_tx, (const char*)msg, msg_size, 0,
                   (struct sockaddr *) &srv_tx, srv_size);
if (rv < 0) {
        printf ("ERROR: failed to send command, %s\n", strerror(errno));
        fflush(stdout);
        return 1;
} else {
         printf ("sendto: %d bytes\n", rv);
         fflush(stdout);
         waiting = 1;

    //*a*a*a*a*a*a*a*a     Reading Loop
    while (waiting){

      memset (buf, 0, sizeof(buf));
      n = recvfrom(sock_rx, (char*)buf, BUFLEN, 0,
                    (struct sockaddr *) &srv_other, &srv_size);
      if (n <= -1) {
        printf ("ERROR: recvfrom error, %s\n", strerror(errno));
      } else {
          printf ("\nrecvfrom: %s:%d  %d bytes\n",
        inet_ntoa(srv_other.sin_addr), ntohs(srv_other.sin_port), n);
        DumpMsg (buf, n);
           plpsdata = (struct tLeuzeLPS_Msg*)buf;

        if (plpsdata->Header.TransactionNo!=lpsdata.Header.CmdNo){
          printf (".");
        } else {
          waiting=0;
          LeuzeLPS_MsgToN (plpsdata);
              printf ("\n");
              //LeuzeLPS_PrintMsg (plpsdata);
          if (plpsdata->Header.CmdNo==LPS_Cmd_Ok){
            printf ("\nLaser Toggled. \n");
          } else {
            printf ("\nLaser NOT Toggled. \n");
          }
        }
      }
    }
//*a*a*a*a*a*a*a*a
}
```

```
    Sleep(1000);  // Windows Sleep for 1000 ms (1 second)
  }
  #endif

//*****************************************Create file
  printf("Enter the name of file to save the data:\n");
  fflush(stdout);
  memset (file_name,0, sizeof(file_name));
  gets(file_name);

  FILE *fw = fopen(file_name, "w");   //open in write mode

  if (fw == NULL)
  {
     printf("Error creating file!\n");
     exit(1);
  }
  else{
    ply_header1=
    "ply\n"
    "format ascii 1.0\n"
    "comment author: Alejandro Alpizar\n"
    "comment object: 3D object with Leuze sensor\n"
    "element vertex ";
    ply_header2=
    "\nproperty float x\n"
    "property float y\n"
    "property float z\n"
    "end_header\n";

       fprintf (fw,"%s",ply_header1);
       offset=ftell (fw);
       fprintf (fw,"%10i%s",81*376,ply_header2);
  }
  fclose(fw);

FILE *f = fopen(file_name, "a");
    if (f == NULL)
    {
     printf("Error opening file!\n");
     exit(1);
    }



//*****************************************Create file END



//************************************************************************
//**************Reading loop for sensor and UR
```

```
   //keep listening for data
   while(1)
   {
      printf("Waiting for data...");
      fflush(stdout);

      //clear the buffer by filling null, it might have previously received data
      memset(buf,'\0', BUFLEN);

      //try to receive some data, this is a blocking call


      client_conection= accept (s, (struct sockaddr *) &si_other , &slen);
      if (client_conection==INVALID_SOCKET){
        wprintf (L"accept() error with error:
%d\n",WSAGetLastError());//WSAGetLastError()//errno
        closesocket(s);
        WSACleanup();
        return -1;
      }
      else
      printf("Client connected\n");

      printf("Handling client %s:%d\n", inet_ntoa(si_other.sin_addr), ntohs(si_other.sin_port));
         fflush(stdout);

      recv_len=recv(client_conection,buf, BUFLEN, 0);

         printf ("ran recv:%i\n",recv_len);
         fflush(stdout);
      if ((recv_len) == SOCKET_ERROR)
      {
         printf("recvfrom() failed with error code : %d" , WSAGetLastError());
         exit(EXIT_FAILURE);
      }
      while(recv_len > 0)
      {
        //print details of the client/peer and the data received
        printf("Received packet from %s:%d\n", inet_ntoa(si_other.sin_addr),
ntohs(si_other.sin_port));
        printf("Data: %s\n" , buf);
        fflush(stdout);

        //*******************
        //clear the buffer by filling null, it might have previously received data
        memset(buf,'\0', BUFLEN);
        while (lecture){
        //User interaction
        while (1)
        {
```

```
      printf("\nMake new scan? (y or n) ");
      fflush(stdout);
      scanf(" %c", &read);
      if (read == 'y' || read == 'Y')
      {
        printf("\nYes selected\n");
        strcpy(buf, "Yes");
        recv_len=strlen(buf);
        finish=0;
        break;
      }
      else if (read == 'n' || read == 'N')
      {
        lecture=0;
        //recv_len=0;
        printf("\nNo selected\n");
        strcpy(buf, "No");
        recv_len=strlen(buf);
        break;
      }
      else
      printf("\nPlease type a correct command\n");
}

//now reply the client with the response
if (send(client_conection, buf, recv_len, 0) != recv_len)
{
    printf("sendto() failed with error code : %d" , WSAGetLastError());
    closesocket(s);
    WSACleanup();
    exit(EXIT_FAILURE);
}
if(strcmp(buf,"No")==0)
{
recv_len=0;
finish=1;
}

while(finish==0)
{
   //clear the buffer by filling null, it might have previously received data
   memset(buf,'\0', BUFLEN);
   printf ("receiving final state\n");
   fflush(stdout);
   recv_len=recv(client_conection,buf, BUFLEN, 0);
   //sleep(0.030);
   printf ("State: *%s*\n",buf);
   fflush(stdout);
   if (strcmp(buf,"1")==0)
      finish=1;
```

```
        if ((recv_len) == SOCKET_ERROR)
        {
          printf("Recv final state failed with error code : %d\n" , WSAGetLastError());
          closesocket(s);
          WSACleanup();
          exit(EXIT_FAILURE);
        }



        if(finish==1)
        break;

        //clear the buffer by filling null, it might have previously received data
        memset(buf,'\0', BUFLEN);
        printf ("receiving coordinates\n");
        fflush(stdout);
        recv_len=recv(client_conection,buf, BUFLEN, 0);
        //sleep(0.030);
        yvec=yvec+1;
        ycoord= (int) (atof(buf)*1000+0.5)*10;
        printf("Y Coord: %i\n\n" , ycoord);
           fflush(stdout);
        if ((recv_len) == SOCKET_ERROR)
        {
          printf("Recv coordinates failed with error code : %d" , WSAGetLastError());
          closesocket(s);
          WSACleanup();
          exit(EXIT_FAILURE);
        }
        //*****************************************************Leutze Scan
  #if 1
// prepare message Trigger Single Measurement
//for (i=0; i < 10; i++) {

 LeuzeLPS_InitHeader (&lpsdata.Header);
 lpsdata.Header.UserDataType = 0x0010;//htons(0x0010);
 lpsdata.Header.CmdNo =   LPS_Cmd_TriggerSingleMeasurement;
 lpsdata.Header.UserDataN =         0x0000;

 lpsdata.UserData[0]=0x0000;

 msg_size = sizeof(lpsdata.Header)+lpsdata.Header.UserDataN;

 msg = (unsigned char*)&lpsdata;

 //LeuzeLPS_PrintMsg (&lpsdata);
 printf ("\n");


 // send the message
```

```
        printf ("Triggering Single Measurement..... \n");
  rv = sendto(sock_tx, (const char*)msg, msg_size, 0,
                (struct sockaddr *) &srv_tx, srv_size);
  if (rv < 0) {
        printf ("ERROR: failed to send command, %s\n", strerror(errno));
        return 1;
  } else {
         printf ("sendto: %d bytes\n", rv);
         waiting = 1;

      //*a*a*a*a*a*a*a*a     Reading Loop
      while (waiting){

        memset (buf, 0, sizeof(buf));
        n = recvfrom(sock_rx, (char*)buf, BUFLEN, 0,
                     (struct sockaddr *) &srv_other, &srv_size);
        if (n <= -1) {
          printf ("ERROR: recvfrom error, %s\n", strerror(errno));
        } else {
            printf ("\nrecvfrom: %s:%d  %d bytes\n",
          inet_ntoa(srv_other.sin_addr), ntohs(srv_other.sin_port), n);
          DumpMsg (buf, n);
            plpsdata = (struct tLeuzeLPS_Msg*)buf;

          if (plpsdata->Header.TransactionNo!=lpsdata.Header.CmdNo){
            printf (".");
          } else {
            waiting=0;
            LeuzeLPS_MsgToN (plpsdata);
                printf ("\n");
                //LeuzeLPS_PrintMsg (plpsdata);
            if (plpsdata->Header.CmdNo==LPS_Cmd_Ok){
              printf ("\nTriggered Single Measurement. \n");
            } else {
              printf ("\nNOT Triggered Single Measurement. \n");
            }
          }
        }
      }
 //*a*a*a*a*a*a*a*a
  }

  //Sleep(400);  // Windows Sleep for 1000 ms (1 second)
 //}
 #endif
//*****************************************************************************
#if 1
 // prepare message LPS_Cmd_GetXCoordinates
 //for (i=0; i < 10; i++) {

  LeuzeLPS_InitHeader (&lpsdata.Header);
```

```
lpsdata.Header.UserDataType = 0x0010;//htons(0x0010);
lpsdata.Header.CmdNo =    LPS_Cmd_GetXCoordinates;
lpsdata.Header.UserDataN =          0x0000;

lpsdata.UserData[0]=0x0000;

msg_size = sizeof(lpsdata.Header)+lpsdata.Header.UserDataN;

msg = (unsigned char*)&lpsdata;

//LeuzeLPS_PrintMsg (&lpsdata);
printf ("\n");


// send the message
       printf ("Getting X coordinates..... \n");
rv = sendto(sock_tx, (const char*)msg, msg_size, 0,
                 (struct sockaddr *) &srv_tx, srv_size);
if (rv < 0) {
       printf ("ERROR: failed to send command, %s\n", strerror(errno));
       return 1;
} else {
        printf ("sendto: %d bytes\n", rv);
        waiting = 1;

    //*a*a*a*a*a*a*a*a     Reading Loop
    while (waiting){

      memset (buf, 0, sizeof(buf));
      n = recvfrom(sock_rx, (char*)buf, BUFLEN, 0,
                  (struct sockaddr *) &srv_other, &srv_size);
      if (n <= -1) {
        printf ("ERROR: recvfrom error, %s\n", strerror(errno));
      } else {
           printf ("\nrecvfrom: %s:%d  %d bytes\n",
        inet_ntoa(srv_other.sin_addr), ntohs(srv_other.sin_port), n);
        nxvec=n-30;
        DumpMsg (buf, n);
           plpsdata = (struct tLeuzeLPS_Msg*)buf;

        if (plpsdata->Header.TransactionNo!=lpsdata.Header.CmdNo){
          printf (".");
        } else {
          waiting=0;
          LeuzeLPS_MsgToN (plpsdata);
              printf ("\n");
              //LeuzeLPS_PrintMsg (plpsdata);
          if (plpsdata->Header.CmdNo==LPS_Cmd_GetXCoordinates_Ans){
              printf ("\nX Coordinates received. \n");
              //LeuzeLPS_TwoComp (plpsdata);
              memset (xvec, 0, sizeof(xvec));
```

```
                memcpy(xvec,&plpsdata->UserData,nxvec);
//              xvec=short
                printf ("\n");
               //LeuzeLPS_PrintMsg (plpsdata);
             } else {
                printf ("\nX Coordinates NOT received. \n");
             }
           }
         }
       }
  //*a*a*a*a*a*a*a*a
   }

   //Sleep(400);  // Windows Sleep for 1000 ms (1 second)
 //}
 #endif
//****************************************************************************
#if 1
  // prepare message LPS_Cmd_GetZCoordinates
  //for (i=0; i < 10; i++) {

   LeuzeLPS_InitHeader (&lpsdata.Header);
   lpsdata.Header.UserDataType = 0x0010;//htons(0x0010);
   lpsdata.Header.CmdNo =   LPS_Cmd_GetZCoordinates;
   lpsdata.Header.UserDataN =          0x0000;

   lpsdata.UserData[0]=0x0000;

   msg_size = sizeof(lpsdata.Header)+lpsdata.Header.UserDataN;

   msg = (unsigned char*)&lpsdata;

   //LeuzeLPS_PrintMsg (&lpsdata);
   printf ("\n");


   // send the message
         printf ("Getting Z coordinates..... \n");
   rv = sendto(sock_tx, (const char*)msg, msg_size, 0,
                  (struct sockaddr *) &srv_tx, srv_size);
   if (rv < 0) {
         printf ("ERROR: failed to send command, %s\n", strerror(errno));
         return 1;
   } else {
          printf ("sendto: %d bytes\n", rv);
          waiting = 1;

       //*a*a*a*a*a*a*a*a     Reading Loop
       while (waiting){
         memset (buf, 0, sizeof(buf));
         //LeuzeLPS_PrintXZ (Xdata,Xdata);
```

```
        n = recvfrom(sock_rx, (char*)buf, BUFLEN, 0,
                    (struct sockaddr *) &srv_other, &srv_size);
        if (n <= -1) {
          printf ("ERROR: recvfrom error, %s\n", strerror(errno));
        } else {
              printf ("\nrecvfrom: %s:%d  %d bytes\n",
          inet_ntoa(srv_other.sin_addr), ntohs(srv_other.sin_port), n);
          nzvec=n-30;
          DumpMsg (buf, n);
              plpsdata = (struct tLeuzeLPS_Msg*)buf;

          if (plpsdata->Header.TransactionNo!=lpsdata.Header.CmdNo){
            printf (".");
          } else {
            waiting=0;
            LeuzeLPS_MsgToN (plpsdata);
                printf ("\n");
                //LeuzeLPS_PrintMsg (plpsdata);
            if (plpsdata->Header.CmdNo==LPS_Cmd_GetZCoordinates_Ans){
              printf ("\nZ Coordinates received. \n");
              memset (zvec, 0, sizeof(zvec));
              memcpy(zvec,&plpsdata->UserData,nzvec);
              //Zdata = (struct tLeuzeLPS_Msg*)&plpsdata;
              //LeuzeLPS_PrintXZ (Xdata,Zdata);
            } else {
              printf ("\nZ Coordinates NOT received. \n");
            }
          }
        }
      }
  //*a*a*a*a*a*a*a*a
   }

   //Sleep(400);  // Windows Sleep for 1000 ms (1 second)
  //}
  #endif
//*****************************************************************************
        //print coordinates X, Y and Z to FILE

   for (i=0; i < (nxvec/2); i++) {
       fprintf (f,"%d\t%d\t%d\n", /*yvec*10,*/ycoord, xvec[i],zvec[i]);
//       fprintf (f,"%d\t%d\t%d\n", yvec*10, xvec[i],zvec[i]);
   }


          //***************************************************Leutze Scan END

       }

      }
        //*******************
      }
     if (lecture==0)
        break;

   }

//***************Reading loop for sensor and UR END

fclose(f);

//*******************Edit header
printf("Editing header...\n");

FILE *fe = fopen(file_name, "r+");
    if (fe == NULL)
    {
     printf("Error opening file!\n");
     exit(1);
    }
    else{
     fseek ( fe, offset , SEEK_SET );
     //offset from beggining of file
     fprintf (fe,"%10i",yvec*376);
    }
fclose(fe);

//*******************Edit header END

//***************************************************************************

   // prepare message Exit Command Mode

    LeuzeLPS_InitHeader (&lpsdata.Header);
    lpsdata.Header.UserDataType = 0x0010;//htons(0x0010);
    lpsdata.Header.CmdNo =    LPS_Cmd_CmdModeExit;
    lpsdata.Header.UserDataN =          0x0000;
    msg_size = sizeof(lpsdata.Header);

    msg = (unsigned char*)&lpsdata;

    //LeuzeLPS_PrintMsg (&lpsdata);
    printf ("\n");


    // send the message
            printf ("Exiting command mode..... \n");
    rv = sendto(sock_tx, (const char*)msg, msg_size, 0,
                      (struct sockaddr *) &srv_tx, srv_size);
    if (rv < 0) {
           printf ("ERROR: failed to send command, %s\n", strerror(errno));
        return 1;
    } else {
         printf ("sendto: %d bytes\n", rv);
         waiting = 1;

      //*a*a*a*a*a*a*a*a     Reading Loop
      while (waiting){

        memset (buf, 0, sizeof(buf));
        n = recvfrom(sock_rx, (char*)buf, BUFLEN, 0,
                  (struct sockaddr *) &srv_other, &srv_size);
        if (n <= -1) {
          printf ("ERROR: recvfrom error, %s\n", strerror(errno));
        } else {
            printf ("\nrecvfrom: %s:%d  %d bytes\n",
          inet_ntoa(srv_other.sin_addr), ntohs(srv_other.sin_port), n);
          DumpMsg (buf, n);
            plpsdata = (struct tLeuzeLPS_Msg*)buf;

          if (plpsdata->Header.TransactionNo!=lpsdata.Header.CmdNo){
            printf (".");
          } else {
            waiting=0;
            LeuzeLPS_MsgToN (plpsdata);
               printf ("\n");
               //LeuzeLPS_PrintMsg (plpsdata);
            if (plpsdata->Header.CmdNo==LPS_Cmd_Ok){
              printf ("\nCommand mode exited. \n");
            } else {
              printf ("\nCommand mode NOT exited. \n");
            }
          }
        }
      }
   //*a*a*a*a*a*a*a*a
    }
//****************************************************************************

    // prepare message Disconnect From Sensor

    LeuzeLPS_InitHeader (&lpsdata.Header);
    lpsdata.Header.UserDataType = 0x0010;//htons(0x0010);
    lpsdata.Header.CmdNo =
LPS_Cmd_DisconnectFromSensor;//htons(LPS_Cmd_ConnectToSensor);
    lpsdata.Header.UserDataN =         0x0000;
    msg_size = sizeof(lpsdata.Header);

    msg = (unsigned char*)&lpsdata;

    printf ("\nDisconnecting..... \n");
    //LeuzeLPS_PrintMsg (&lpsdata);
    // send the message
    rv = sendto(sock_tx, (const char*)msg, msg_size, 0,
                    (struct sockaddr *) &srv_tx, srv_size);
    if (rv < 0) {
          printf ("ERROR: failed to send a message, %s\n", strerror(errno));
          return 1;
    } else {
          printf ("sendto: %d bytes\n", rv);
    }


    memset (buf, 0, sizeof(buf));
    n = recvfrom(sock_rx, (char*)buf, BUFLEN, 0,
                     (struct sockaddr *) &srv_other, &srv_size);
    if (n <= -1) {
          printf ("ERROR: recvfrom error, %s\n", strerror(errno));
    } else {
          printf ("\nrecvfrom: %s:%d  %d bytes\n",
                    inet_ntoa(srv_other.sin_addr), ntohs(srv_other.sin_port), n);
          DumpMsg (buf, n);

          plpsdata = (struct tLeuzeLPS_Msg*)buf;

          LeuzeLPS_MsgToN (plpsdata);

       printf (":P\n");

          printf ("\n");
          //LeuzeLPS_PrintMsg (plpsdata);
    if (plpsdata->Header.CmdNo==LPS_Cmd_Ok){
       printf ("\nDisconnected. \n");
    } else {
       printf ("\nError: NOT Disconnected. \n");
    }

          //puts (buf);
    }
//***************************************************************************


    closesocket(s);
    close (sock_tx);
    close (sock_rx);

    return 0;
}
```

Code for UR robot scanning sequence

```
Program
  Init Variables
  BeforeStart
    set_tcp(p[0,0,0.20,0.013,-0.0073,0.3904])
    port≔30002
    ip_server≔"192.168.28.51"
    delta_max≔0.400
    delta≔0.0010
    dist_x≔0.250
    dist_y≔0.150
    dist_z≔-0.400
    home≔p[dist_x,dist_y,dist_z,0,0,0]
    home≔pose_trans(Plane_1_var,home)
    open≔socket_open(ip_server,port)
    MoveJ
      home
    dist_x≔0.250
    dist_y_start≔0.050
    dist_z≔-.300
    start_point≔p[dist_x,dist_y_start,dist_z,0,0,0]
    start_point≔pose_trans(Plane_1_var,start_point)
    'MoveJ'
    While open≟ False
      open≔socket_open(ip_server,port)
    string_1≔""
    socket_send_string("Make a new scan?")
    Wait: 0.01
  Robot Program
    finish≔0
    While string_1≟""
      string_1≔socket_read_string()
      Wait: 0.01
    If string_1≟"No"
      Halt
    MoveJ
      start_point
    dist_y≔dist_y_start
    pos_table≔pose_trans(pose_inv(Plane_1_var),get_actual_tcp_pose())
    socket_send_string(finish)
    Wait: 0.15
    socket_send_string(pos_table[1])
    Wait: 0.15
    While string_1≟"Yes"
      If pos_table[1]≤dist_y_start+delta_max
        'Popup'
        'Wait'
        dist_y≔dist_y+delta
        next_point≔p[dist_x,dist_y,dist_z,0,0,0]
        next_point≔pose_trans(Plane_1_var,next_point)
        socket_send_string(finish)
        Wait: 0.15
        MoveL
          next_point
        pos_table≔pose_trans(pose_inv(Plane_1_var),get_actual_tcp_pose())
        socket_send_string(pos_table[1])
        Wait: 0.15
      Else
        finish≔1
        socket_send_string(finish)
        MoveJ
          home
        string_1≔""
    'If pos_table[1]≤dist_y_start+delta_max'
```

Code for Image detection and classification

```
#include "opencv2/core.hpp"
//#include "opencv2/imgproc.hpp"
//#include "opencv2/highgui.hpp"
#include "opencv2/videoio.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <iostream>

#include <cstdio>
//#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

#include <errno.h>


using namespace cv;
using namespace std;

//void drawText(Mat & image);
Point prevPt(-1, -1);
Mat dst_borders, gray_borders,dst_gs,dst_gs_temp;
int selection;

static void onMouse( int event, int x, int y, int flags, void* )
{
    if( x < 0 || x >= dst_borders.cols || y < 0 || y >= dst_borders.rows )
        return;
    if( event == EVENT_LBUTTONUP || !(flags & EVENT_FLAG_LBUTTON) )
        prevPt = Point(-1,-1);
    else if( event == EVENT_LBUTTONDOWN )
    {
      prevPt = Point(x,y);
      if  ((int) dst_borders.at<unsigned char>(prevPt.y,(prevPt.x*3))!=0)
      {
        cout<<"coordinates= "<<prevPt<<endl;
        cout<<"pixel value= "<<(int) dst_borders.at<unsigned char>(prevPt.y,(prevPt.x*3))<<endl;
        cout<<"height value= "<<(((int) dst_borders.at<unsigned char>(prevPt.y,(prevPt.x*3))*6)-
100+30)<<endl;
        selection= (int) (gray_borders.at<unsigned char>(prevPt.y,prevPt.x)-1);
        cout<<"Object "<<selection<<endl;
      }

    }
    else if( event == EVENT_MOUSEMOVE && (flags & EVENT_FLAG_LBUTTON) )
    {
        Point pt(x, y);
        if( prevPt.x < 0 )
            prevPt = pt;
        //line( markerMask, prevPt, pt, Scalar::all(255), 5, 8, 0 );
        line( dst_borders, prevPt, pt, Scalar::all(255), 1, 8, 0 );
        prevPt = pt;
        imshow("Components", dst_borders);
    }
}



int main(int argc, char** argv)
{
    Mat image;
#if 0
    string imageName("image.png");//../../../../../data/HappyFish.jpg"); // by default
    if( argc > 1)
    {
        imageName = argv[1];
    }

    image = imread(imageName.c_str(), IMREAD_COLOR); // Read the file

    if( image.empty() )                     // Check for invalid input
    {
        cout <<  "Could not open or find the image" << std::endl ;
        return -1;
    }

    namedWindow( "Display window", WINDOW_AUTOSIZE ); // Create a window for display.

    imshow( "Display window", image );                // Show our image inside it.

    waitKey(0); // Wait for a keystroke in the window

    destroyWindow("Display window");
```

```
#endif

//    destroyAllWindows();
    //*****************************************************************
//    cout << "Bye byeeee" << endl;
  //capture.release();
  char file_name[25],lines[200],word[200];
  int ch;
  int bottom_offset=30;
  int val_max=5150;
  int maxh=val_max-bottom_offset;
  int offset=100;
  int border=5;
  int stepx=0;
  int stepy=0;
  int i,j,k,coord,imat,jmat,jw,vertex_search;
  int startx=0;
  int error_allowed=21;
//  double yval;//xval,yval,zval;
  unsigned short xval,zval,zmax,zmin,xmax,xmin,xval_pre;
  signed short ymax,ymin,yval;
  int lectures;
  double mat[20][20];

  memset (mat,0, sizeof(mat));
  memset (file_name,0, sizeof(file_name));

//read coordinates X, Y and Z from FILE
  //cout << "Enter the name of file you wish to open\n" << endl;

  printf("Enter the name of file you wish to open:\n");
  fflush(stdout);
  gets(file_name);
  //fgets(file_name,sizeof(file_name),stdin);
    //file_name="ff.txt";

  FILE *f = fopen(file_name, "r");   //open in read mode

  if (f == NULL)
  {
     printf("Error opening file!\n");
     exit(1);
  }

#if 1
//   while( ( ch = fgetc(fp) ) != EOF )
//  for (i=0; i < 20; i++) {
  vertex_search=1;
  while(1){
      j=0;
```

```
jw=0;
memset (lines, '\0', sizeof(lines));
memset (word, '\0', sizeof(word));
while( ( ch = fgetc(f) ) != '\n' ){
lines[j]=ch;
j=j+1;
  if(( ( ch ) != ' ' )&& vertex_search ){
      word[jw]=ch;
      jw=jw+1;
  }
  else if(vertex_search){
       if (strcmp(word,"vertex")==0){
          //printf("Vertex found\n");
          //fflush(stdout);
          memset (word, '\0', sizeof(word));
          jw=0;
          while( ( ch = fgetc(f) ) == ' ' ){
             //printf("ch=*%c*\n",ch);
             //fflush(stdout);
            ;
          }
          word[jw]=ch;
          jw=jw+1;
          while (( ch = fgetc(f) ) != ' ' ){
             //printf("ch=*%c*\n",ch);
             //fflush(stdout);
             //printf("Word progress=*%s*\n",word);
             //fflush(stdout);

             if(ch=='\n'){
               break;
             }
             word[jw]=ch;
             jw=jw+1;
          }
          lectures=atoi(word);
          //printf("String Number=*%s*\n",word);
          lectures=lectures/376;
          printf("Number=*%d*\n",lectures);
          fflush(stdout);
          vertex_search=0;
       }
       else{
       memset (word, '\0', sizeof(word));
          jw=0;
       }
  }
}
//printf("Last lines value=*%s*\n",lines);
//printf("\n");
```

```
      if (strcmp(lines,"end_header")==0){
        //printf("*****EOH found*****\n");
        break;
      }
  }

  //printf("Loop terminated\n");

//  Mat M(2,2, CV_16UC2, Scalar(0,8100));
float factor=1; //1.3

  Mat M(lectures,376, CV_16UC3, Scalar(0,0,0));


   i=0;
   k=0;
   xval=0;
   while( !feof(f) ){

      if (i==376){
        i=0;
      }

      memset (lines, '\0', sizeof(lines));
      j=0;
      coord=0;


      while( ( ch = fgetc(f) ) != '\n'){
         if(feof(f)){
         break;
         }
         if ( ch != '\t' ) {
            lines[j]=ch;
            j=j+1;
         }
         else{
           if (coord==0){
              xval_pre=atoi(lines)/10;
              if (  xval!=xval_pre)
              {
                 k=k+1;
                 xval=xval_pre;
              }
             if (startx==0)
                startx=xval;  //first X value from the sensor lectures

           }
           else{
             yval=atoi(lines);
           }
```

```
            coord=coord+1;
            memset (lines, '\0', sizeof(lines));
            j=0;
          }
       }
       zval=atoi(lines);

       M.at<short>(k-1,(i*3)+0)=xval;//(i*3)+0)=xval;
       M.at<short>(k-1,(i*3)+1)=yval;//xval-startx
       M.at<short>(k-1,(i*3)+2)=zval;


//*************imprimir valores de documento
       //printf("x: % 10d\ty: %10d\tz: % 10d\n",xval,yval,zval);
       //fflush(stdout);
//*************imprimir valores de documento

       i=i+1;
  }

   fclose(f);


//*************imprimir valores de matriz
   //cout << "New Mat = " << endl << " " << M << endl << endl;
   //return -1;
//*************imprimir valores de matriz END

#endif
//    M=M*10;
/// Apply Histogram Equalization
    vector<Mat> spl;
    Mat src, dst, tmp;

    split(M,spl);
    src=spl[2];

//*************imprimir valores de matriz Zval
   //cout << "New Mat = " << endl << " " << spl[1] << endl << endl;
   //return -1;
//*************imprimir valores de matriz Zval END

//  cout << "Val short = " << endl << " " << spl[1].at<short>(0,4) << endl << endl; //Access Yval
SIGNED

//*************imprimir valores de matriz Zval
  // cout << "New Mat = " << endl << " " << src << endl << endl;
  // return -1;
//*************imprimir valores de matriz Zval END

//    equalizeHist( M, dst );
```

```
//*************Filter unknown values
   //cout<<sizeof(src)<<endl;
   //cout<<src.size()<<endl;
   //cout<<src.rows<<endl;
   //cout<<src.cols<<endl;

//***************blur image
     //blur( src, tmp, Size(3,3) );
     //src=tmp;
//***************blur image

   zmax=0;
   zmin=0xffff;// 64
   ymax=0x8000;
   ymin=0x7fff;//8000;  // -32768
   xmax=0;
   xmin=0xffff;

   for (i=0; i < src.rows; i++) {        //FIND MAX VALUE IN MAT
      for (j=0; j < src.cols; j++) {
         if (src.at<short>(i,j)>zmax){
            zmax=src.at<short>(i,j);
         }
         if ((src.at<short>(i,j)<zmin)&&(src.at<short>(i,j)!=0)){
            zmin=src.at<short>(i,j);
         }
         if (spl[1].at<short>(i,j)>ymax){
            ymax=spl[1].at<short>(i,j);
         }
         if (spl[1].at<short>(i,j)<ymin){
            ymin=spl[1].at<short>(i,j);
         }
         if (spl[0].at<short>(i,j)>xmax){
            xmax=spl[0].at<short>(i,j);
         }
         if (spl[0].at<short>(i,j)<xmin){
            xmin=spl[0].at<short>(i,j);
         }
      }
   }
   cout<<"Max z value found="<<zmax<<endl;
   cout<<"Min z value found="<<zmin<<endl;
   cout<<"Max y value found="<<ymax<<endl;
   cout<<"Min y value found="<<ymin<<endl;
   cout<<"Max x value found="<<xmax<<endl;
   cout<<"Min x value found="<<xmin<<endl;

   //return -1;

   Mat Mnew((xmax-xmin+1),(int)((float)(ymax-ymin+1)/10+0.5), CV_16UC1, Scalar(0));
   cout<<"Size new mat= "<<Mnew.size()<<endl;

   for (i=0; i < spl[0].rows; i++) {        //ADJUST UNKNOWN VALUES IN MAT
       for (j=0; j < spl[0].cols; j++) {
           if ((spl[2].at<short>(i,j)>(maxh))||(spl[2].at<short>(i,j)==0)){ //offset of 3mm to filter ground
values of noise -30
              spl[2].at<short>(i,j)=maxh+offset;//4600;//4560;zmax;  //offset between figures and
base plane to find borders
           }
           xval= (spl[0].at<short>(i,j)-xmin);
           yval= (int)((float)(spl[1].at<short>(i,j)-ymin)/10+0.5); //valor en mm redondeado a partir
de 0.5 mm
//           zval= (zmax-spl[2].at<short>(i,j))/6; //factor de normalizacion a 256 para 15 cm de
altura maxima
           zval= ((maxh+offset)-spl[2].at<short>(i,j))/6; //factor de normalizacion a 256 para 15 cm
de altura maxima
           //cout<<"Control 2. line.364: i "<<i<<" j "<<j<<" xval "<<xval<<"\tyval "<<yval<<"\tzval
"<<zval<<endl;
           //cout<<"Size new mat= "<<Mnew.size()<<endl;

           if( ( Mnew.at<short>(xval,yval))<zval ){
           //if( (( Mnew.at<short>(xval,yval))==0)||( ( Mnew.at<short>(xval,yval))<zval )){
               Mnew.at<short>(xval,yval)=zval;
           }
       }
   }

   //***************blur image
         //blur( src, tmp, Size(3,3) );
         //src=tmp;
   //***************blur image

   //***************ZOOM image x2
         //pyrUp( src, tmp, Size( src.cols*2, src.rows*2 ));
   //***************ZOOM image x2

//***************normalize image (

     //normalize(tmp, dst, 0, 65535, NORM_MINMAX);
     //bitwise_not ( src, dst );
     int val=1;

   Mnew.convertTo(dst, CV_8UC1);

   tmp=dst;

   namedWindow( "GS0 proportional unfiltered",WINDOW_AUTOSIZE);//
WINDOW_NORMAL|WINDOW_KEEPRATIO ); // Create a window for display.
   imshow( "GS0 proportional unfiltered", dst );              // Show our image inside it.
   //waitKey(0);
   //return -1;
   //waitKey(0); // Wait for a keystroke in the window



   dst_borders = Mat::zeros(dst.rows, dst.cols, CV_8UC3);
   dst_gs = Mat::zeros(dst.rows, dst.cols, CV_8UC1);
   dst_gs_temp= Mat::zeros(dst.rows, dst.cols, CV_8UC1);
   Mat only_borders = Mat::zeros(dst.rows, dst.cols, CV_8UC3);
   gray_borders = Mat::zeros(dst.rows, dst.cols, CV_8UC1);//fondo negro
   Mat rect_gradient = Mat::zeros(dst.rows, dst.cols, CV_8UC1);//fondo negro
//    Mat gray_borders = Mat::ones(dst.rows, dst.cols, CV_8UC1)*255;//fondo blanco
   Mat poly_borders = Mat::zeros(dst.rows, dst.cols, CV_8UC3);
   //cout<<"size Mat: "<<dst_borders.size()<<endl;
//    return -1;
   Mat element = getStructuringElement(
MORPH_RECT,Size(3,3));//Point(1,1));//MORPH_RECT //MORPH_CROSS
//MORPH_ELLIPSE

   namedWindow( "GS proportional", WINDOW_NORMAL|WINDOW_KEEPRATIO ); // Create
a window for display.

   //imshow( "GS proportional", dst );              // Show our image inside it.
   //waitKey(0);
   dilate(tmp,dst,element);
   tmp=dst;
   //imshow( "GS proportional", dst );              // Show our image inside it.
   //waitKey(0);
   erode(tmp,dst,element);
   tmp=dst;
   //imshow( "GS proportional", dst );              // Show our image inside it.
   //waitKey(0);
   erode(tmp,dst,element);
   tmp=dst;
   //imshow( "GS proportional", dst );              // Show our image inside it.
   //waitKey(0);

   element = getStructuringElement( MORPH_RECT,Size(9,3));  //eliminate noise in the border
of the images

   erode(tmp,dst,element);
   tmp=dst;
   //imshow( "GS proportional", dst );              // Show our image inside it.
   //waitKey(0);

   element = getStructuringElement( MORPH_RECT,Size(9,3));

   dilate(tmp,dst,element);
   tmp=dst;
   //imshow( "GS proportional", dst );              // Show our image inside it.
   //waitKey(0);
```

```
  for (i=0; i < dst.rows; i++) {        //copy in dst_borders original filtered image
     for (j=0; j < dst.cols; j++) {
        dst_gs.at<unsigned char>(i,j)= dst.at<unsigned char>(i,j);
        dst_borders.at<unsigned char>(i,(j*3)+0)= dst.at<unsigned char>(i,j);
        dst_borders.at<unsigned char>(i,(j*3)+1)= dst.at<unsigned char>(i,j);
        dst_borders.at<unsigned char>(i,(j*3)+2)= dst.at<unsigned char>(i,j);
     }
  }


  imshow( "GS proportional", dst_gs );                // Show our image inside it.
  //waitKey(0);
  //return -1;
  //***************blur image                   //NOT NEEDED TO BLUR
    //blur( tmp, dst, Size(3,3) );
    //tmp=dst;
    //blur( tmp, dst, Size(3,3) );
    //tmp=dst;
  //***************blur image


  Canny( dst, dst, val, val*3, 3 );

  dilate(tmp,dst,element);   //filter stage for borders found, to close objects completely
  tmp=dst;
  erode(tmp,dst,element);
  tmp=dst;

  //imwrite( "objetos9_border.jpg", dst );

  //namedWindow( "GS proportional borders",
WINDOW_AUTOSIZE);//WINDOW_NORMAL|WINDOW_KEEPRATIO ); // Create a window for
display.
  //imshow( "GS proportional borders", dst );              // Show our image inside it.
  //waitKey(0); // Wait for a keystroke in the window


  //src = src > 1;
  namedWindow( "Source", 1 );
  imshow( "Source", dst );

  vector<vector<Point> > contours;
  vector<Vec4i> hierarchy;

  findContours( dst, contours, hierarchy,
  CV_RETR_EXTERNAL, CV_CHAIN_APPROX_SIMPLE );
```

```
//create variables for next section
   vector<vector<Point> > poly_contours( contours.size() );
   vector<Rect> boundRect( contours.size() );
   vector<Point2f>center( contours.size() );
   vector<vector<Point2f> > rect_vtx( contours.size() );
   Point2f vtx[4];
   vector<float>radius( contours.size() );
   vector<vector<Point2f> > boundTriang( contours.size() );
   vector<RotatedRect> boundRect2( contours.size() );

   vector<double> area_poly( contours.size() );
   vector<double> area_rect( contours.size() );
   vector<double> area_triang( contours.size() );
   vector<double> area_circle( contours.size() );

   vector<bool> is_object( contours.size() );
   vector<bool> is_rectangle( contours.size() );
   vector<bool> is_triangle( contours.size() );
   vector<bool> is_circle( contours.size() );
   vector<int> object_identified( contours.size() );

   Mat rot_M;

   bool closed=true;
   int start_x;
   int start_y;
   int end_x;
   int end_y;
//create variables for next section

   // iterate through all the top-level contours,
   // draw each connected component with its own random color
   int idx = 0;
   for( ; idx >= 0; idx = hierarchy[idx][0] )
   {
        //define color for each object
       Scalar color( rand()&255, rand()&255, rand()&255 );
       Scalar color2( idx+1);//asigna valores escala grises a cada objeto encontrado ( de 0 a n
objetos)
       //Scalar color2( idx*255/contours.size(), idx*255/contours.size(), idx*255/contours.size() );
       //Scalar color2( (idx+1)*255/(contours.size()+1));//asigna valores escala grises a cada
objeto encontrado (0-255)

   //******************************************************************************
   // Calculate data required for each object found

       approxPolyDP(contours[idx], poly_contours[idx], 3, closed);                  //approximate
polygon with obtained contours
       boundRect[idx] = boundingRect( Mat(poly_contours[idx]) );                    //bounding straight
rectangle
       minEnclosingCircle( (Mat)poly_contours[idx], center[idx], radius[idx] );
```

```
        boundRect2[idx]=minAreaRect((Mat)poly_contours[idx]);
        boundRect2[idx].points(vtx);
        minEnclosingTriangle((Mat)poly_contours[idx], boundTriang[idx]);

        //****draw surrounding rectangle
        //rectangle( poly_borders, boundRect[idx].tl(), boundRect[idx].br(), color, 1, 8, 0 );
        //****draw surrounding rectangle

        //****draw surrounging circle
        circle( poly_borders, center[idx], (int)radius[idx], Scalar(255, 0, 0), 1, 8, 0 );
        //****draw surrounging circle

        // Draw the triangle
        //************draw triangles surrounding
        for( i = 0; i < 3; i++ )
          line(poly_borders, boundTriang[idx][i], boundTriang[idx][(i+1)%3], Scalar(0, 255, 0), 1,
LINE_AA);
        //************draw triangles surrounding

        //**** Draw the bounding box tilted
        for( i = 0; i < 4; i++ ){
        line (poly_borders,vtx[i],vtx[(i+1)%4],Scalar(0, 0, 255),1,8);//LINE_AA);
        rect_vtx[idx].push_back(vtx[i]);
        }
        //cout<<endl<<"poly "<<idx<<endl<<rect_vtx[idx];

        //**** Draw the bounding box tilted

        drawContours( poly_borders, poly_contours, idx, color, 1, 8, hierarchy );//CV_FILLED

        drawContours( dst_borders, contours, idx, color, 1, 8, hierarchy );//CV_FILLED
        drawContours( only_borders, contours, idx, color, 1, 8, hierarchy);//CV_FILLED
        drawContours( gray_borders, contours, idx, color2 , -1, 8, hierarchy );//CV_FILLED

        //draw center of objects (circles and cross) in center of each object
        circle( poly_borders,(Point) boundRect2[idx].center , 5, Scalar(255,0,0), 1, 8, 0 );
        int length_cross=1;
        line (poly_borders, (Point){boundRect2[idx].center.x-
length_cross,boundRect2[idx].center.y},

(Point){boundRect2[idx].center.x+length_cross,boundRect2[idx].center.y},Scalar(255,0,0),1,8);//
LINE_AA);
        line (poly_borders,(Point) {boundRect2[idx].center.x,boundRect2[idx].center.y-
length_cross},
        (Point)
{boundRect2[idx].center.x,boundRect2[idx].center.y+length_cross},Scalar(255,0,0),1,8);//LINE_
AA);
        //draw center of objects (circles and cross) in center of each object END

        //write number to each object in POLYBORDERS
        char buffer[100] = {0};
        putText(poly_borders, itoa(idx,buffer,10),
        (Point) {(int)boundRect2[idx].center.x+4,(int)boundRect2[idx].center.y-4},
         FONT_HERSHEY_SIMPLEX, 0.4, color, 1, 8, false );   //to_string(idx)

         //write number to each object in COMPONENTS
         putText(dst_borders, itoa(idx,buffer,10),
        (Point) {(int)boundRect2[idx].center.x+4,(int)boundRect2[idx].center.y-4},
         FONT_HERSHEY_SIMPLEX, 0.4, color, 1, 8, false );   //to_string(idx)
         //write number to each object in COMPONENTS END

         //***********calculate different information
        cout<<"********************************"<<endl;
        cout<<"rect "<<idx<<" size= "<<(Point) boundRect2[idx].size<<endl;//
(width,height)
        cout<<"rect "<<idx<<" width= "<< (float)boundRect2[idx].size.width<<endl;//
(width,height)
        cout<<"rect "<<idx<<" height= "<< (float) boundRect2[idx].size.height<<endl;//
(width,height)
        //cout<<"rect "<<idx<<" X= "<< boundRect2[idx].center.x<<endl;//(Point)
        //cout<<"rect "<<idx<<" Y= "<< boundRect2[idx].center.y<<endl;//(Point)
        cout<<"rect "<<idx<<" center= "<< (Point) boundRect2[idx].center<<endl;//
(Point)Position
        cout<<"rect "<<idx<<" angle= "<< boundRect2[idx].angle<<endl;//                (angle)

        area_poly[idx] = contourArea(poly_contours[idx]);
        area_rect[idx]= contourArea(rect_vtx[idx]);
        area_triang[idx]= contourArea(boundTriang[idx]);
        area_circle[idx]= radius[idx]*radius[idx]*3.14159;
        cout << "area poly "<<idx <<"= "<< area_poly[idx] << endl;
        cout << "area rectangulo "<<idx <<"= "<< area_rect[idx] << endl;
        cout << "area triangulo "<<idx <<"= "<< area_triang[idx] << endl;
        cout << "area circulo "<<idx <<"= "<< area_circle[idx] << endl;

        if (area_poly[idx]>100) //value for an object of 1x1cm area
        {
          is_object[idx]=true;
          if((abs(area_poly[idx]-area_circle[idx])/area_poly[idx]*100)<error_allowed)   //umbral de
error del 5% de coincidencia de areas
          {
              is_rectangle[idx]=false;
              is_triangle[idx]=false;
              is_circle[idx]=true;
          }
          else if((abs(area_poly[idx]-area_rect[idx])/area_poly[idx]*100)<error_allowed)   //umbral
de error del 5% de coincidencia de areas
          {
             is_rectangle[idx]=true;
              is_triangle[idx]=false;
              is_circle[idx]=false;
          }
          else if((abs(area_poly[idx]-area_triang[idx])/area_poly[idx]*100)<error_allowed)   //umbral
de error del 5% de coincidencia de areas
          {
              is_rectangle[idx]=false;
              is_triangle[idx]=true;
              is_circle[idx]=false;
          }


        }
        else
        {
          is_object[idx]=false;
          is_rectangle[idx]=false;
          is_triangle[idx]=false;
          is_circle[idx]=false;

        }

        cout << "***Figure "<<idx <<" is ";
        if (is_object[idx]==false)
          cout<<"NOT and object ***"<<endl;
        else if (is_rectangle[idx]==true)
          cout<<"a rectangle ***"<<endl;
        else if (is_triangle[idx]==true)
          cout<<"a triangle ***"<<endl;
        else if (is_circle[idx]==true)
          cout<<"a circle ***"<<endl;
        else
           cout<<"an UNKNOWN object ***"<<endl;

      if (is_rectangle[idx]==true)   //algoritmo para rotacion de prisma cuadrado y analisis de
gradiente
      {
         rot_M=getRotationMatrix2D((Point) boundRect2[idx].center,boundRect2[idx].angle,1);
//Rotation matrix, scale factor=1
         cout<<rot_M<<endl;
         dst_gs_temp= Mat::zeros(dst_gs.rows, dst_gs.cols, CV_8UC1);
         for (i=0; i < dst_gs.rows; i++) {        //copy in dst_borders original filtered image
             for (j=0; j < dst_gs.cols; j++) {
                 if ((int)gray_borders.at<unsigned char>(i,j)==(idx+1)) //object n has GS value n+1
(zero reserved for background)
                 {
                    dst_gs_temp.at<unsigned char>(i,j)=dst_gs.at<unsigned char>(i,j);
                 }
             }
         }

         warpAffine(dst_gs_temp, rect_gradient, rot_M, Size
(rect_gradient.cols,rect_gradient.rows));//(rect_gradient.cols,rect_gradient.rows));//, int
```

```
flags=INTER_LINEAR, int borderMode=BORDER_CONSTANT, const Scalar&
borderValue=Scalar())
        namedWindow( "Components", WINDOW_NORMAL|WINDOW_KEEPRATIO );

        cout<<"center: "<<(Point2f)boundRect2[idx].center<<endl;
        cout<<"X: "<<boundRect2[idx].center.x<<" Y: "<<boundRect2[idx].center.y<<endl;

        start_x=(int)(boundRect2[idx].center.x-boundRect2[idx].size.width/2+border);
        start_y=(int)(boundRect2[idx].center.y-boundRect2[idx].size.height/2+border);
        end_x=(int)(boundRect2[idx].center.x+boundRect2[idx].size.width/2-border);
        end_y=(int)(boundRect2[idx].center.y+boundRect2[idx].size.height/2-border);

        cout<<"startx= "<<start_x<<", starty= "<<start_y<<", endx= "<<end_x<<", endy=
"<<end_y<<endl;

        cout<<"pixel value x0y0= "<<(int) rect_gradient.at<unsigned char>(start_y,start_x)<<endl;
        cout<<"pixel value x0y1= "<<(int) rect_gradient.at<unsigned char>(end_y,start_x)<<endl;
        cout<<"pixel value x1y1= "<<(int) rect_gradient.at<unsigned char>(end_y,end_x)<<endl;
        cout<<"pixel value x1y0= "<<(int) rect_gradient.at<unsigned char>(start_y,end_x)<<endl;
        //line (rect_gradient,(Point2f)(boundRect2[idx].center),(Point){boundRect2[idx].center.x-
boundRect2[idx].size.width/2+border,boundRect2[idx].center.y-
boundRect2[idx].size.height/2+border},Scalar(255),1,8);//LINE_AA);

        int pxn;    //px n
        int pxn1;   //px n+1
        int contpx=0;   //pixels counter
        int gradx[end_x-start_x];
        int grady[end_y-start_y];

        //*************gradient X
        //*****gradient analisys top horizontal line
        for (i=start_x;i<(end_x);i++)
        {
           //cout<<(int) rect_gradient.at<unsigned char>(start_y,i)<<",";
           pxn=(int) rect_gradient.at<unsigned char>(start_y,i);
           if (pxn!=0)
           {
             pxn1=(int) rect_gradient.at<unsigned char>(start_y,i+1);
             if (pxn1!=0)
             {
               gradx[contpx]=pxn1-pxn;
               contpx=contpx+1;
             }
           }
        }
        //cout<<(int) rect_gradient.at<unsigned char>(start_y,i)<<endl;
        cout<<"Top horizontal Gradients: ";

        int cont_pos=0;
        int cont_neut=0;
        int cont_neg=0;
        int actual=-1;
        int last=-1;

        //for (i=0;i<contpx;i++)
          //cout<<gradx[i]<<",";

//        cout<<endl<<endl;

        for (i=0;i<contpx;i++)
        {
          //cout<<gradx[i]<<",";

          if (gradx[i]<0)
          {
             if (last!=1)
                cont_neg=cont_neg+1;
             last=0;
          }
          else if (gradx[i]>0)
          {
             if (last!=0)
                cont_pos=cont_pos+1;
             last=1;
          }
          else
          {
             cont_neut=cont_neut+1;
             last=2;
          }

          if ((last==0)            )//&&(actual!=0))
          {
             if(cont_neg>4)
             {
                //actual=0;
                cont_pos=0;
                cont_neut=0;
                cont_neg=1;
                //cout<<"neg ";    //prints every other neg
                if (actual!=0)
                {
                   cout<<"neg ";   //prints only one consecutive neg
                   actual=0;
                }
             }
          }
          else if ((last==1)   )//&&(actual!=1))
          {
             if(cont_pos>4)
             {
                cont_neg=0;
          cont_neut=0;
          cont_pos=1;
          if (actual!=1)
          {
            cout<<"pos ";
            actual=1;
          }

          //cout<<"pos ";
        }
      }
      else if (last==2)
      {
        if(cont_neut>4)
        {
          if(cont_neg>3)
          {
            cont_neut=0;
          }
          else if(cont_pos>3)
          {
            cont_neut=0;
          }
          else
          {
           if (actual!=2)
             {
               cout<<"neut ";
               actual=2;
             }

           cont_neut=0;
           cont_pos=0;
           cont_neg=0;
          }
        }
      }
    }
    cout<<endl<<endl;

    //*****gradient analisys top horizontal line END

    contpx=0;   //pixels counter
    memset (gradx, '\0', sizeof(gradx));
//    int gradx[end_x-start_x];

    //*****gradient analisys central horizontal line
    for (i=start_x;i<(end_x);i++)
    {
       //cout<<(int) rect_gradient.at<unsigned char>(boundRect2[idx].center.y,i)<<",";
       pxn=(int) rect_gradient.at<unsigned char>(boundRect2[idx].center.y,i);
```

```
            if (pxn!=0)
            {
              pxn1=(int) rect_gradient.at<unsigned char>(boundRect2[idx].center.y,i+1);
              if (pxn1!=0)
              {
                gradx[contpx]=pxn1-pxn;
                contpx=contpx+1;
              }
            }
          }
          //cout<<(int) rect_gradient.at<unsigned char>(boundRect2[idx].center.y,i)<<endl;
          cout<<"Central horizontal Gradients: ";

          cont_pos=0;
          cont_neut=0;
          cont_neg=0;
          actual=-1;
          last=-1;

          //for (i=0;i<contpx;i++)
            //cout<<gradx[i]<<",";

//          cout<<endl<<endl;

          for (i=0;i<contpx;i++)
          {
            //cout<<gradx[i]<<",";             // desactivar print gradient*************************

            if (gradx[i]<0)
            {
              if (last!=1)
                cont_neg=cont_neg+1;
              last=0;
            }
            else if (gradx[i]>0)
            {
              if (last!=0)
                cont_pos=cont_pos+1;
              last=1;
            }
            else
            {
              cont_neut=cont_neut+1;
              last=2;
            }

            if ((last==0)          )//&&(actual!=0))
            {
              if(cont_neg>4)
              {
                //actual=0;
```

```
      cont_pos=0;
      cont_neut=0;
      cont_neg=1;
      //cout<<"neg ";    //prints every other neg
      if (actual!=0)
      {
         cout<<"neg ";   //prints only one consecutive neg
         actual=0;
      }
   }
}
else if ((last==1)   )//&&(actual!=1))
{
   if(cont_pos>4)
   {
      cont_neg=0;
      cont_neut=0;
      cont_pos=1;
      if (actual!=1)
      {
         cout<<"pos ";
         actual=1;
      }

      //cout<<"pos ";
   }
}
else if (last==2)
{
   if(cont_neut>4)
   {
     if(cont_neg>3)
     {
        cont_neut=0;
     }
     else if(cont_pos>3)
     {
        cont_neut=0;
     }
     else
     {
      if (actual!=2)
         {
           cout<<"neut ";
           actual=2;
         }

      cont_neut=0;
      cont_pos=0;
      cont_neg=0;
     }
```

```
            }
          }
        }
        cout<<endl<<endl;
        //*****gradient analisys central horizontal line END

        contpx=0;   //pixels counter
        //int gradx[end_x-start_x];
        memset (gradx, '\0', sizeof(gradx));

        //*****gradient analisys bottom horizontal line
        for (i=start_x;i<(end_x);i++)
        {
           //cout<<(int) rect_gradient.at<unsigned char>(end_y,i)<<",";
           pxn=(int) rect_gradient.at<unsigned char>(end_y,i);
           if (pxn!=0)
           {
              pxn1=(int) rect_gradient.at<unsigned char>(end_y,i+1);
              if (pxn1!=0)
              {
                 gradx[contpx]=pxn1-pxn;
                 contpx=contpx+1;
              }
           }
        }
        //cout<<(int) rect_gradient.at<unsigned char>(end_y,i)<<endl;
        cout<<"Bottom horizontal Gradients: ";

        cont_pos=0;
        cont_neut=0;
        cont_neg=0;
        actual=-1;
        last=-1;

        //for (i=0;i<contpx;i++)
          //cout<<gradx[i]<<",";

//        cout<<endl<<endl;

        for (i=0;i<contpx;i++)
        {
          //cout<<gradx[i]<<",";

          if (gradx[i]<0)
          {
             if (last!=1)
                cont_neg=cont_neg+1;
             last=0;
          }
          else if (gradx[i]>0)
          {
```

```
    if (last!=0)
      cont_pos=cont_pos+1;
    last=1;
  }
  else
  {
    cont_neut=cont_neut+1;
    last=2;
  }

  if ((last==0)           )//&&(actual!=0))
  {
    if(cont_neg>4)
    {
      //actual=0;
      cont_pos=0;
      cont_neut=0;
      cont_neg=1;
      //cout<<"neg ";    //prints every other neg
      if (actual!=0)
      {
        cout<<"neg ";   //prints only one consecutive neg
        actual=0;
      }
    }
  }
  else if ((last==1)   )//&&(actual!=1))
  {
    if(cont_pos>4)
    {
      cont_neg=0;
      cont_neut=0;
      cont_pos=1;
      if (actual!=1)
      {
        cout<<"pos ";
        actual=1;
      }

      //cout<<"pos ";
    }
  }
  else if (last==2)
  {
    if(cont_neut>4)
    {
     if(cont_neg>3)
     {
       cont_neut=0;
     }
     else if(cont_pos>3)
```

```
       {
         cont_neut=0;
       }
       else
       {
        if (actual!=2)
           {
             cout<<"neut ";
             actual=2;
           }

        cont_neut=0;
        cont_pos=0;
        cont_neg=0;
       }
     }
   }
 }
 cout<<endl<<endl;         //*****gradient analisys bottom horizontal line END
 //*************gradient X END

 //waitKey(0);

/* for (i=start_x;i<end_x;i++)
    cout<<(int) rect_gradient.at<unsigned char>(boundRect2[idx].center.y,i)<<",";
 cout<<endl;
 for (i=start_x;i<end_x;i++)
    cout<<(int) rect_gradient.at<unsigned char>(end_y,i)<<",";
 cout<<endl;*/
 cout<<"********************************"<<endl;
 //*************gradient Y
 //*****gradient analisys first vertical  line
 //for (i=start_y;i<end_y;i++)
 //   cout<<(int) rect_gradient.at<unsigned char>(i,start_x)<<",";
 contpx=0;   //pixels counter
 for (i=start_y;i<end_y;i++)
 {
     //cout<<(int) rect_gradient.at<unsigned char>(start_y,i)<<",";
     pxn=(int) rect_gradient.at<unsigned char>(i,start_x);
     if (pxn!=0)
     {
        pxn1=(int) rect_gradient.at<unsigned char>(i+1,start_x);
        if (pxn1!=0)
        {
           grady[contpx]=pxn1-pxn;
           contpx=contpx+1;
        }
     }
 }
 //cout<<(int) rect_gradient.at<unsigned char>(start_y,i)<<endl;
 cout<<"First vertical Gradients: ";
```

```
cont_pos=0;
cont_neut=0;
cont_neg=0;
actual=-1;
last=-1;

//for (i=0;i<contpx;i++)
  //cout<<gradx[i]<<",";

//        cout<<endl<<endl;

for (i=0;i<contpx;i++)
{
  //cout<<gradx[i]<<",";

  if (grady[i]<0)
  {
    if (last!=1)
      cont_neg=cont_neg+1;
    last=0;
  }
  else if (grady[i]>0)
  {
    if (last!=0)
      cont_pos=cont_pos+1;
    last=1;
  }
  else
  {
    cont_neut=cont_neut+1;
    last=2;
  }

  if ((last==0)            )//&&(actual!=0))
  {
    if(cont_neg>4)
    {
      //actual=0;
      cont_pos=0;
      cont_neut=0;
      cont_neg=1;
      //cout<<"neg ";    //prints every other neg
      if (actual!=0)
      {
        cout<<"neg ";   //prints only one consecutive neg
        actual=0;
      }
    }
  }
  else if ((last==1)   )//&&(actual!=1))
```

```
          {
            if(cont_pos>4)
            {
              cont_neg=0;
              cont_neut=0;
              cont_pos=1;
              if (actual!=1)
              {
                cout<<"pos ";
                actual=1;
              }

              //cout<<"pos ";
            }
          }
          else if (last==2)
          {
            if(cont_neut>4)
            {
             if(cont_neg>3)
             {
               cont_neut=0;
             }
             else if(cont_pos>3)
             {
               cont_neut=0;
             }
             else
             {
              if (actual!=2)
                {
                  cout<<"neut ";
                  actual=2;
                }

              cont_neut=0;
              cont_pos=0;
              cont_neg=0;
             }
            }
          }
        }
        cout<<endl<<endl;
//*****gradient analisys first vertical  line END

        contpx=0;   //pixels counter
        memset (grady, '\0', sizeof(grady));
//        int gradx[end_x-start_x];
        //for (i=start_y;i<end_y;i++)
        //   cout<<(int) rect_gradient.at<unsigned char>(i,start_x)<<",";
        //*****gradient analisys central vertical  line
        for (i=start_y;i<end_y;i++)
        {
           //cout<<(int) rect_gradient.at<unsigned char>(boundRect2[idx].center.y,i)<<",";
           pxn=(int) rect_gradient.at<unsigned char>(i,boundRect2[idx].center.x);
           if (pxn!=0)
           {
              pxn1=(int) rect_gradient.at<unsigned char>(i+1,boundRect2[idx].center.x);
              if (pxn1!=0)
              {
                 grady[contpx]=pxn1-pxn;
                 contpx=contpx+1;
              }
           }
        }
        //cout<<(int) rect_gradient.at<unsigned char>(boundRect2[idx].center.y,i)<<endl;
        cout<<"Central vertical Gradients: ";

        cont_pos=0;
        cont_neut=0;
        cont_neg=0;
        actual=-1;
        last=-1;

        //for (i=0;i<contpx;i++)
          //cout<<gradx[i]<<",";

//        cout<<endl<<endl;

        for (i=0;i<contpx;i++)
        {
          //cout<<gradx[i]<<",";

          if (grady[i]<0)
          {
             if (last!=1)
                cont_neg=cont_neg+1;
             last=0;
          }
          else if (grady[i]>0)
          {
             if (last!=0)
                cont_pos=cont_pos+1;
             last=1;
          }
          else
          {
             cont_neut=cont_neut+1;
             last=2;
          }
```

```
if ((last==0)           )//&&(actual!=0))
{
   if(cont_neg>4)
   {
      //actual=0;
      cont_pos=0;
      cont_neut=0;
      cont_neg=1;
      //cout<<"neg ";    //prints every other neg
      if (actual!=0)
      {
         cout<<"neg ";   //prints only one consecutive neg
         actual=0;
      }
   }
}
else if ((last==1)   )//&&(actual!=1))
{
   if(cont_pos>4)
   {
      cont_neg=0;
      cont_neut=0;
      cont_pos=1;
      if (actual!=1)
      {
         cout<<"pos ";
         actual=1;
      }

      //cout<<"pos ";
   }
}
else if (last==2)
{
   if(cont_neut>4)
   {
     if(cont_neg>3)
     {
        cont_neut=0;
     }
     else if(cont_pos>3)
     {
        cont_neut=0;
     }
     else
     {
      if (actual!=2)
         {
           cout<<"neut ";
           actual=2;
         }
         cont_neut=0;
         cont_pos=0;
         cont_neg=0;
        }
      }
   }
 }
 cout<<endl<<endl;
 //*****gradient analisys central vertical  line END

 contpx=0;   //pixels counter
 //int gradx[end_x-start_x];
 memset (grady, '\0', sizeof(grady));
 //for (i=start_y;i<end_y;i++)
 //   cout<<(int) rect_gradient.at<unsigned char>(i,start_x)<<",";

 //*****gradient analisys last vertical line
 for (i=start_y;i<end_y;i++)
 {
    //cout<<(int) rect_gradient.at<unsigned char>(end_y,i)<<",";
    pxn=(int) rect_gradient.at<unsigned char>(i,end_x);
    if (pxn!=0)
    {
       pxn1=(int) rect_gradient.at<unsigned char>(i+1,end_x);
       if (pxn1!=0)
       {
          grady[contpx]=pxn1-pxn;
          contpx=contpx+1;
       }
    }
 }
 //cout<<(int) rect_gradient.at<unsigned char>(end_y,i)<<endl;
 cout<<"Last vertical Gradients: ";

 cont_pos=0;
 cont_neut=0;
 cont_neg=0;
 actual=-1;
 last=-1;

 //for (i=0;i<contpx;i++)
   //cout<<gradx[i]<<",";

//         cout<<endl<<endl;

 for (i=0;i<contpx;i++)
 {
   //cout<<gradx[i]<<",";

   if (grady[i]<0)
```

```
{
  if (last!=1)
    cont_neg=cont_neg+1;
  last=0;
}
else if (grady[i]>0)
{
  if (last!=0)
    cont_pos=cont_pos+1;
  last=1;
}
else
{
  cont_neut=cont_neut+1;
  last=2;
}

if ((last==0)          )//&&(actual!=0))
{
  if(cont_neg>4)
  {
    //actual=0;
    cont_pos=0;
    cont_neut=0;
    cont_neg=1;
    //cout<<"neg ";    //prints every other neg
    if (actual!=0)
    {
      cout<<"neg ";   //prints only one consecutive neg
      actual=0;
    }
  }
}
else if ((last==1)   )//&&(actual!=1))
{
  if(cont_pos>4)
  {
    cont_neg=0;
    cont_neut=0;
    cont_pos=1;
    if (actual!=1)
    {
      cout<<"pos ";
      actual=1;
    }

    //cout<<"pos ";
  }
}
else if (last==2)
{
```

```
              if(cont_neut>4)
              {
                if(cont_neg>3)
                {
                  cont_neut=0;
                }
                else if(cont_pos>3)
                {
                  cont_neut=0;
                }
                else
                {
                 if (actual!=2)
                    {
                      cout<<"neut ";
                      actual=2;
                    }

                 cont_neut=0;
                 cont_pos=0;
                 cont_neg=0;
                }
              }
           }
         }
         cout<<endl<<endl;
         //*****gradient analisys last vertical line END
         //*************gradient Y END
/*         for (i=start_y;i<end_y;i++)
           cout<<(int) rect_gradient.at<unsigned char>(i,start_x)<<",";
         cout<<endl;
         for (i=start_y;i<end_y;i++)
           cout<<(int) rect_gradient.at<unsigned char>(i,boundRect2[idx].center.x)<<",";
         cout<<endl;
         for (i=start_y;i<end_y;i++)
           cout<<(int) rect_gradient.at<unsigned char>(i,end_x)<<",";
         cout<<endl;*/



         //rectangle analized from each rectangle
         line
(rect_gradient,(Point){end_x,start_y},(Point){start_x,start_y},Scalar(255),1,8);//LINE_AA);
         line
(rect_gradient,(Point){start_x,start_y},(Point){start_x,end_y},Scalar(255),1,8);//LINE_AA);
         line
(rect_gradient,(Point){start_x,end_y},(Point){end_x,end_y},Scalar(255),1,8);//LINE_AA);
         line
(rect_gradient,(Point){end_x,end_y},(Point){end_x,start_y},Scalar(255),1,8);//LINE_AA);
```

```
        imshow( "Components", rect_gradient );
        waitKey(0);
        //return -1;

      }

    }

//*********************************************************************************
// Show images of obtained results


    namedWindow( "Components", WINDOW_NORMAL|WINDOW_KEEPRATIO );
    imshow( "Components", dst_borders );

    namedWindow( "Only borders", WINDOW_NORMAL|WINDOW_KEEPRATIO );
    imshow( "Only borders", only_borders );

    namedWindow( "INT borders", WINDOW_NORMAL|WINDOW_KEEPRATIO );
    imshow( "INT borders", gray_borders );

    namedWindow( "Poly borders",WINDOW_NORMAL|WINDOW_KEEPRATIO
);//WINDOW_NORMAL|WINDOW_KEEPRATIO      //WINDOW_AUTOSIZE
    imshow( "Poly borders", poly_borders );

    cout<<endl<<"Objects found: "<<contours.size()<<endl;

    setMouseCallback( "Components", onMouse, 0 );


    waitKey(0); // Wait for a keystroke in the window
    return 0;
}
```