VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

GBENGA MONDAY OMOSEKEJI

INDUSTRIAL VISION ROBOT WITH RASPBERRY PI USING PIXY CAMERA

Stereo Vision System

Information Technology, Embedded System Engineering
2018

ACKNOWLEDGEMENTS

I am most grateful to the Almighty God for the successful completion of my Bachelor’s degree program. Also, I am grateful to the Finnish government, Vaasa University of Applied Sciences, the Information Technology Group for giving me the privilege to learn and apply my knowledge to solving real life problem at no cost.

My appreciation goes to my supervisor and Lecturer – Jukka Matila for His professional, care and inspiration in this project. Also, I will like to thank all the lecturers who have taught me all through my study.

Big thanks to my lovely fiancée and longtime friend; Olwaseun Salami, for the love, understanding and encouragement at all time. Special thanks to my Mother Mrs Omosekeji for her prayers and continue support and to my late Father Mr Omosekeji thank you for showing me what needed to be successful in life. And to my siblings; Ilemobola, Adedowo, Adegbite, Atinuke, Olabisi, Esther, Adeniyi, Mercy, and Adetola and their families, thanks for the love we share and encouragement all through my studies. Also to my friends in Vaasa for your support.

Finally, my appreciation goes to Professor Manuel Silver and Germano Veiga and my wonderful colleague Alexandre Filipe at ISEP Portugal for their assistance in times of need in this project.

ABSTRACT

Author	Gbenga Monday Omosekeji
Title	Industrial Vision Robot with Raspberry Pi using Pixy Camera
Year	2018
Language	English
Pages	106
Name of Supervisor	Jukka Matila

Industrial robots are not human, they are machines. They are programmable manipulator devices which can move tools or parts via a set sequence of motions. In addition, they can be reprogrammable, that is, the robot's action can be modified by changing the control settings without replacing the hardware. They add some characteristics of traditional machines likewise as characteristics of machine operators. For an operator, it is easy to be taught to do a new task. But, for a machine, a task can be repeated for prolonged times with great precision.

This project focused at developing a Robot Vision system using a combination of low-cost camera hardware and computer algorithms to enable robots to process visual data from the world. The stereo vision algorithm which consists of two cameras, and the developed application are able to calculate a 3D position from s 2D detected object. In addition, the detection algorithm based on color differences was used by the cameras which enable 2D object tracking and outputted data coordinates of the object being detected. Then, the 3D object position is produced through the calculated 2D object data coordinates, which made ready for robot teaching.

Furthermore, the developed application was based on OpenCV API in C++, which was an interest in the development of this project. The use of this was to treat the image capture by the cameras. TIY software with modification was used to do the object tracking. BOOST is a set of C++ libraries that provide image processing, and linear algebra functionalities. This library was the appropriate choice because of the reliance of TIY on it, and some other aspects of it that are important to the rest of the project.

Finally, the test results showed that the project was successfully developed. In addition, with the developed project, my expertise in embedded system programming has been consolidated and I have obtained further knowledge in the field of robotics and computer vision.

Keywords: Stereo Vision, OpenCV API, TIY, C++, Boost, MatLab, Linux

TABLE OF CONTENTS

Table of Figures

List of Tables

ABBREVIATIONS

1-G	One Gaussian
2-D	Two-Dimensional
3-D	Three-Dimensional
API	Application Programming Interface
ARM	Advanced RISC Machine
ASIC	Application Specific Integrated Circuits
CPU	Central Processing Unit
DC	Direct Current
DSP	Digital Signal Processor
FET	Field Effect Transistor
FIFO	First In, Fist Out
FPGA	Field Programmable Gate Array
GFLOPS	Giga Floating-point Operations Per Second
GigE	Gigabit Ethernet
GLCM	Gray Level Co-occurrence Matrices
GMM	Gaussian Mixture Model
GND	Ground
GNU	GNU’s Not Unix
GPIO	General Purpose Input Or output
GPU	Graphical Processing Unit

HD	High Definition
HDMI	High Definition Multimedia Interface
HSV	Hue, Saturation, Value
I/O	Input/Output
I2C	Inter-Integrated Circuit
ICSP	In-Circuit Serial Programming
IP	Internet Protocol
IPC	Inter Process Communication
L*a*b, L*u*v	Typical color space used in MATLAB program
LED	Light Emitting Diode
OpenCV	Open source Computer Vision Library
RAM	Random Access Memory
RGB	Red, Green, Blue
SCL	Serial Clock Line
SD	Secure Digital
SDA	Serial Data Line
SoC	System on a Chip
SS	Slave Select
TCP/IP	Transmission Control Protocol/Internet Protocol
TIY	Track It Yourself
UART	Universal Asynchronous Receiver Transmitter

USB	Universal Serial Bus
VGA	Video Graphics Array

1. INTRODUCTION

Robots used in industrial work are not human, they are machines. They are programmable manipulator devices which can move tools or parts via a set sequence of motions. In addition, a robot can be reprogrammable, that is, a robot's action can be modified by changing the control settings without replacing the hardware. They have some characteristics of traditional machines as well as characteristics of machine tool operators. For an operator, it is easy to be taught to do a new task. But, for a machine, a task can be repeated for prolonged times with great precision.

Today, industrial robots are not used frequently in medium-size and small sectors or companies because of the costs and the need for skilled labor to program them. Therefore, it is necessary to develop cheaper robotic systems, that are easier to program for these sectors.

In addition, in recent years, hardware prices have reduced. However, it costs more to acquire robots and other industrial equipment. One other factor worthy of consideration is the cost of the skilled labor required to program them. Therefore, to increase the business productivity and to increase the ease of reprogramming or programming robots, it is relevant to employ innovative methods using the technology available as tools.

The robot programming is made up of more conventional interfaces which are difficult, prone to errors which also takes too much time. Thus, the use of more user-friendly interfaces as the proposed interface in this project. Also, the industrial vision robot in this work enabled the programming of a robot for various purposes with the use of an intuitive method; using only one pointer to the robot teaching through a computer vision system as input.

More so, the system uses a computer vision such that, 2-D (two-dimensional) to obtain the needed pointer position in three-dimensional (3-D) space, software is used to process data. The data obtain is eventually used to design trajectories for the robotic teaching.

1.1. Contextualization

This thesis is inspired by Mika Billing, Master of Science (Technology), a senior lecturer, University of Applied Science, School of Technology, Mechanical Engineering department, Vaasa, Finland. The project covers areas such as computer vision systems, robotics, programming and microcontrollers, which are interesting and useful areas for both manufacturing industries and miscellaneous areas. As my expertise has been in the field of Embedded System and minor in Telecommunication, this project added value to my academic and professional development as well as my personal life.

My knowledge in programming and microcontroller is reasonable, but basic in robotics, computer vision or machine vision, therefore this project deepened my knowledge in the known area and added new skills, which improve my training.

1.2. Objectives

Based on the previous gaps in knowledge, this project succinctly develops a robot system for an industrial robot. As a low cost hardware camera system, the robot is fast with easily programmable robotics system.

Firstly, the required software and hardware to develop a marker tracking system using computer vision was chosen. During the teaching, a trajectory is then calculated for the robot from the marker position and movement in sequence.

In this project, two cameras were utilized because a 3-D positional sensing is required, hence, the cameras needed calibration. Subsequently, the application that collects data from the two cameras was developed. The data from the two cameras acquired from the application is used in triangulation and obtaining the 3D position of the object.

Lastly, the application that determines the desired trajectory of the robot controller is required.

1.3. Thesis Report Organization

The report contains eight chapters and four annexes. chapter 1 is an introduction to the developed application project while chapter 2 contains the information needed when developing a 3D robot vision system, and in chapter 3, tracking systems are analysed, while the information concerning computer vision, machine vision are presenter in chapter 4. In chapter 5, is the system architecture, which explains the hardware and software technologies used in this project and the 6 chapter is the project implementation while chapter 7 contains test and system validation, lastly, the chapter 8 presented the conclusion of the project and the prospective work.

2. DEVELOPING ROBOT 3D VISION SYSTEM

This chapter presents some ways to develop an efficient robot 3D vision system and the likely problem that may occur during their development. It also shows proposed stereo vision methodology used in this project taken into an account that detection of an object via colors should be user friendly and as simple as possible.

Blind robots need some help once one interrupts a variable the way they are presented, to enhance flexibility, robots need visual understanding. This brings to existence the machine vision. This mean, when users are finding a way to save money or need more flexibility, choosing robotics vision application over a blinded robotics application is a great idea.

Thus, the following questions should be answered as prerequisite for automation of the vision application. These questions include the following:

- What is to be moved and how will I present it?
- Are the sizes of the parts determined?
- Does the orientation matter (up/down)?
- What is the importance of radial orientation?
- How will the system acquire the part and how will the space be a factor?
- How will the parts integrate with the robot after acquiring it?
- Stereo vision using disparity map algorithm

2.1 Knowing the type of camera to used

Three-dimensional, are the real-world scenes a camera faces today. In most cases, when exposed to different depths, objects would appear to each other in form of two-dimensional mapped scene of the sensor of the camera. In figure 1, is a presentation of a figure obtained from Middlebury picture dataset as illustrated in /38/. The picture succinctly provides details that the motorbike at a view from the background of the photo is about two meters closer to the camera when compared to the shelf at the background. Watch point 1 and 2 as marked in the figure [1]. Point 1 as illustrated in the red box place in the background of the figure shows to be adjacent to the second fork of the motorcycle in the image, even with two-meter distance from the camera. Owing to the power of perspective in the human brain, it makes the process of deciding the depth easy from a 2D scene. However, for a forward mounted camera on top of the workshop desk, the ability to conceptualise the perspective becomes a challenge compared to human brain /38/.

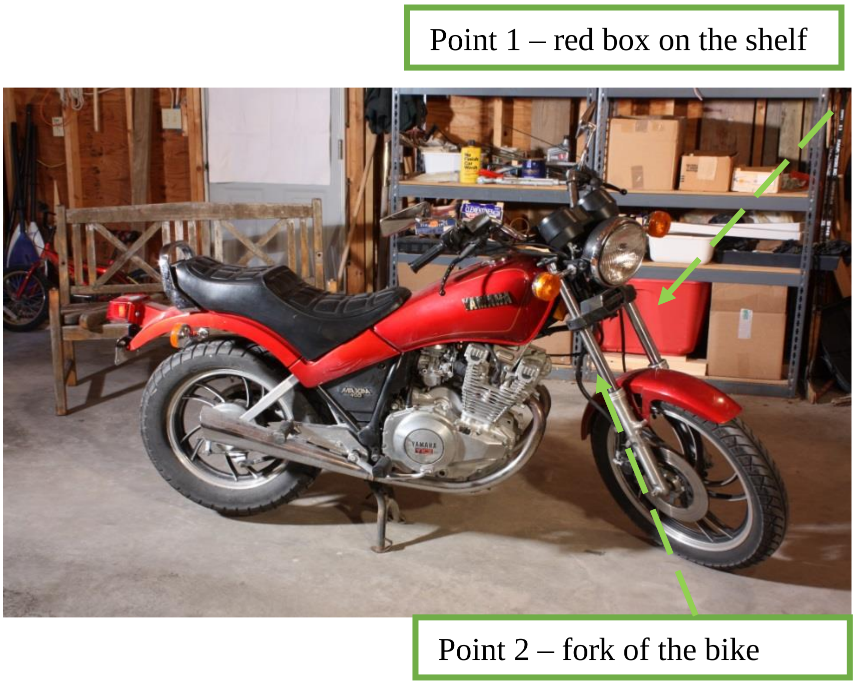


Figure 1: the foreground of the bike is closer to camera than the shelf, meanwhile, the appearance of all the object is in 2D view. The photo is from 2014 Middlebury database */38/*

Using a single mounted camera sensor to capture a video that needs to be processed and analysed, the system is called monocular system, while with two cameras separated from each other is called a stereo vision. The table [1] below makes a succinct comparison between a stereo vision system with the primary characteristics of a monocular system of a camera.

Table 1: High level system attributes comparison between stereo and monocular camera /38/

Comparison Parameter	Monocular camera System	stereo-vision system
Number of lenses, assembly and image sensors	1	2
physical size of the system	2.1” x 2.0” x 1.4	Two small assemble separated by 1.5m away
frame rate	50 times per second	50 time per second

system is reliable for:	object detection (colors) and tell the position of the object	Object detection and calculate the coordinate of the object, save it as data file for 3D processing.
image processing requirements	Medium	High
software and algorithm complexity	High	Medium
Reliability of detecting object for quick analysis	Medium	High
System cost	1x	1.5x

2.2 Monocular System

Monocular cues are used by humans such as texture gradients, texture variations, occlusion, interposition, light and shading, known object size, defocus, haze and so on. For instance, /7/: when viewed at different distance, the texture of many object will appear or look different. For a texture gradient, it is meant to capture the distribution of edges. Additionally, it helps in determine the depth. This system can do many things reasonable, well; it can identify object, line, and color of the path with good accuracy. However, the challenge is that it is not robust and reliable when used to calculate the 3D scene of the world from the 2D frame as received from the monocular camera sensor.

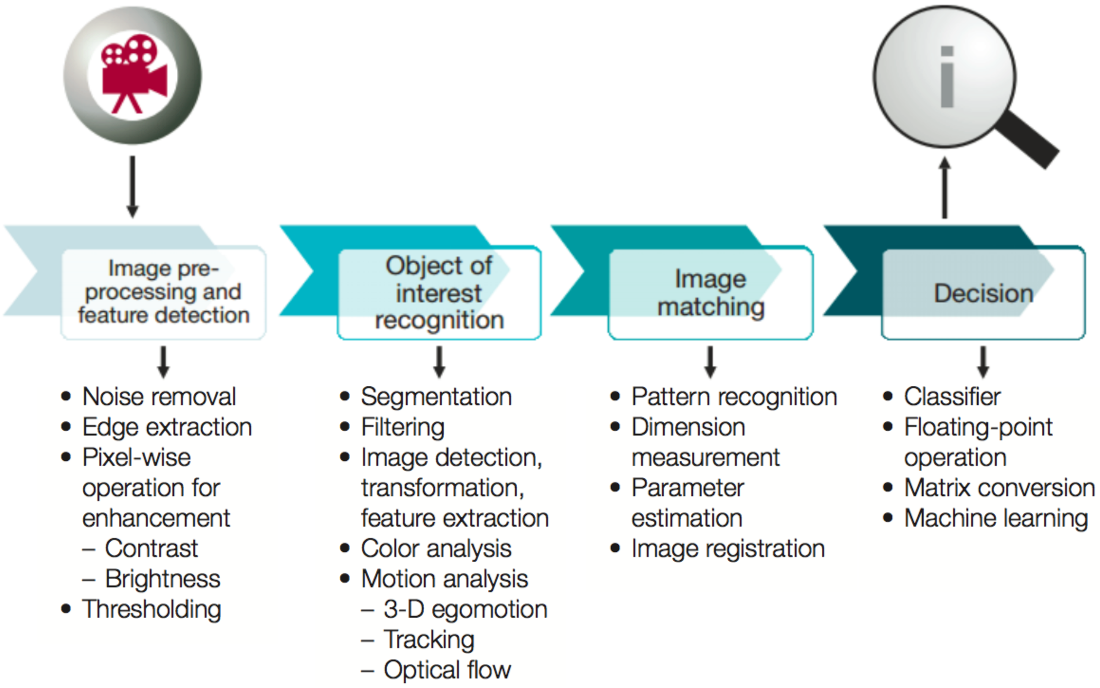


Figure 2: Process of image analysis and high level algorithm flow */39/*

Before going deep in analysing this problem, figure [2], describes the process and algorithms used to analyse image (video) frame received from a camera sensor at high level. The image processing step, which is the first stage, entails different filters which are run on the image with the intention of removing noise from the sensor and other unnecessary information. The conversion of the received format BAYER data obtained from the sensor of the camera to either RBG or YUV also takes place at this stage. Importantly, YUV or RGB can also be analysed by the subsequent steps. It is based on preliminary feature extraction, among others, haar, histogram of orientated gradients edges, Gabor filters.

In addition, there is need for evaluation of areas of interest. As such, the second and third stage critically analyse the image which entails running the algorithm such as pattern recognition, block machine and segmentation. However, with final stage, there is utilisation of the part information as well as feature of the data that has been created from the previous stage. The aim is to generate well-form analysis decision regarding the class of the object one is interested in.

2.3 Measuring object distance from the camera

Recently, object based on distance measurement technique has become a field of major research interest in computer vision and robotics. The two approaches among other, methods will be discus, which are monocular vision and stereo vision based distance measurement technique.

2.4.1 Measuring Object Distance Using Monocular Camera (Fixed)

There are several methods presented in the literature to measure object distance for a moving object. Zhang et al /30/, created a three steps method or algorithm to calculate the position of 3-D of the object marked a camera(s) coordinate frame. This system measures the distance between the object(s) feature such as a point on the object, and the principal such as the central point in the image plane according to the calculated area in the object. Therefore, taken to account the algorithm suggested by Zhang et al, the intrinsic camera(s) must be calibrated first.

2.4.2 Stereo Vision Method

The method of stereo vision uses two cameras to find the depth and the disparity map with the use of complex method. This technique is highly accurate but because of the simultaneous processing of the images of the same object, its requires extensive computation time. In addition, the development of this system is costly, because it requires the use of two cameras.

Moreover, because of the accuracy of the stereo vision system, it is chosen as the proposed technique used in this project to extract 3D data from digital images which was used to examine the position of object in two images, because of its similarity to human biological system, they both have identical features. As an example, taking the same environment, a two-eyed human being can have a slightly different view. It was observed that when a stereo system having two cameras located at a known distance is used, it took images of the scene simultaneously /13/.

2.4.3 Chapter 2: Conclusion

In this chapter, was explained the background knowledge required for this thesis which includes, what a 3D vision system is, the monocular vision system and stereo vision technique and why stereo vision chosen as the proposed method used in this project. The next chapter describes the detail of a tracking systems and the used technique.

3. TRACKING SYSTEMS

This chapter is a presentation of the overview on the tracking system, also it defines the purpose of tracking system used in this project.

3.1 Tracking System

A tracking system is frequently used in robot vision to observe moving objects and supplying its position data in a timely ordered sequence for further processing. A few examples of tracking systems are sonar tracking, vehicle tracking, video tracking, air traffic control tracking and so on.

Different kind of sensors can be used to carry out the tracking, some examples are listed in /46/:

Mechanical Sensing: this includes some form of direct physical linkage between the target and the environment. The use of encoder and potentiometers can be applied to measure rotation, such as positioning of an object, or rotation of a motor;

Sonars: using the triangulation system, at least three sonars must be adopted in obtaining the state, position and orientation of any object. Sonars typically have a long range, but they are sensitive to noise;

Optical sensor: this system depends on measurement of emitted or reflected light. Usually, the tracking system depends on optics which consists of one or more optical sensors and one or more light sensors. Common types of optical sensors include cameras with fixed filters to capture a particular spectrum of light, such as ultraviolet light. The light source can be artificially generated light or ambient light.

Inertial sensors: sensors like gyroscopes and accelerometers are in the form of integrated circuits, they are commonly available and can be easily integrated with electronic components. They have low latency and are less sensitive to environmental noise, but their accuracy is poor. They measure acceleration; which makes the calculation of the coordinate and orientation of the object possible.

Thus, 3D tracking of one or more objects is implemented in this project, specifically its coordinates, position and orientation-tracking, with the use of video-based object tracking method.

This system can handle several applications, for example, robot stereo vision, robot leaning by demonstration, human computer interaction and so on. It uses makers to perform tracking or the other way around.

Therefore, without a marker, the video tracking systems have two distinct types: a) model-based, b) appearance-based. The model-based system is used to obtain the orientation (pose) and the position of the observer when partial or complete model of the environment pre-exist. The system steadily provides quick fixed to a problem. The disadvantages are that it is expensive and depend on good visual information which is usually provided by multi-camera systems. While an appearance-based system depends on lower data processing power and hardware complexity but provide discrete and limited number of coordinate or position of detected object /29/.

3.2 Object **Tracking**

In robot vision, object tracking is quite challenging due to various factors: for example, non-rigid object structures, occlusions, camera motion and quick changes in both the object and the scene. Object tracking can be useful in the following topics /30/:

- Traffic monitoring: simultaneous traffic inspection to direct traffic flows.
- Surveillance systems: monitoring change information or behaviour to detect unusual activities.
- Vehicle navigation: for real-time path planning and obstacle avoidance capabilities in robotics.
- Video indexing: for the retrieval and recovery of videos in databases.

3.3 Problem in Object Tracking

To get a desired object in the tracking system can be challenging, taken into an account the accuracy in guessing trajectory of a moving object. Tracking object can be difficult because of /30/

- Loss of information during projection from a 3D world to a 2D image,

- Complex object motion,
- Complex object shape,
- Full and partial object occlusion,
- Presence of noise in the images,
- Scene illumination change,
- Complex attribute of an object, like Articulated or non-rigid object,
- Real time processing requirements.

3.4 Feature for Tracking an Object

Typically, it is the algorithm that tracks objects with the use of an amalgamation feature to perform their functions. The features used commonly include /30/:

Colour: Two physical factors influence the colour of an object; the surface reflectance characteristic of the object, and the spectral power distribution of the light source. Typically, the RGB colour spectrum is utilised to depict colour in image processing. The RGB colour spectrum is not optimal in depicting the colour of an object because it is not perceptually uniform, i.e. the colours in the RGB colour spectrum do not tally with the perception of colour by the average human eye. Furthermore, RGB dimensions are correlated. Contrariwise, *l*a*b* and *l*u*v* (where "*l*" is for lightness and "*a*" and "*b*" for the colour opponents green-red and blue-yellow) which explains mathematically all colours in the 3D, are perceptually even colour spectra, although HSV (Hue, Saturation, Value) can be approximated as a uniform colour spectrum. The colour spectra presented here are susceptible to noise. In the final analysis, since none of the different color spaces in clearly more efficient than the other, a combination of these color spaces is used in tracking /30/.

Edges: The edges of many objects tend to result in fluctuation in image intensity. These changes can be tracked by using edge detection. Edges are known to be less susceptible to changes in illumination vise-visa the color characteristics of an object. For this reason, algorithms with the ability to track the limits of the given objects will most likely, use edges as a representative characteristic. The Canny Edge detector is commonly used in the regard because of its simplicity and accuracy /10/.

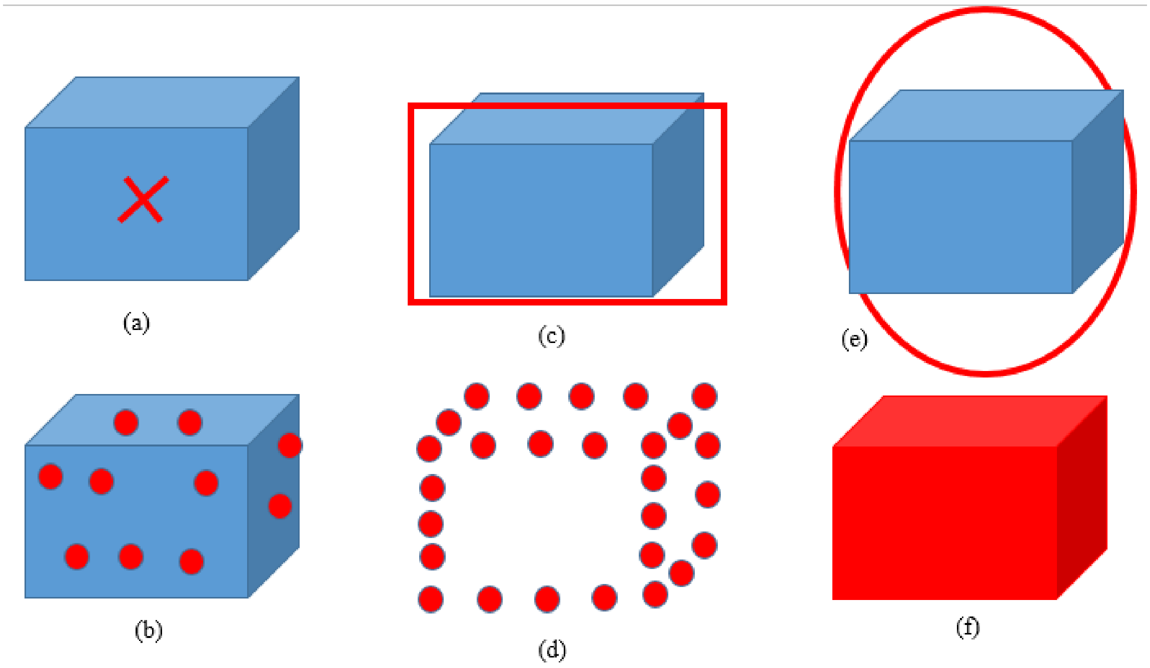


***Figure 3: types of object in different shape, a: centroid b: points set, c: rectangle, d: object contours, e: elliptical, f: silhoutte object** /38/*

Optical Flow: This is a rich field of displacement vectors with the ability to define the translation of every pixel in consecutive frames /30/. When it comes to segmentation of motion-based and tracking application, the adopted feature is Optical flow.

Texture: This is a measure of the variation of intensity of a given surface. Such variations identifies features like the regularity and smoothness. Again, texture needs a processing step that helps in the generation of the descriptors in contrast of colour. Such texture entail Gray-Level Co-occurrence Matrices (GLCM’s). Just like it is with edge features, studies found that text features are found to be less sensitive to illumination changes when a comparison is made to colour /30/.

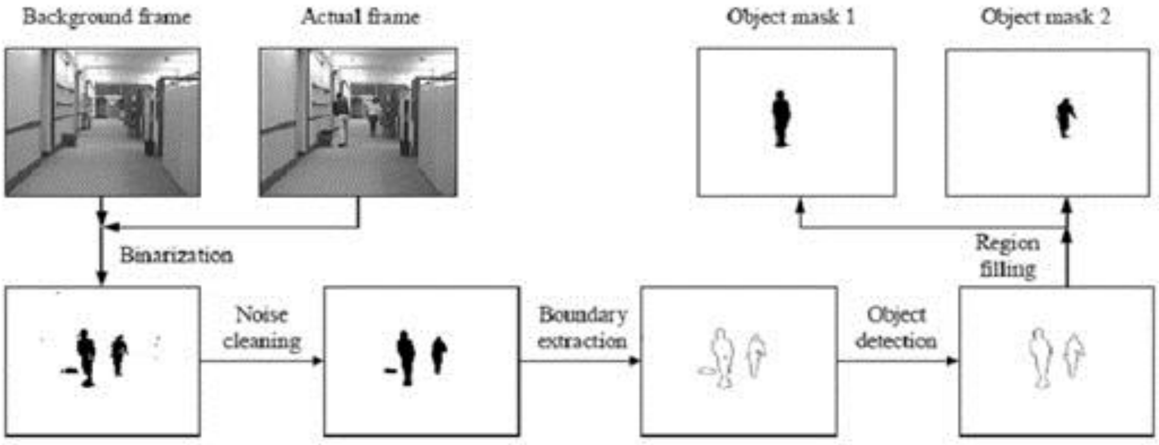
3.5 Algorithms or Methods

In this part, the main algorithms used in this project are shown.

3.5.1 Algorithm: background subtraction

As the name suggests, a background subtraction algorithm is the way of separating out foreground objects from the background in a sequence of video frames. This method relies on the background of a scene at rest, which later assists to detect the targeted object.

In addition, background is recognized as background model or background image. Typically, in this project, the camera is stationary and the identification of an object is carried out by detecting areas on a frames sequence which changes in position over time, and then calculate the difference between the current frame and the image background /21/.



Source: Goldmann et al.

Figure 4: Background subtraction method

Therefore, some important factors need to be considered in background subtraction algorithm:

Selecting the size of the feature (a pixel, a cluster or a block);

Select the type of feature or attribute (attribute such as: stereo feature, color, feature texture, feature corners, and the feature movement).

Background modelling or representation is the first step in background subtraction also is a core tool in motion analysis. The central idea behind this method is to generate or create a probabilistic copy of the static scene that compares the current input to execute subtraction. The aim of straightforward approach of background modelling is to obtain an image which does not entail any moving objects. Nevertheless, there are some problems inherent in this approach. In addition, because of the change in brightness of an object being removed or inserted from the scene or reflection from some mirrored surface, etc, in some environment the background may be change in a dramatic way or may not be available /6/.

When the background has been obtained, the next background is initialization which does the generation or extraction in respect to the initialization of the model.

In the process of tracking method/algorithm, it is important to enable the maintenance of the background which relies on approaches needed to adapt the model with regard to the change in the scene within a given period of time. Accordingly, the cost of maintenance of the three core issues entail the following:

1. Maintenance rules: these are used to adapt image background.
2. Leaning rate: it is used to determine the speed of the adjustment in correlation to occurrence change in the scene.
3. Adaptation rate: this determines the frequency of the maintenance.

As a result, the foreground detection is entailed when classifying the pixel which may be considered as background or foreground pixel. Which mean, it is made up of detecting object in the scene. There are lots of foreground detection algorithm, some of them are: one Gaussian (1-G), basic motion detection, Gaussian Mixture Model (GMM), kernel density estimation or the codebook, MinMax inter-frame difference /6/.

3.5.2 Tracking system bases on color

Color model is one of the easiest and the quickest ways for tracking and object from an image frame, it allows the separation of a specific color in a frame and return its orientation and the position. It has low costs and is useful for other modest systems of its processing power.

Basically, every unique color would be blue, red and green value which help in the understanding of the extent to which each channel is mixed into the specific color. Therefore, compulsory the minimum and maximum values for each of the channels to perform tracking needs to define, putting into consideration that the brightness is not perfectly reliable either the color of the object, in this case there can be small difference in the recognition by the tracking system /11/.

Therefore, once the color(s) to be detected has been determined, it is then possible to apply several algorithms or methods to process the frames and then return the objects being detected. One of the easiest ways of doing this is scanning each pixel of the image so as to identify the pixels with the required color, it is necessary to group these pixels supposing they apply to an object with more than one pixel. Lastly, it then returns the position of the object. Although, for the case of the Pixy camera - the camera used in this project - the center position of the object will be returned. In addition, the method of noise filter can be used to boost the work of the algorithms.

3.6 Track It Yourself (TIY)

To ease the development of 3D tracking system, Track It yourself library was used. TIY is an open source library that runs on Window and Linux environments and it can detect several objects simultaneously. TIY tools can work with varieties of cameras; therefore, one can mount an inexpensive computer vision system from simple webcams to industrial cameras. It comes with ready-to-use examples which does not need additional hardware and that also have an operational cycle approximately 10ms. With the use of TIY to build a simple computer vision, one needs two cameras that has infrared detection capability plus an object with a marker to detect /49/.

The 3D tracking object is done with the use of triangular from feedback of the two used cameras. To detect the 3D position object in it, it demands placing two markers and save the initial output as template. When the TIY is in process, it compares the current coordinate of the marker for each present time with the order of the previous saved template. Through this process, it determines the coordinate of the object in 3D space.

After the camera and the object have been calibrated, the software is responsible for producing the object in 3D format. The TIY method is used in this project to give an easy way to get the 3D positioning of a maker /15/.

Here are some of the important features of TIY /15/.

- It supports Aravis GigE (Gigabit Ethernet) and OpenCV (Open Source Computer Vision) camera;
- It is capable of capture image, record video and log data;
- It can send data via a network to multiple computer devices;
- The 2D points or the videos saved as file can be used as input, instead of cameras. In this project data is passed to the Track It Yourself by file with object in two dimensional points.

TIY software can be installed and configure by following the online documentation in, /15/ also in Annex [B].

3.6.1 Chapter 3: Conclusion

In this chapter, was described some parts of little tracking systems method, there are several algorithms and methodology approaches to this, it depends on the type of system one is implementing and the literature that can be found on the internet. Also, Track It Yourself, the proposed technique used for 3D tracking system was explained. Next chapter describes the general overview of computer vision, machine vision and their applications also the financial justification machine vision and camera calibration will be discussed.

4. COMPUTER VISION

This chapter presents some computer vision topics. Today the field of computer vision is constantly evolving, and the use of this technology is being driven especially in industrial control, robotics, and artificial intelligence also for innovation purposes. Computer vision is being used in the industry today to increase productivity, increasing quality; reducing cost. Also, it provides comfort and safety for the workers.

Computer vision system can be divide into two implementations: the use of special hardware like Digital Signal Processors (DSP), Application Specific Integrated Circuits (ASIC) or Field Programmable Gate Array (FPGA) also the use of image processing software on a computer. The use of a devoted hardware system is of greatest advantage because of the computer power and the lower processing time, but because of the low adaptability in the case of ASIC or DSP is of the disadvantage /3/.

A computer vision system can be used in different firm as shown the figure [5], the use of mobile robot can be used in the robotics field for deciding and planning path, which enable the mobile robot to move freely in a specified space. The knowledge of the surrounding environment is provided by the computer vision system plus other type of sensors and the information gathered by this means is sent to the robot for decision making.

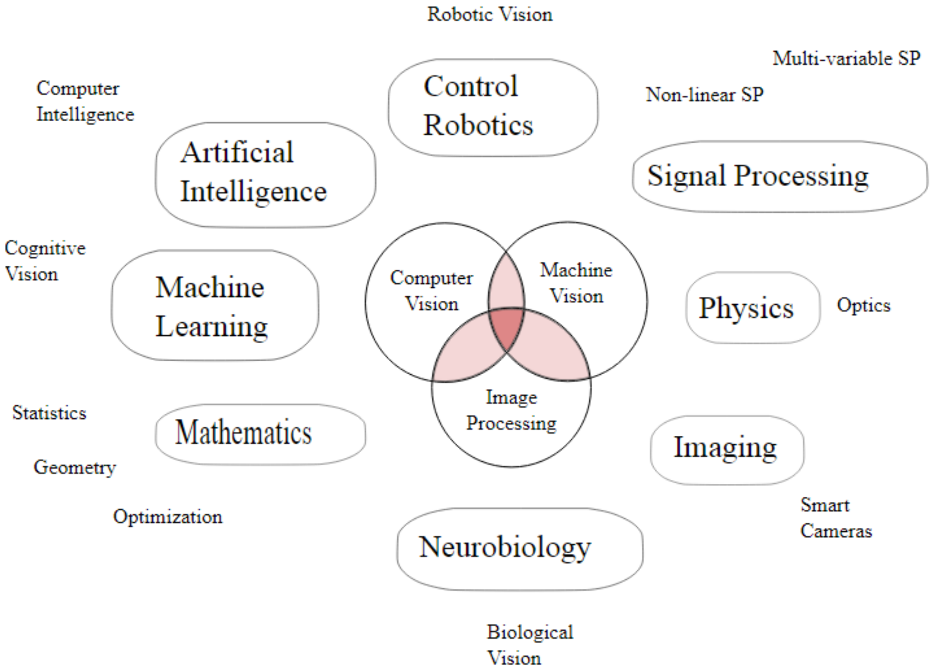


Figure 5:Computer Vision Application in several area /12/

4.1 Machine Vision

In the industrial sector, computer vision is known as machine vision, usually it is used for processing and quality control.

4.1.1 Machine Vision Overview

Machine vision consists of the automatic acquisition and analysis of real images to obtain desired data for evaluating or controlling a specific part. The image can be can be obtained from X-ray, infrared, visible light and so on. It serves as a swap of the human optic sense and judgemental competence which includes computer and video camera to carried out an inspection. Some of the key points of machine vision are: Automated, Acquisition, Analysis and data.

4.1.2 Machine Vision Operation

Typically, for image processing, an industrial machine system vision contains one or more cameras, an interface system to transfer the image to the PC or processor, lenses for the camera, image processor which may be substitute by cameras that contain image processing, an interface implemented with software to provide information and receive command for the world [figure 6].

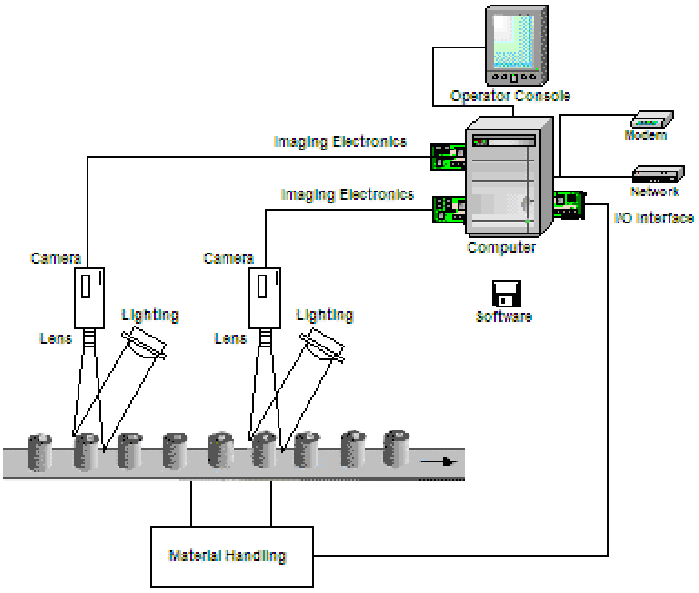


Figure 6: overview of Vision Sensors operation /45/

The captured image by the camera sensor may be characterised by a 2-D array of energy levels. Each element of the image is known as a "pixel" and these pixels form rows and columns covering the whole area of the image. In addition, a pixel contains one energy level, for monochrome image it can be classified as grey level while in a color image; the information which is describing the color of a pixel is more difficult. The vital point is that a pixel cannot be subdivided into any smaller region either of another color or grey level /49/.

For each pixel, the amount of energy captured by the camera must be digitised, and this process can be done internally by the camera. The analogue levels with continuous variation produced by the camera must be represented by a finite number of steps. For a monochrome image, 8 bits per pixel are mostly used to give 256 grey levels while in more demanding application, 14 bits are used. But for a color image, like RGB, it gives about 16 million colors, but more or less bits are used a representation of each pixel.

The processor interprets the image in two essential steps: segmentation and analysis. Segmentation: depending on each application, determines which parts of the image that needs interpretation and which are background. While, for analysis, when the satisfactory image has been segmented, then the processor makes different tests and measurements on the object of interest /23/.

Machine Vision Applications

Machine vision application functions include /23/:

- **Inspection measurement:** it doe the conformity checking, fault detection, structure light and other triangular techniques, One, 2D and 3D measurement (in respect to stereo, fringe method)
- **Recognition:** such as object and components. The object recognition can be used to identify a specific object in a digital videos or image.
- **Guidance:** this could be predetermined guidance or continuous guidance. Predetermine guidance, this is a situation whereby overhead camera takes a picture of an event and the vision system controls the robot to the direction to drop down or pick up the object. In addition, continuous guidance usually involves a mounted camera on a robot arm, which the direction is always control by a vision system.
- **Special systems:** this is based on industrial specification.

4.1.3 Financial Justification of Machine Vision Systems

There are lots of justification for using machine vision system, among others are: avoid scraps, because of the use of inspection systems with machine vision somehow will lead to a very short pay-off period. Therefore, automatic inspection system provide means of checking the quality parameters that is different from perfect and the chance of making a correction before these parameters exceed the limit is provided. Thus, every product inspected are regarded as scraps.

Other financial justification is the reduction of cost with human resources. There are lots of task done by human operator that can be replaced by machine vision systems and the savings are more significant for multiple shift workers. In addition, it offers the savings related to no annual pay increases and recruitment. Thus, those involved in working with machine vision system are often required to be more highly skilled.

Lastly, the machine vision systems provide savings in term of cost related to product quality. It provides inspection methods which are more objective, reliable and consistent than humans. In addition, optimising the use of materials, suppliers' quality and ensuring good quality of finished products saves costs. Also, the reduction in the cost of warranty repair work and customers confidence is increased which can lead to repeat orders and greater market share /23/.

4.2 Capture Image

In human vision, the images captured allow the perception of the surrounding space. This scenario is applied to robotics, computer vision system also the industrial systems, the objectives are the same in terms of perception of the surrounding space. It is possible to change the shape, texture, size, position and color at the field of view from the images. However, the question is to determine how the human eye work /49/.

When the light falls on objects, in figure [5] (the object is a face), it reflected back and different rays make different angle when is been passed via the lens and an invert image of the object has been created on the back wall. From the last figure, it denotes that the object has been decoded by the brain.

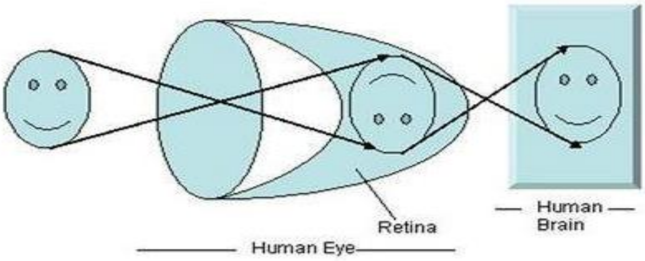


Figure 7: Image Formation on Camera /44/

But, in the digital camera, a change-coupled device (CCD) array of sensors is used for the image transformation. The sensors sense the value and convert them to electric signals. The CCD is typically in the shape of a rectangular grid or an array. It is like each cell in a matrix which contains a sensor that sense the photon intensity.

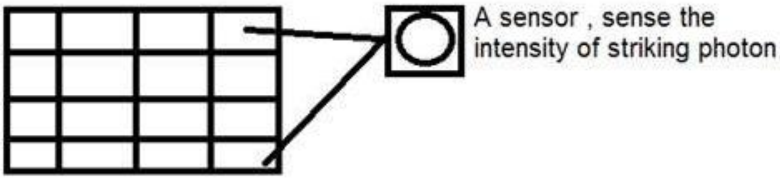


Figure 8: Change coupled device in array shape /44/

Every sensor of the CCD rectangular grid is an analog sensor. A small electrical charge is generated in the photo sensor when photo of light strike on the chip, therefore due to two-dimensional structure of the CCD rectangular grid, it can then generate a complete image from CCD array /44/.

The technique of generating an image, either in case of digital camera or human eye, consists of projecting a 3D space on 2D surface, which lead to lost information about the depth. The re-creation of three dimensions to two dimensions is called perspective transformation or projection.

4.2.1 Perspective Projection

When an object is close to a human eye, it looks bigger but when they are far it looks smaller, which similar in digital camera, the bigger the lens the higher the brightness (vice versa). This process is called perspective in general way. In addition, projection is the transfer of an object from one state to another. The processes of projection control the conversion of three dimensions world to two-dimension objects or images. The convex lens usage also increases the brightness of the image captured /44/.

Image with convex lens shows in figure [9]

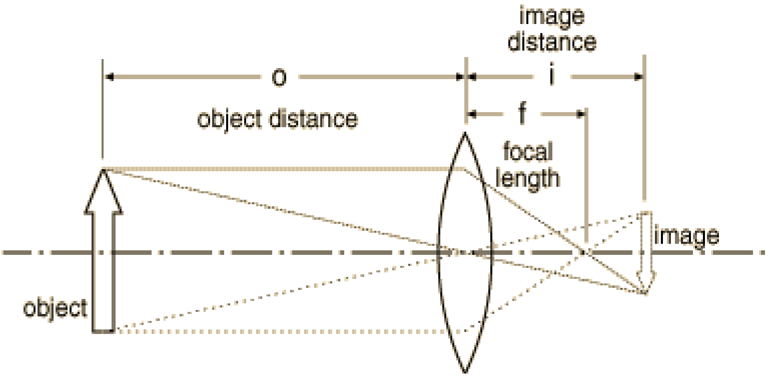


***Figure 9: Distance used to describe formation of Image with convex lens** /18/*

The mathematic correlation between the distance of the lens to the projected image (i), the focal length of the lens (f), and the distance from the lens to the object (o). The equation bellow illustrates the relationship /18/:

$$\frac{1}{f} = \frac{1}{i} + \frac{1}{o} \; where \; f = focal \; length, i = image \; distance, o = obeject \; distance \qquad (1)$$

It is accepted to use **C**enter **O**f Projection (COP) model in [figure]. The rays converge at the origin of the camera frame (COP) and put **P**roject **P**lane, which is the image plane in front of the COP. The main reason for doing this is that it helps in avoiding geometrically equivalent and inverter image. Additionally, to have the right-handed coordinates, the camera is made to face down the negative **z** axis.

The projection perspective of the real world to the projection plane:

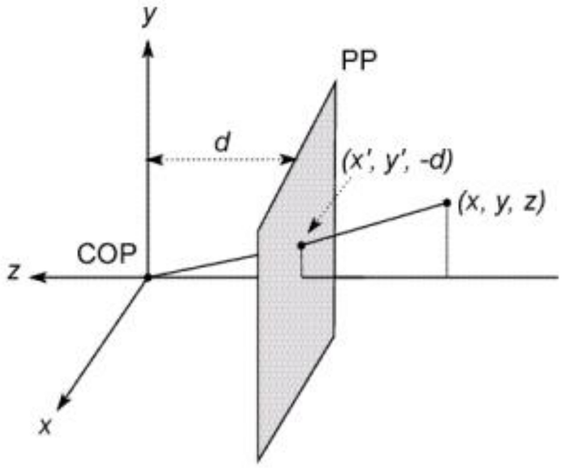


***Figure 10: Pinhole Camera model for COP** /22/*

The equations of the projection:

At this stage, one computes intersection with plane PP of ray obtained from x, y, z coordinate to COP. Derivation of this approach takes similar triangle as shown below:

$$(x, y, z) \rightarrow (-d\frac{x}{z}, -d\frac{y}{z}, -d) \quad \text{Equation 2}$$

Derive the projection coordinate on image by ignore the last coordinate:

$$(x, y, z) \rightarrow (-d\frac{x}{z}, -d\frac{y}{z}) \quad \text{Equation 3}$$

The coordinate's points in the image plane on the homogeneous form are:

The question is to determine whether it is a linear projection (however, to have a linear projection, there has to be a division by z, thus it is nonlinear).

$$(x, y, z) \rightarrow (-d\frac{x}{z}, -d\frac{y}{z}, -d) \quad \text{Equation 4}$$

The trick is adding one more coordinate: $(x, y) \;=>\; \begin{bmatrix} x \\ y \\ z \end{bmatrix}$ Homogeneous image coordinate

$$(x, y, z) => \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$ Homogeneous scene coordinates

Converting from homogeneous coordinates:

$$\begin{bmatrix} \mathrm{x} \\ \mathrm{y} \\ \mathrm{w} \end{bmatrix} => ({}^{\mathrm{x}}/_{\mathrm{w}}, {}^{\mathrm{y}}/_{\mathrm{w}}) \qquad \begin{bmatrix} \mathrm{x} \\ \mathrm{y} \\ \mathrm{z} \\ \mathrm{w} \end{bmatrix} => ({}^{\mathrm{x}}/_{\mathrm{w}}, {}^{\mathrm{y}}/_{\mathrm{w}}, {}^{\mathrm{z}}/_{\mathrm{w}})$$

Equation 5

Homogeneous coordinate in Geometric intuition:

It provides a leeway of extending -d space to (N+d)-d space. Conceptualising this approach to the case, a point in the 2D image is regarded as a ray in 3D projective space. Each point (x,y) on the image plane is denoted by the **(sx, sy, s)** to mean that all point on the ray remain equivalent (that is, **(x, y, 1 ≡ (sx, sy, s))**:

It can be converted back to two dimensions by dividing with the last coordinate **(sx, sy, s)/s → (x, y)**

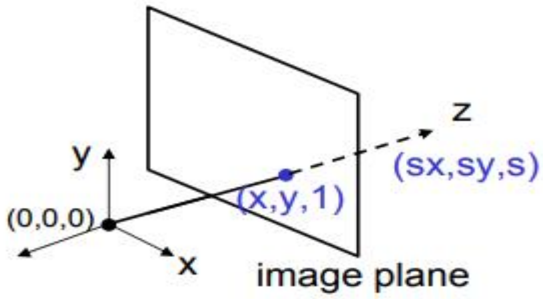


Figure 11: Image Plane in Geometric Intuition */22/*

- The modelling projection:

The projection is referring to as matrix multiply with the use of homogeneous coordinates:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -{}^{1}/_{\mathrm{d}} & 0 \end{bmatrix} \begin{bmatrix} \mathrm{x} \\ \mathrm{y} \\ \mathrm{z} \\ 1 \end{bmatrix} = \begin{bmatrix} \mathrm{x} \\ \mathrm{y} \\ -{}^{\mathrm{z}}/_{\mathrm{d}} \end{bmatrix} => \left(-\mathrm{d}\frac{\mathrm{x}}{\mathrm{z}}, -\mathrm{d}\frac{\mathrm{y}}{\mathrm{z}}\right)$$

Equation 6

To get coords image, divide by the third coords and throw it out

This process is called **perspective projection** and the matrix is called projection matrix.

Perspective projection: here one needs to know how the scaling matrix projection converts the transformation. This can be express in the equation:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1/\mathrm{d} & 0 \end{bmatrix} \begin{bmatrix} \mathrm{x} \\ \mathrm{y} \\ \mathrm{z} \\ 1 \end{bmatrix} = \begin{bmatrix} \mathrm{x} \\ \mathrm{y} \\ -\mathrm{z}/\mathrm{d} \end{bmatrix} => \left(-\mathrm{d}\frac{\mathrm{x}}{\mathrm{z}}, -\mathrm{d}\frac{\mathrm{y}}{\mathrm{z}}\right) \qquad \text{Equation 7}$$

Scaling by **C**:

$$\begin{bmatrix} c & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -c/d & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} cx \\ cy \\ -cz/d \end{bmatrix} \quad => \left(-d\frac{x}{z}, -d\frac{y}{z}\right) \qquad \text{Equation 8}$$

Therefore, if (x, y, z) scaled by C, we still get same result. This mean:

In the image, a bigger object at a distance (scaled x, y, z) can have same size as smaller object that is near.

Projection models Simplification:

Here we look at it in two different ways: Weak Perspective and Orthographic

- Weak perspective Transformation:

Let’s recall the (*Equation 7*)

Let’s say the relative depths of points on object are far smaller than average distance z_{av} to centre of projection (COP). Then, at every point on the object,

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -\frac{z_{av}}{d} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ -\frac{z_{av}}{d} \end{bmatrix} \quad => (cx, \qquad cy) \qquad \text{Equation 9}$$

Where c = $-\frac{d}{z_{av}}$

In this case, there was reduction of the projection to uniform scaling of all object point axes.

- Orthographic Projection:

Let’s assume that d → ∞ in perspective projection model:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1/d & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ -z/d \end{bmatrix} \quad => \left(-d\frac{x}{z}, -d\frac{y}{z}\right)$$

Equation 10

Where, z → -∞ then we have $-d/z$ → 1 thus, (x, y, z) → (x, y)

This process is known as parallel/orthographic projection

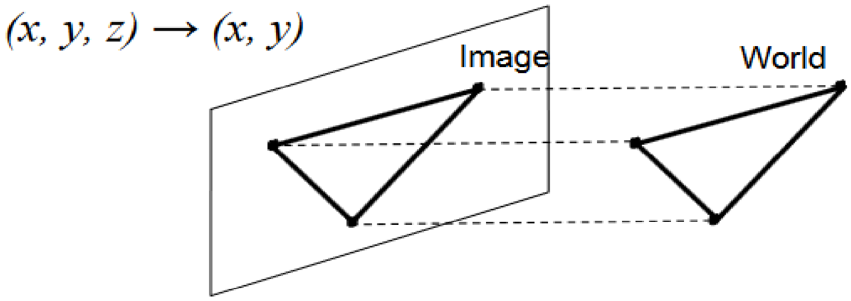


Figure 12: Parallel View in world and Image */22/*

Taking into an account the orthographic projection matrix in homogeneous coordinate

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad => (x, y)$$

Equation 11

Let us revisit Weak Perspective Projection:

The previous theory explained that:

Weak perspective transformation *(x, y, z)* → *(cx, cy)* = orthographic projection *(x, y, z* → *(x, y)*

And then uniform scaling by a factor c = $-\frac{d}{z_{av}}$

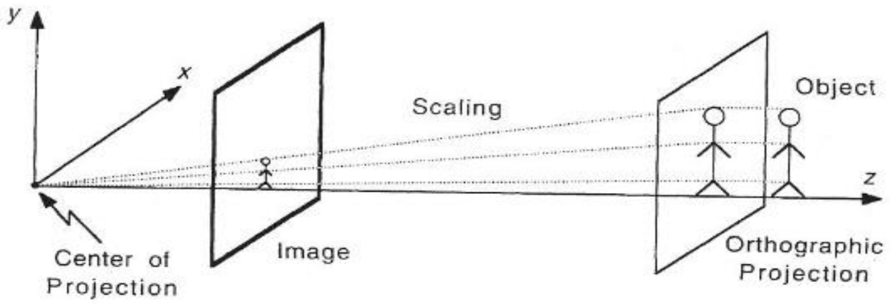


Figure 13: Weak Perspective Projection */22/*

- Radial distortion of the image. Causes:
- This can be caused by the imperfect lenses or,
- The difference is discovered from image surrounding (rays going through the edge of the lens).

Modeling Radial Distortion

This is typically modes as:

$$x = x_d\,(1 + k_1 r^2 + k_2 r^4) \qquad \text{Equation 12}$$

$$y = y_d\,(1 + k_1 r^2 + k_2 r^4) \qquad \text{Equation 13}$$

$$where\; r^2 = x_d^2 + y_d^2$$

- (x_d, y_d) mean the coordinates of distorted points wrt image center, (x, y) mean the coordinate of the corrected points
- The radial displacement of image points - increases with distance from center is known as distortion
- While $k_1$and k_2 are the parameters to be estimated. Typically, k_1 accounts for 90% of distortion.

Combining all together: Camera parameters: in this case, the link axes of points in 3D external space with their axis in the image. Thus, we define perspective project in terms of camera reference frame as shown in the figure [14]:

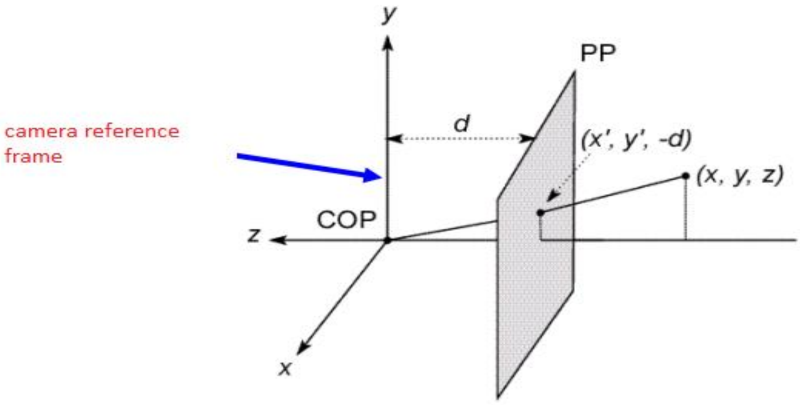


Figure 14: Perspective Projection in terms of camera reference frame */22/*

It is possible to find the orientation and location of the camera reference frame in correlation to a known world reference frame, commonly known as extrinsic parameters.

Extrinsic parameters of the camera

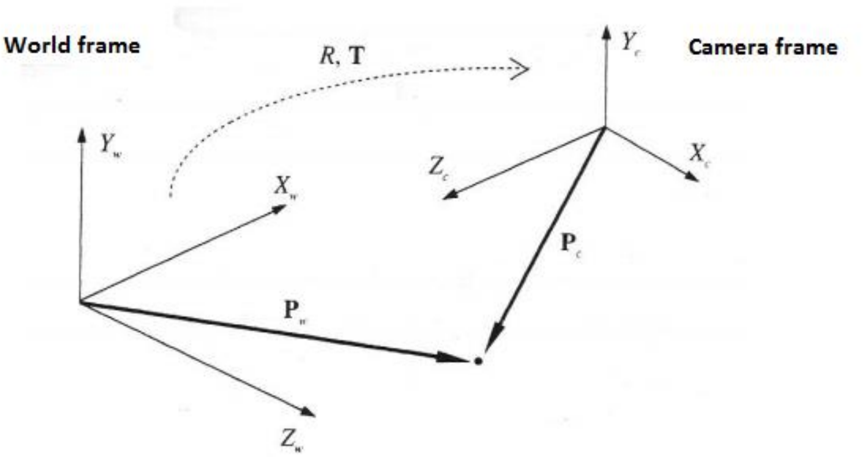


Figure 15: Parameter which describe the projection between camera and world frames */22/*

Extrinsic parameters mean parameters that describe the projection between the camera and world frames. The point position in respect to the camera in homogeneous coordinate is:

$$P_c = R(P_w - T) \qquad \text{Equation 14}$$

Where: T remains as the 3D translation vector describing the relative displacement of the origins of the two reference frames. On the other hand, R is regarded as rotation matrix with the ability to align the axes of the two frames onto each other and P_w is the transformation or projection of point in world frame to the point P_c in the camera frame.

In general, we can say that:

- Intrinsic camera parameters refers to those which describes the geometric digital as well as optical features of the camera.
- Transformation or perspective parameter: focal length **d**
- Distortion resulting from optics: this refers to parameters of radial distortion which is k_1, k_2
- Transformation from camera frame to the coordinates of pixel units includes:
  - The coordinates (x_{im}, y_{im}) of the point of the image in pixel units with regard to axes (x, y) of same point in camera reference frame through:
    x = - $(x_{im}, o_x)\ s_x$, y = - $(y_{im}, o_y)\ s_y$
    Where (o_x, o_y) is the image center ands_x, s_y as the size of the pixel. (However, based on the equation above the, signs are denoted by the opposite of the orientations of x/y coordinates in the camera and image ref. frame).
- The estimation of the intrinsic and extrinsic is known as camera calibration.

4.3 Camera Calibration

This is needed if one needs to get an accurate three-dimensional (3D) positioning from two-dimensional scene. The camera used determines the understanding of the calibration. In addition, for each camera its uses several or multiple images of a checkboard pattern to carried out the calibration in [16]

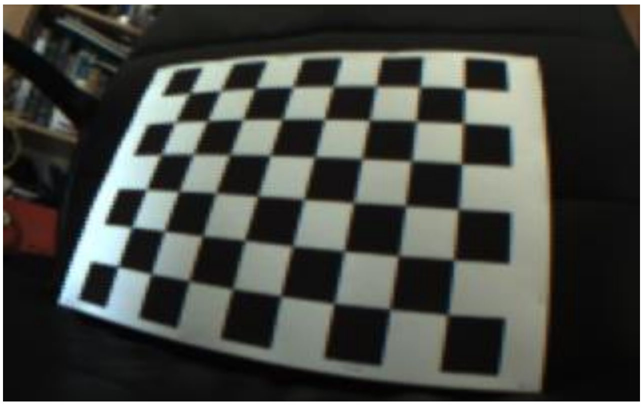


Figure 16: A checkboard used for calibration, the image is taken by Pixy camera that enable clear view of camera distortion.

Therefore, it is required to calculate the camera's intrinsic parameters. The intrinsic depend on the features of the cameras, and the extrinsic values that depend on their positioning, after the images needed have been taken in each camera.

In intrinsic, there are five parameters which are: focal length (f), the size of the pixel in x, y direction (s_x and s_y) and the image center (o_x and o_y). Sometime in particular when the pixels are square. This result to $s_x = s_y = s$ which mean the number of intrinsic parameter decrease to 4 which is the image center (o_x and o_y) and the focal length /22/

Here are the equations relating to image projection: from the world coordinates to pixel coordinates:

Plugging P_c = R (P_w – T), x = - (x_{im}, o_x) s_x , y = - (y_{im}, o_y) s_y into perspective projection equation, then, we get

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \mathrm{M_{int}M_{ext}} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} \text{where}(\mathrm{X_{im}}, \mathrm{Y_{im}}) = ({}^{x}/_{z}, {}^{y}/_{z}) \qquad \text{Equation 15}$$

Note: M_{int} is the camera to image reference frame while M_{ext} is the world to camera reference frame.

Next, we have:

$$M_{int} = \begin{bmatrix} d/s_x & 0 & o_x \\ 0 & d/s_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \quad M_{int} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & -R_1T \\ r_{21} & r_{22} & r_{23} & -R_2T \\ r_{31} & r_{32} & r_{33} & -R_3T \end{bmatrix}$$

Equation 16

Note: (r_{ij} represent the elements of rotation matrix R; R_i is its i-th row)

Consequently, estimations of all parameters are made by procedure or camera calibration rules.

There can be an error in the focal length, which can be rectified if the lens is focused at infinity. In a situation, whereby the lens is being replaced, and aperture of focus adjustment, then, it required to recalculate the intrinsic parameter. Why? Because there can be changes. Thus, the common ways of calibration method are based on sets of world points, in this case their corresponding coordinates in the image plane, plus their relative coordinates are known.

As reported by Zhengyou Zhang /48/, the calibration techniques roughly can be classified into two categories: self-calibration and photogrammetric calibration.

1. Photogrammetric techniques. This approach begins with known where one calibrates by observing an object whose 3D geometry is succinctly known.
2. Self -calibration: unlike photogrammetric techniques, self-calibration does not use calibration object. However, this category opts for moving a camera in a static scene. While this approach/technique is easy, it has a number of parameters for estimation.

Zhang's calibration techniques /48/, are the techniques used in this project for camera calibration. This technique is widely used, also it lined between the self-calibration and photogrammetric calibration. But photogrammetric has one main diversity, that is, it uses 2D pattern instead of 3D object.

4.4 Chapter 4: Conclusion

This chapter focuses mainly on machine vision, also computer vision was introduced. An overview was also given to explain some areas where these technologies can be applied. In addition, it explained the camera model, and also described the cameras calibration.

5. SYSTEM ARCHITECTURE

The main objectives of this project are to develop a system that track the 3D coordinates of an object and position, extract the data in real time and send that via socket to the industrial robot for other processing also the use of low cost hardware and software are essential.

This project is motivated by Mika Billing, Master of Science (Technology), a senior lecturer, University of Applied Science, School of Technology, Mechanical Engineering department, the aim is to replace the expensive Cognex Camera on ABB industrial camera with a low-cost system for the same task.

This system uses a low-cost solution to carry out tracking system, it has an infrared teaching system for correction and part positioning. Below are the main components used in this project:

- Two Pixy Cameras;
- Track It Yourself (including modifications) to carry out the tracking object;
- Raspberry Pi computer

Figure 17 explains the intended work flow of the systems. In the working station representing the IRB 1200 ABB industrial robot, the user placed a color object in front of the camera to teach the task to do. The pixy camera tracks the position of the color object and send the data to the computer. Then, the modified version of TIY software developed in this project, which is responsible for the changing of 2D data from two cameras into 3D tracking position is the execute by the raspberry pi computer. The 3D data result will then be transmitted from the system. This is meant to be an easy and efficient method of industrial robot vision system that can be combined in the clients' services.

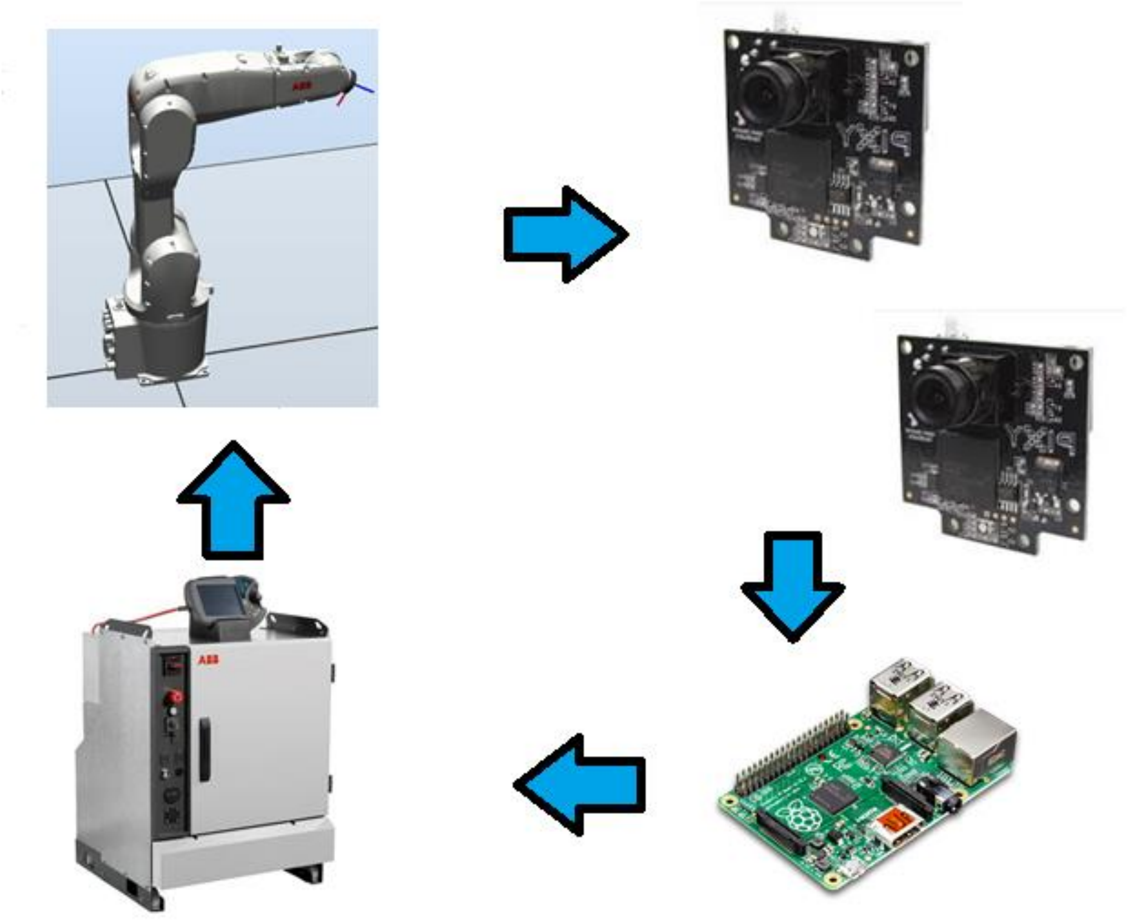


Figure 17: Industrial Vision System main components

5.1 Positioning Tracking System Cameras

RobotStudio, a software built on the ABB virtual controller that enables robot programming to be done on a PC in the offline without production being shut down, is used. The purpose of selecting this software was because it enables the simulation of artificial vision systems in robotic cells. The use of this software is to locate the appropriate position to place the tracking system cameras.

5.2 Raspberry Pi Overview

A Raspberry Pi is a credit card sized computer, developed by Pi foundation from the university of Cambridge in the United Kingdom (UK), which, alongside, including small processing power compare to a normal computer, it includes features that make it useful for little electronics projects and is cheap, it is available in most online shops.

Considering the accessibility, having the largest community of enthusiasts, and my little experience with other microcontrollers available in the market, Raspberry Pi was chosen as the most suitable computer for this project.

The first Raspberry pi computer is based on the System on a Chip (SoC) Broadcom BCM2835, and consists of ARM1176JZF-S processor, with a clock speed of 700MHZ, 526 MB of RAM, which later upgraded to 1GB of RAM in Raspberry Pi 3, which is the model type used in this project and later explain in sequel, also one VideoCore IV Graphics Processor Unit (CPU) /49/.

Moreover, the minicomputer was designed for the Linux operating system, and now, many Linux distributions have a version optimized for the Raspberry Pi. In addition, it provide some programming tools for Ruby, Python, C++, C, Perl and Java programming languages /36/.

5.2.1 Processor

The processor includes ARM1176JZF-S, it has only one core operating system which runs at 700MHz, there is also a possibility of increasing the speed up to 1000 MHz without losing the warranty if using configuration system setting – Rasbian (the used operating system in this project). One can make an overlocking in Raspberry by running command *sudo rasp-config* and choose the desire speed. You do not have to use heat sink in the SoC as the temperature is taken care of within desired levels as overlocking is disabled if detected high temperature in the processor /36/.

5.2.2 Peripherals

The B Model Raspberry Pi has a 10/100 Mbps Ethernet port with two USB 2.0 ports, but without wireless communications except one add it to it. The figure [18] shows the model B of Raspberry Pi layout.

Considering the B Model Video outputs support it supports most common interface protocols like Full High-Definition (HD) or Videos Graphic Array, inclusive maximum resolution of 1920 x 1200. It also has a High-Definition Multimedia interface (HDMI) video output connector. In addition, it also has 3.5 mm for audio output; the board contains a 15-pin video input connector for camera connection.

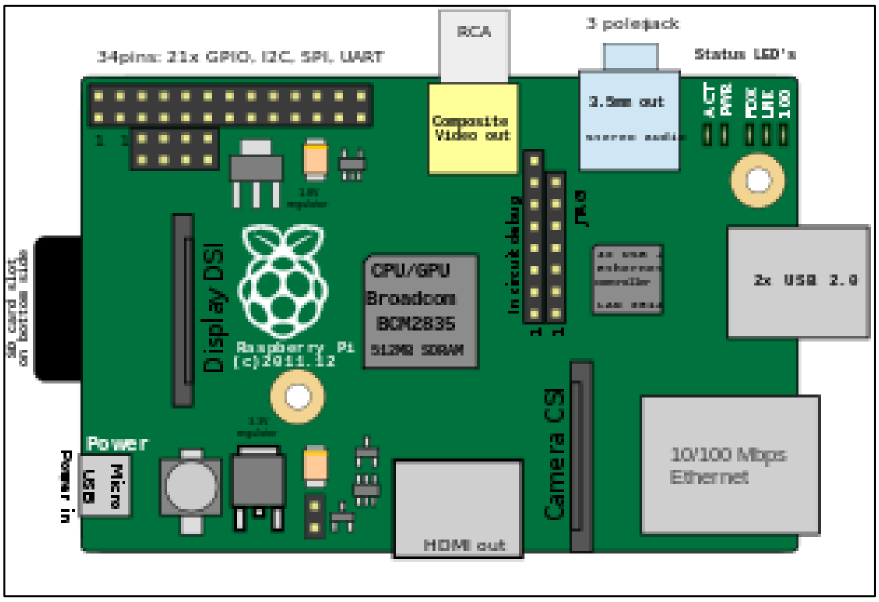


Figure 18:Model B board layout /12/

General Purpose Input Output (GPIO): The board has no real-time clock but one can synchronize it through the network or it can be added using the GPIO pins. In addition, the GPIO pin feature on the board enable the analog inputs and outputs also 3.3 V and 5 V power supply pins and the board is with 26 pins connector grouped in 2 x 13 form. Besides, in the GPIO pins there is I2C, SPI and UART, communication interfaces.

SD Card: Nevertheless, Rasberry do not have an onboard storage system. As such, it is easy to install the operating system on the SD Card. To achieve this, the SD Card is inserted on the SD card slot in the raspberry Pi. The operating system is installed with the use of a card reader on any computer. The use of SD card on raspberry Pi has its own advantages which are: that it is easy and quick to change the card that has a different system on the raspberry Pi and the cards are cheap and easy to buy in all electronics shop. Thus, the disadvantage is that the SD card is slower than a flash memory.

Power Source: The Pi computer consumes 700mA or 3W or power, according to the manufacturer /36/. The power can be supplied through GPIO pins related to 5 V and GND (Ground) also MicroUSB charger or any good 5 V mobile phone charger will do the work of powering the Pi computer.

5.2.3 Operating Systems

The Pi computer primarily supports Linux kernel based operating systems. Because ARM11 based on version 6, which is an outdated version and thus not supported by different versions of Linux. A good example of such versions includes Ubuntu. However, NOOBS which is the install manager for raspberry Pi, its operating systems are OenELEC, Raspbmc, Archlinux ARM, RISC OS (the operating system of the primary ARM-based computer) and Raspbian (recommended)

In this project, Raspbian (Jessie) was used. this is the recommended version by the maker of Raspberry pi computers and being the one that come with more available support. Raspbian is supported independently of the foundation, based on Debian ARM hard-float which is design for this minicomputer hardware. Jessie is an updated new stable version of Debian. To use Raspbian it is required a minimum memory space of 4 GB SD card.

5.2.4 Raspberry Pi 2 Overview

This computer was released in 2015 by the same developers. It provides more processing power compared to the previous version. Thus, considering the increase in the refresh rate in the tracking system with a greater processing power, Raspberry Pi 2 chosen instead of Model B for better results in case the object to be detected have a faster movement. Both Model have same cost (about ± 35$).

In addition, both Model B and Raspberry Pi 2 has same GPU and their power consumption are equal, also the peripheral of all the models are identical, but the 26 GPIO pins in Model B increases to 40 in and the 2 USB ports increases to 4 in Raspberry Pi 2.

In the main processor the Raspberry Pi 2 has roughly six times processing power than Model B, that is the Pi 2 consists of a quad-core processor ARM Cortex A7 900 MHz (BCM2836), with a 1GB RAM, while Model B has an ARM Cortex A6 processor 700 MHz (BCM2835), and a 512 MB RAM /35/.

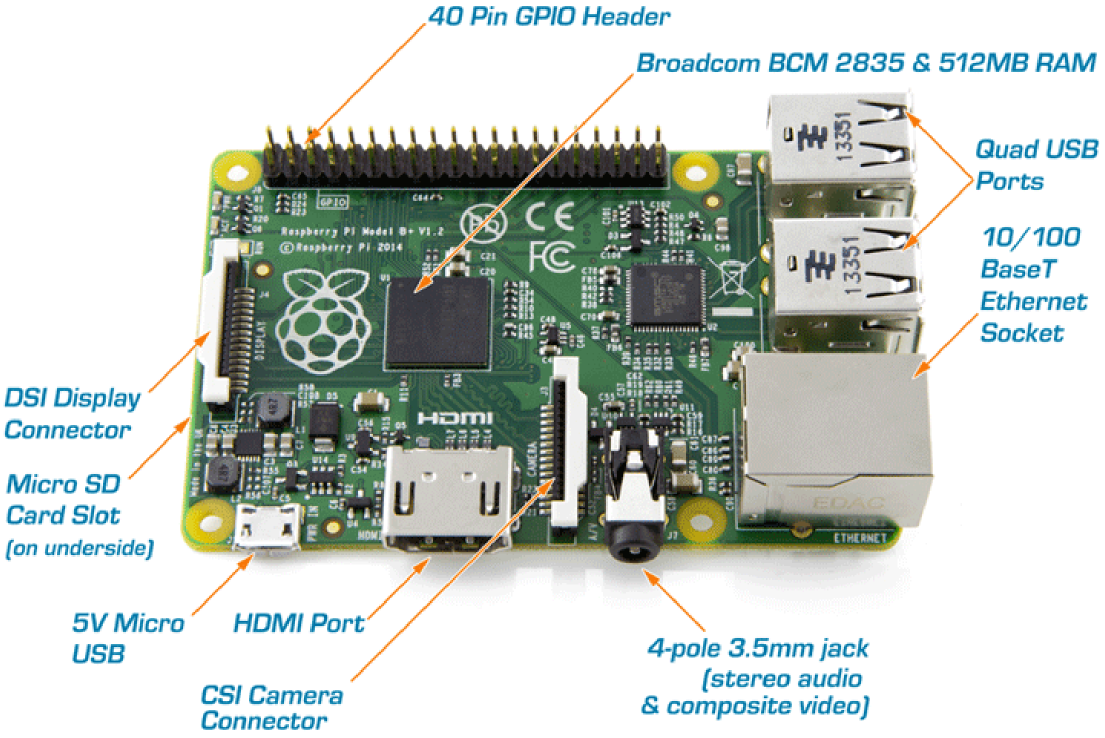

40 Pin GPIO Header
Broadcom BCM 2835 & 512MB RAM
Quad USB Ports
10/100 BaseT Ethernet Socket
DSI Display Connector
Micro SD Card Slot (on underside)
5V Micro USB
HDMI Port
CSI Camera Connector
4-pole 3.5mm jack (stereo audio & composite video)

Figure 19: Raspberry Pi 2 Model B Layout /16/

5.3 CMUcam5 Pixy

The CMUcam5 Pixy are the used cameras in this project. Pixy is a collaboration between Chamed labs and the Carnegie Mellon Robotics Institute and Pixy are trends from CMUcams cameras and they came onto the shops in March 2014. As a result of its low cost and its ease of use, it is commonly used in small projects related to robotics. The principal features include /28/:

- It can learns and detect the object that is being thought
- Its outputs 50 times per second data from the objects detected
- It comes with libraries for universal controllers, like the Arduino or the Raspberry Pi
- It supports C/C++ and Python programming languages.
- Multiple communication interfaces like: Serial Peripheral Interface (SPI), Inter-Integrated Circuit (I2C), Univeral Serial Bus (USB), Universal Asynchronous Receiver/Transmitter (UART), or digital/analog outputs
- Configuration tools run on Linux, MacOS and Windows

- All software/firmware is open-source GNU licensed

The Pixy camera is unique in such a way that it enables tracking an object using its own hardware and firmware, carrying out all the significant processing, and outputted the object position. For vision systems with low processing power since capturing of images and processing task, also the detection of objects from the main external controller, Pixy with integrated are ideal for this kind of systems. In addition, aside from the image sensors in the camera, its include also a processor to carried out tracking and other features.

The Pixy camera uses filtering algorithms for detection of an object. The filtering algorithms are well-known algorithms because they are efficient and fast. The camera does the color calculation and separate each RGB pixel by pixel from the image captured and apply these parameters to filter the image content. The color hue is good approach because color remains relatively unchanged with change in brightness. Thus, it means that if the color objects you are trying to detect are not of good specific, the color will not work.

Pixy remembers up to seven different color signatures, which mean, with different color signatures, pixy can detect seven different objects. In case of detecting more than 7 different color object, the use of color code can make this possible.

Multiple objects can be detected by Pixy at a time. It adopts a connect filters algorithms in the determination of areas where an object can start and another end. Thereafter, Pixy takes the role of compiling the sizes as well as locations for every object and transfer the results via the selected communication interface.

In addition, the camera has high resolution, it processed an image in a resolution of 640 x 400 per frame in every 50 times per second (20 milliseconds) and the result are givens with a resolution of 320 x 200 per frame or pixels. Processing constraints in the embedded controller, leads to downscaling of the resolution. This resolution may be too small for some projects, but it is acceptable even for detecting some moving objects.

A Pixy camera is a special device because it enables one to physically teach it what one wants to sense, which is of great advantage. In the camera, there is a button available for this. Place the object to be detected in front of the camera and press the button [figure 20] down until the LED light turns red, after which upon releasing the button, the Pixy will generate a statistical model of the color that contains the object and are stored in the flash.

Consequently, the statistical model is used to located an objects with similar color signatures in a given frame.

Lastly, an alternative method to teach the object is to use the PixyMon software. This software also allows recording what the camera captures either as raw or processed video. In addition, the software enables the configuration of the cameras: like, the color signatures and communication interface. The standard mini USB cable is used by PixyMon to communicates with the cameras, more so it allows receiving the video signals at the same time as the interface is sending data to the external controller. PixyMon is useful for debugging purposes.

Principal features of the Pixy camera include/28/:

- processor: NXP LPC4330, 204 MHz, dual core
- image sensor: Omnivision OV9715, 1/4", 1280 × 800
- Lens field-of-view: 75 degrees horizontal, 47 degrees vertical
- Lens type: standard M12 (several different types available)
- Power consumption: 140 mA typical
- Power input: USB input (5 V) or unregulated input (6 V to 10 V)
- Random Access Memory (RAM): 264 Kbyte
- Flash Memory: 1M bytes
- Available data outputs: UART serial, SPI, I2C, USB, digital, analog
- Dimensions: 2.1" × 2.0" × 1.4
- Weight: 27 grams.

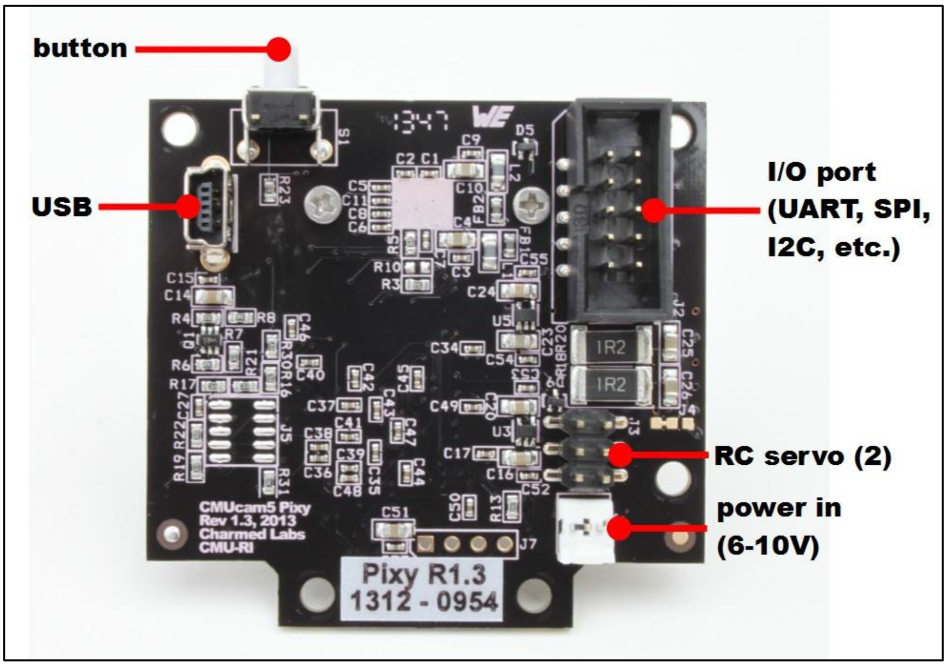


Figure 20: CMUcam5 Pixy camera layout /28/

5.4 Open Source Computer Vision Library (OpenCV)

OpenCV is an open source software, specially made for computer vision, it is written in C++, C, Python and java interfaces and runs on Windows, Linux, Mac OS, iOS and Android. Initially, OpenCV software was developed by Intel and officially published in 1999, which ready for use for the public in that year. It has Python, Ruby, Java, Matlab, C, and C++ interfaces. The software was designed for computational efficiency, to be used in real-time application.

In computer vision, the involvement of OpenCV is critical to the development of artificial vision by enabling license-free usage to many people. In addition, it is also present for all interested people in the field of computer vision for developing their work. OpenCV have been used in several applications, research and products. Application such as image alignment, objects analysis, members of several satellite image to produce map, autonomous vehicles, reduction of noise in medical image diagnostic, camera calibration systems and security, military application and intrusion detection.

OpenCV's goal is to create an efficient way to implement computer vision systems that has above 500 functions that meet the requirements of many of the common applications like security, medical imaging, user interfaces, robotics, industrial inspection product, and stereo vision. OpenCV provides libraries for both machine vision and computer vision because their features are common /25/.

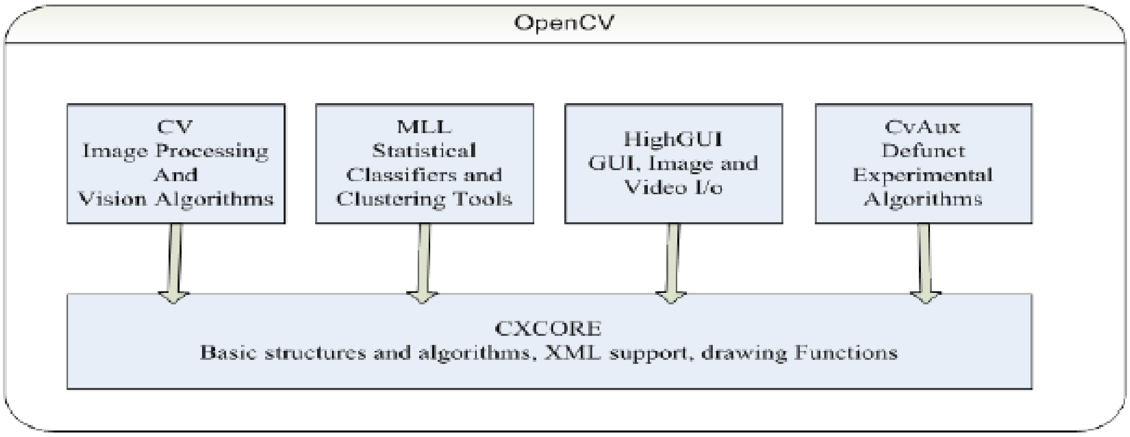


Figure 21: Basic Structure of OpenCV */33/*

OpenCV has five basis structures shown in figure [21]:

- CV component: it includes processing and high-level computer vision
- MLL component: it includes the machine learning library which again, has lots of statistical classifiers as well as clustering tools
- GUI component: it includes input/output routines, uploading of images and videos, and functions for data storage
- CXCore component: this contains basic data structures and the core of the system.
- CvAux component: it includes extinct areas of OpenCV plus experimental algorithms /25/.

OpenCV version from 2.x forward, Application Programming Interface (API), is essentially a C++ API which is an interest interface used for development of this project. This API was used to handle capturing of image from cameras. In addition, OpenCV has a modular structure, that is, it comes with a package which includes many static libraries. Some of the modules are /19/:

- Core: This is a module that explains the basic structures such as multi-dimensional arrays. Mat acts as basic functions that can be used for other modules.

- Imgproc: module for image processing which contains filtering such as among others, linear and non-linear image and geometrical image transformations.
- Video: a video analysis module with the ability for background subtraction, motion estimation and objects tracking algorithms.
- Calib3d: as the name suggest, this is a module providing basic multi-view geometries algorithms. Others included are object pose estimation, stereo and vision calibration, element of 3D transformation and stereo matching algorithms.
- Features2d: it acts as salient descriptors, feature detectors and descriptor matchers.
- Objdetect: it serves that purpose of detecting objects and cases of predefine classes such as eyes, faces among others.

5.5 Boost

The boost collection of libraries is based on the C++ programming language. Image processing, multithreading, random number generation and linear algebra are some of the functionalities provided for by Boost. Boost contains more than 100 individual libraries which allow anyone to use, modify and distributes the library for free. In addition, are platform independent and support most common compiler such as Linux and Window. The aim of boost community is to develop and gather high quality libraries that complement the standard library /42/.

This collection of libraries was uses in this project because of Track It Yourself (TIY) dependence in Boost; include also some useful features needed in the development of this project. The boost libraries are used for instance, timestamp the captured data from the cameras.

5.6 Chapter 5: Conclusion

In this chapter it was introduced first the overview of the developing system, such as how the system is intending to interact, the use of ABB RobotStudio and the require hardware components for developing the system. In addition, it tells about the type of operating system installed on the used Raspberry Pi computer. Also, the functionality of OpenCV and Boost library in the developing system.

6. IMPLEMENTATION

This chapter explains how this project was developed. It shows the design circuit that is the electrical connections, the development of the software and incorporation of all components required to meet the goal of the project. The figure [26] shows the data flow on how the project work was implemented also, the Annex [C] shows the electrical circuit design in addition with the cameras, the computer and the level converter.

6.1 Raspberry Pi Communication and Power Connection

6.1.1 Power Connection

Typically, Raspberry Pi has 5 V DC voltage supply with micro USB power connector, the power connector is usually used in today mobile phones and tablets, the board contain excessive power consumption and protection against polarity. The used Model in this project, the charger gives 700mA and 1500mA, and the computer consumed about 500mA, it has the capacity of producing 500mA via the USB and 50mA via GPIO /36/. The cameras used, each of it needs 140mA supply from the raspberry Pi power source. Therefore, the total power needed to power up the complete system 780mA, (500mA+140mA+140mA)

Moreover, the computer can be powered by the GPIO pins, though the pins have no protection. In addition to the USB, this is another way to power the Raspberry Pi computer through one of the four USB ports; also, there is a possibility of using a USB hub for this purpose. Nevertheless, with USB 2.0 standard, it should be considered that USB port can only supply 500mA, commonly, this is the power provided by the hub from each of the ports /36/.

6.1.2 Communication

The model B board contains difference type communication interface, it comes with four USB ports, 40-pins 2.54 mm (100) expansion header: 2x20 strip, which can be configured to support I2C, UART and SPI protocols. It has also one internet port 10/100 with RJ45 connector.

The expansion header has a connector label as Pin1 as shown in figure 23 it has 40 pins and the space between those pins is about 2.54 mm, and set as 2 x 20 strip. In addition, the GPIO also provides +3.3 V, +5v and ground (GND) pins. From the front view of its layout, the P1 connector is the first column of the bottom line. But in this project the communication between cameras and Raspberry Pi computer is made via I2C interface and power are supplied to the cameras by the raspberry Pi computer, the needful pins are: 1, 2, 3, 4, 5, and 6 pins.

- Pin 1: +3.3 V DC output voltage,
- Pin 2: +5 V DC voltage output,
- Pin 3: I2C, SDA (Serial DAta line) with pull-up resistance of 1.8 kΩ,
- Pin 4: +5 V DC voltage output,
- Pin 5: I2C, SC L(Serial Clock Line) with pull-up resistance of 1.8 kΩ, and
- Pin 6: Ground (GND).

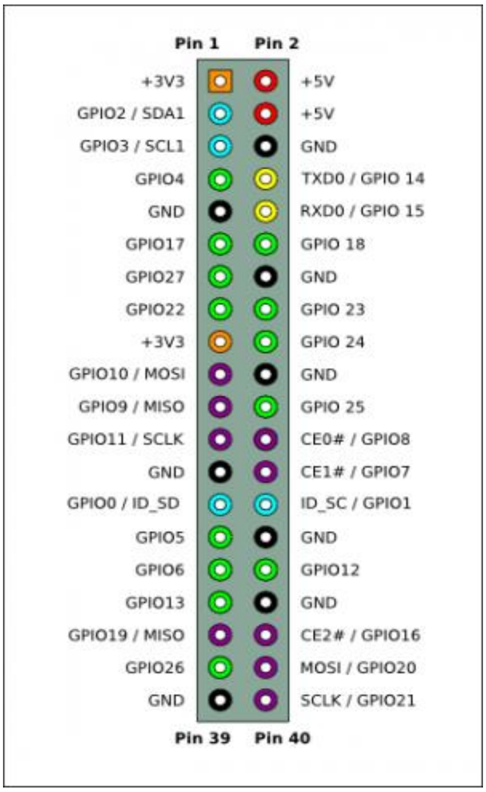


Figure 22: Raspberry Pi GPIO output Pins */37/*

6.1 Pixy Communication and Power Connection

6.2.1 Power Connection

A Pixy camera can be powered in three different ways:

i. It can be powered through USB connector (5V regulated),
ii. It can be powered via I/O connector (5V regulated), and
iii. It can also be supply through power connector (ranges between 6 V to 10 V unregulated).

Simultaneously, power can be supplied to the camera through I/O and USB connector, without any issues. Typically, the camera is connected to the raspberry Pi through USB and the power supplied by the computer is regulated with 5 V output.

The I/O connector 5 V; this uses pin 2 as the Input/Output of Direct Current +5 V and Pins 6, 8 and 10 as the ground. One should be very careful with the power supply wiring connection from the input or output connector, because it has no polarity reversal protection (insert figure from below)

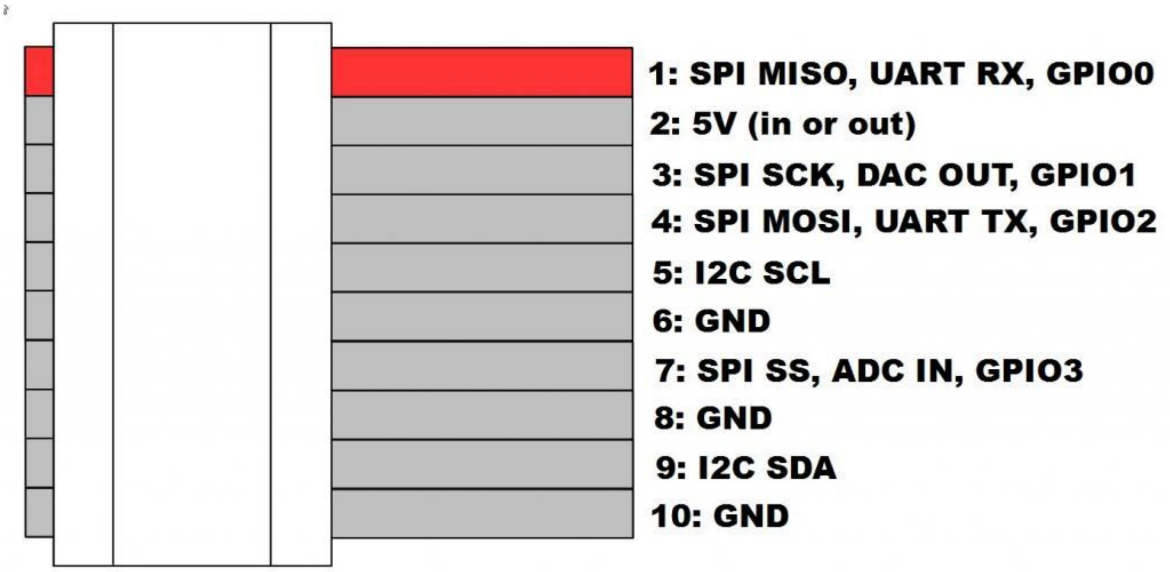

1: SPI MISO, UART RX, GPIO0
2: 5V (in or out)
3: SPI SCK, DAC OUT, GPIO1
4: SPI MOSI, UART TX, GPIO2
5: I2C SCL
6: GND
7: SPI SS, ADC IN, GPIO3
8: GND
9: I2C SDA
10: GND

Figure 23: Pixy; Input and Output Connector */34/*

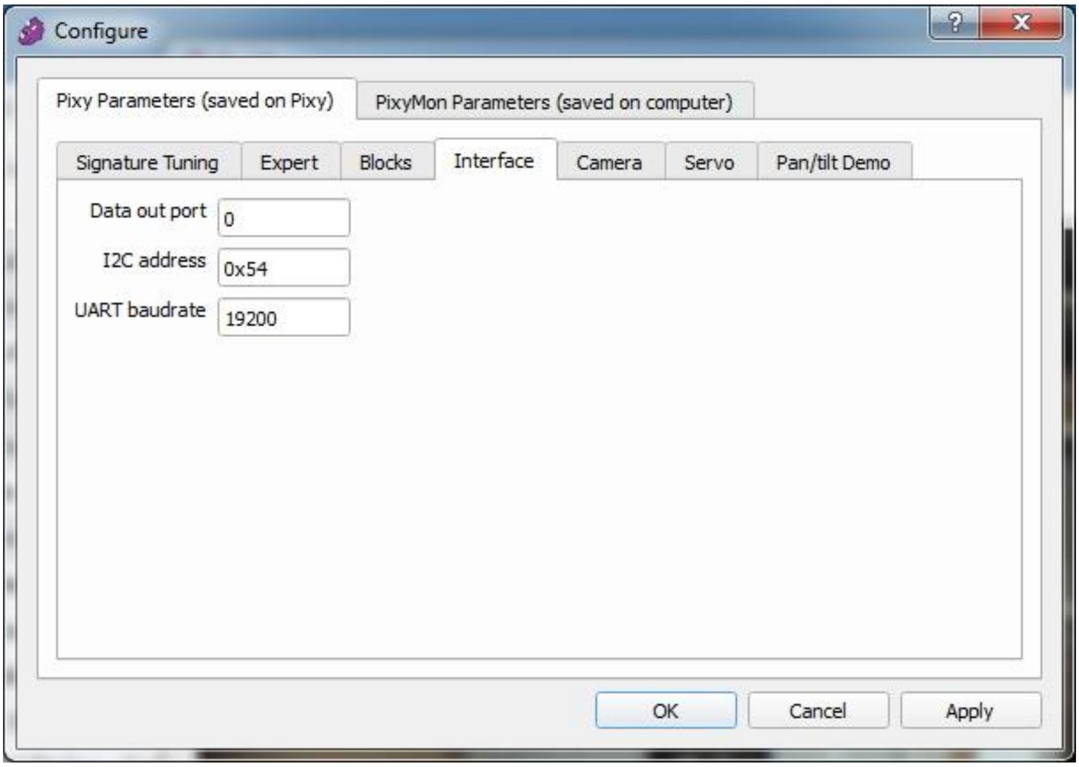
Lastly, the power connector; this system provides more power supply to the camera with a threshold of 1.5 A, also is capable of supply the pan/tit servomotors of the camera and there will still be enough current left to power a microcontroller from the Input or Output (I/O) connector. This offers protection against reverse polarity /34//49/.

6.2.2 Communication

The communication between the Pixy camera and other devices can be achieved in five different ways, either from I/O connector supporting interface like I2C, UART and SPI, or using the USB ports. In addition, digital and analog outputs can also be used by Pixy to communicate outside. But SPI is the fastest among all the three interfaces provided by I/O connector, next is I2C interface. But of all interfaces, USB is the fastest, also the easiest and the smoothly means of transferring images between the used micro-controller computer and Pixy. The slowest among all the interfaces is UART, this interface should only be used if the rest mentioned interfaces are not possible. Hence, if the controller used is not supported by any of the interfaces, it is recommended to use analog and digital output alternatively.

6.2.3 Setting the Interface

The PixyMon software can be used to configure the interface of your pixy camera, depending on the type of interface you want to use. the example figure 25, shows the I2C set interface. The “Data out port” parameter in the *interface* tab determines the type of interface you are using, the I2C set the interface for the camera while UART set the communication interface /32/.



Data out port: 0

I2C address: 0x54

UART baudrate: 19200

*Figure 24: **Parameters configuration of Pixy Via PixyMon*** /32/

When a mouse is being pointed at the Data out port text which shows a help string that explains the correspond value to the type of port. In the interface box below is the option available:

ICSP/SPI (In-Circuit Serial Programming interface) → is the interface standard 3 wires which uses 1, 3 and 4 pins in the Input/Output connector and it used mainly to communicate with Arduino controller. This type of interface does not use a slave select signal.

- I2C → a multi-drop 2-wires (5 and 9 pins of the input or output connector) also it allows with up to 127 slaves, signal master to communicate. With “I2C address” parameter one can set the I2C address.
- SPI → slave select (SS), this is identical with ICSP but it has slave and uses pin 7 of the I/O connector for slave select signal.
- UART → this interface in mostly used, it uses 1 and 4 pins of the I/O connector, typically, the pin 1 is used by the camera as receiver and uses pin 4 as transmitter. The “UART baudrate” parameter is used to set the baudrate.
- Analog/digital x → it has the ability to output the x value regarded as the largest of the analogue output of the object being detected, with a value between 0 and

3.3V (pin 3). Additionally, it shows if an object is detected or not as a digital signal (pin 1 of the I/O connector).

- Digital/Analogue y → it capable of outputting the y value of the largest object detected as an analogue output, with a value between 0 and 3.3V (pin 3). Consequently, it can illustrate whether or not the object is detected as a digital signal (pin 1 of the I/O connector).

In this project, the chosen interfaces for inter-communication between the cameras used (Pixy) and the Raspberry Pi computer was I2C interface, since more than one camera was used the use of this interface allow settings of address for each camera. These functions are not available in the others above mentioned communication interface. The interface on the pixy camera has the following features:

- The Pixy cameras work in a slave mode and required polling
- There are two weak 4.7 pull up to 5V resistors connected to the SDA and SCL signals.
- The signals are 5V tolerant.
- The I2C interface can be configured in configuration parameter of each camera.

The serial protocol

All interfaces have common serial protocol, whether you are using I2C, UART or SPI communication interfaces by Pixy camera, which are:

- In each frame, objects are sorted by size usually the larger object is sent first.
- The protocol is data efficiency
- Possibility of configuring the maximum number of object to be sent by each image frame, that is, "*Max blocks*" parameter.
- The I2C and SPI operate in slave mode and need polling to receive updates.
- If SPI and I2C interface is selected, when no objects are detected, and zeros will be send by Pixy, because, the camera work in a slave mode and something needs to be transmit.
- Each object is sent in an "object block" (see Table 2).
- All values in the object block have a size of 16-bit words, and the least significant byte is sent first (little endian). For example, when transmitting the sync word

0xaa55, the camera transmits 0x55 (that is; first byte) after it will send 0xaa (that is; the second byte).

The table [2] described the object block format.

Table 2: Object block format /32/

Bytes	16-bit word	Description
0, 1	y	Sync: 0xaa55 = normal object, 0xaa56 0 color code object
2, 3	y	Checksum (sum of all 16-bit words 2-6)
4, 5	y	Signature number
6, 7	y	Center object of x
8, 9	y	Center object of y
10, 11	y	Object Width
12, 13	y	Object Height

To separate frames, an extra sync word (0xaa55) is added. This mean that a new image frame is indicated by either, a) two sync words (0xaa55, 0xaa55) sent in a sequence or b) a normal sync followed by color code sync (0xaa55, 0xaa56) /32/.

6.3 Level Shifter

In this project, because the two cameras GPIO can only work with 5V while the Raspberry Pi GPIO runs at 3.3V, there is a need for level shifter for I2C communication between the two devices. Usually the most common converter does not work properly with I2C [], the 4-channel I2C safe bi-directional Logic converter – BSS138 by Adafrauit have been select as the preferable one (figure) /5/.

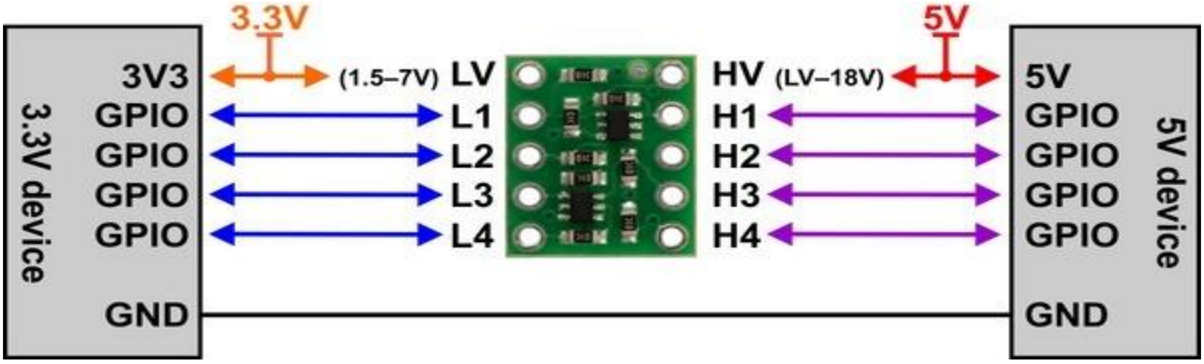


***Figure 25: Wiring diagram for connecting 5v and 3v devices through level shifter** /31/*

The 4-BSS138 Field Effect Transistors [FET] which have pull-up resistor of 10 kΩ; In which, at the side of the high level, it uses 10 V voltages up and 1.8 V voltages down on the low voltage side /1/.

6.4 Inter Processing Communication

Typically, the inter-process communication (IPC) means, a communication between processes (as the name implies), this process allows the transfer of information or data between different process which can be saved in the different machine or same physical machines. The most frequently used one are /4/:

- Pipes → this enable sequential communication related process from one place to the other.
- FIFO → this is like pipes; thus, it enables the distinct processes to communicate with each other since, in the file system, it assigns name to the pipe.
- Shared Memory → enable processes to communicate just by writing and reading data in a specific memory spaces.
- Mapped Memory → this is like shared memory; the slightly difference is the use of system file sharing of data.
- Sockets → this allow communication among the distinct processes also with different machines

According to criteria bellow; these kinds of inter processing communication (IPC) differ:

- To understand whether there is restriction of communication linked to processes such as processes with a common ancestor. It also checks whether there is restriction to unrelated processes that share the same file system or to any system (computer) connected to a network.
- Even if, the communicating process is restricted to only read or write data
- The number of processes allowed to be transmits.
- Even if, IPC controls transmission processes (for example, in the process of reading process halts until data is made available).

6.4.1 Mapped Memory

A mapped memory lets different processes to communicate through a shared file. Mapped memory assigns a name to that file that forms a link between process memory and a file. As operating system Linux is able to slit the file into page sized chunks after, it then copies the file into a virtual memory for its uses in a processes address space. Therefore, the process can read the contents of the file through memory access. This allows quick access to the file. In addition, mapped memory allocates a buffer to store the content of the file in the system memory. Read and write are done in this buffer also, it takes care of system update if the buffer is modified /4/.

6.4.2 Shared Memory

Shared memory is one of the easiest inter-process communication methods. The memory shared allows for two or more processes to have access to the same memory as though pointers and malloc were returned to the same actual memory. It is worth noting that if there were changes in the memory by a given process, other process will notice the change.

In shared memory, a piece of memory is shared by all processes, thus, this make it the fastest form of inter-processing communication, it does not require a system call or entry to the kernel, more so, copying of unnecessary data are avoided.

The kernel does not synchronize accesses to shared memory, it is necessary to provide a mechanism that handles synchronization. Because, data should not read by processes before is being written, and data should not be written by processes into the same memory space at the same time. The use of semaphores handles the synchronization problem /4/.

6.4.3 Pipes

The device allows unidirectional communication. That is, the written data contained in the end of the pipe is read back from the end. In addition to this, pipes are regarded as serial devices, meaning that the reading is done in the same sequence as it was written. Basically, a pipe serves the function of communicating between processes between parent and child or thread inside a single process.

There is a limitation to capacity of pipes data storage. Meaning that in case the write process writes faster than the reader process as well as pipe and that pipe will not be able to store data, then the process of writing will be blocked until there is availability of space. Contrariwise, if the reader will attempt to read something but there is no available data it will be blocked until there is data availability. Thus, the pipe automatically synchronizes the two processes.

Finally, file descriptor is caused by the creation of a pipe which can only be used for this process or for a child process. The use of pipe is not possible by unrelated processes /4/.

6.4.4 Sockets

It acts as bidirectional communication existing between device used to communicate between different processes going on at the same time as well as processes running on other machines. Sockets are widely used on internet programs such as Telnet, FTP, rlogin, talk and World Wide Web due to its facilities.

The socket concept is that, when it is being created, three parameters need to be specified such as: communication style, namespace and the protocol.

The need for communication style is that it tells how transmission of the data is done by the sockets. Data transmitted through sockets, it is composed into packets. This parameter decides how packets are taken care of, also the addressed from the sender to the receiver.

Socket Namespace, specifies ways in which socket addresses can be written. For instance, one socket address is signed to identify one end of one socket connection. Giving an example, when dealing with the local namespace, socket addresses remain ordinary filenames, with regard to internet namespace, a socket is made of internet protocol address or IP address of the machine and the port number. On the same machine, the port number distinguishes a socket among multiple others.

6.4.5 FIFOs

A first-in, first-out (FIFO) file is a pipe that is named in the file system. Any process can open or close FIFO, the linked processes to this kind of pipe need not to be related to each other. The terms “FIFOs” are also called “name pipes”.

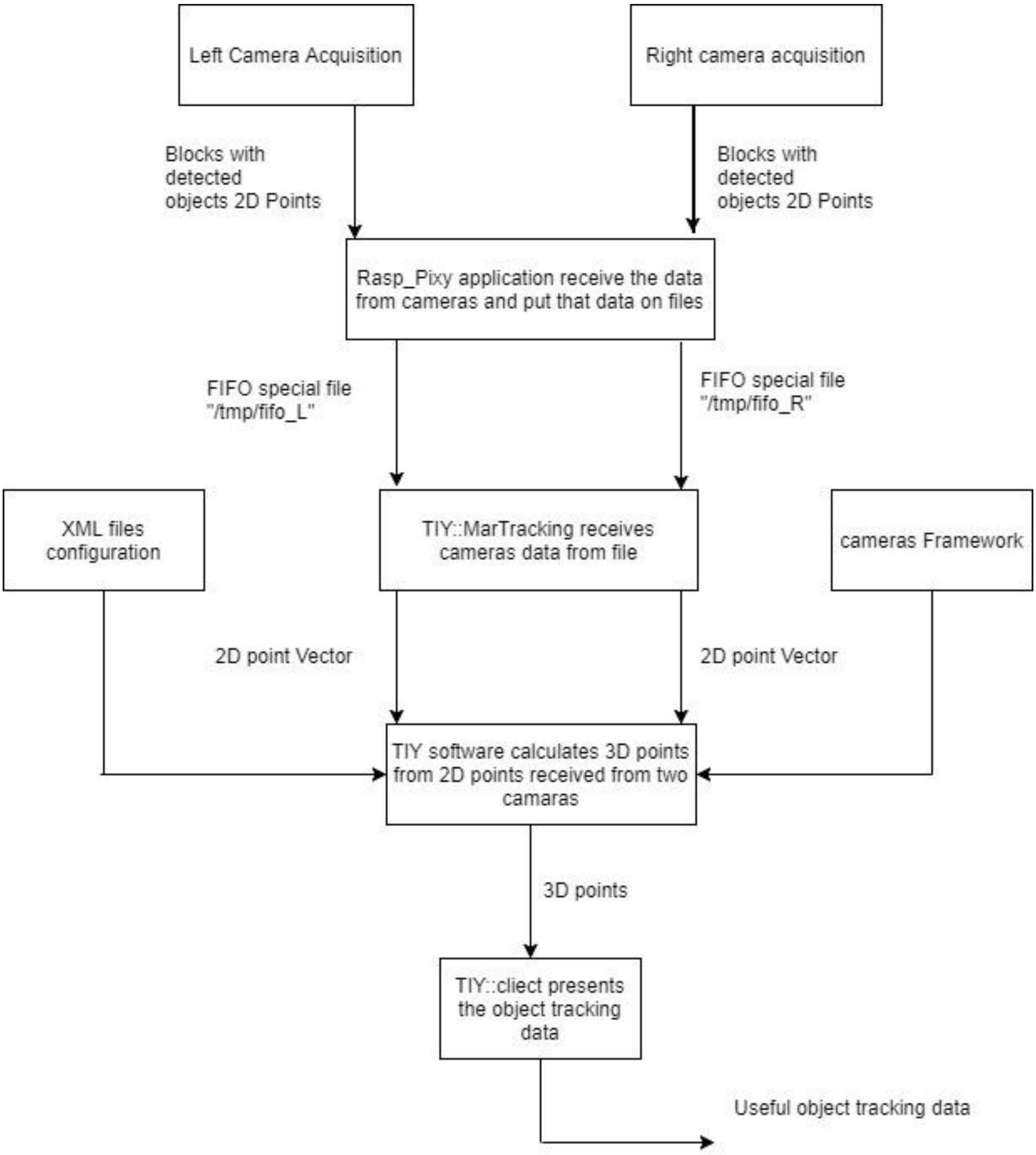
When creating a FIFO, it is required to specify its path and also the permissions level which will enable the processes to read and write in related file. Accessing FIFO can be archive just like an ordinary file. Also, the communication through FIFO need, one program to open it for writing and another program open it for reading. The C language provides libraries that can be used to perform such functions as read, write, and open a FIFO.

In FIFO, it can have multiple readers or multiple writers. For the pipes, there is a limited capacity in FIFO, so it depends on the system. The behaviour of FIFO differs in different systems, in windows, the behaviour of FIFO is like sockets that is, it can be used to communicate between process of different machine also it can allow two-way communication. While FIFO in Linux are unidirectional /4/.

6.4.6 Method Used

In this project, two named pipe (FIFO), one for each camera were used for the inter-process communication between the process that carries out the data acquisition by the Pixy cameras and Track It Yourself (TIY) software, the process responsible to get the 3D coordinates or positioning. This type of inter processing is chosen because it allows any two processes to communicate, unlike regular pipe, only processes with a common ancestor can communicate.

It was done in such a way that the dual processes run on the same machine, the process that receives data from the two cameras were created using two FIFO; one for either camera such as: `/tmp/fifo_Right` for the right camera data and `/tmp/fifo_Left` for data in left camera. Therefore, blobs written for acquisition of data process after it then read by process which does the calculations for getting the three-dimensional (3D) positioning, correspondingly for one of the two FIFO cameras.



Left Camera Acquisition
Right camera acquisition
Blocks with detected objects 2D Points
Blocks with detected objects 2D Points
Rasp_Pixy application receive the data from cameras and put that data on files
FIFO special file "/tmp/fifo_L"
FIFO special file "/tmp/fifo_R"
XML files configuration
TIY::MarTracking receives cameras data from file
cameras Framework
2D point Vector
2D point Vector
TIY software calculates 3D points from 2D points received from two camaras
3D points
TIY::cliect presents the object tracking data
Useful object tracking data

Figure 26: Data flow overview of the complete project.

6.5 Obtaining Blobs from Pixy Camera (Rasp_Pixy Process)

A part of the task in this project, is to obtain a binary large object (Blob(s)) using object detected position. This process is carried out by obtaining information or data from the used cameras and transfer the data to Track It Yourself software, the TIY then do the three-dimensional (3D) positioning from the two-dimensional (2D) data received from the two cameras. The Rasp_Pixy application using C++ programming language was developed to carry out this process and is explained below. In addition, this application depends on part of the developed library in (Annex A).

Firstly, it is required to set the I2C addresses interfaces for the two cameras. Note, the set addresses for these two cameras must correlate with the addresses set for each of the cameras using PixyMon software. In the case of this project, the right camera initialized with hexadecimal 10 address while the left camera initialized with 20 address as shown below:

```
Pixy pixy_R(0x10); // Pixy address of right Pixy camera
Pixy pixy_L(0x20); // Pixy address of left Pixy camera
```

For inter processing communication between cameras and computer, pipes were selected to serve the purpose and these pipes were created inside the directory proposed by Linux for temporary file storage location with the use command mkfifo, which includes write and read permission. Next two file descriptors were declared and initialize for each of the camera, to serve the purpose of pipe handling.

The used O_NONBLOCK flag is to open pipe only for writing, then when there is no process to read pipe an error will be returned.

```
const char* fifo_R = "/tmp/fifo_R";
const char* fifo_L = "/tmp/fifo_L";
// create the FIFO (named pipes)for both cameras
mkfifo(fifo_R, 0666);
mkfifo(fifo_L, 0666);
//open message from the fifo
int fd_R = open(fifo_R, O_WRONLY|O_NONBLOCK);
int fd_L = open(fifo_L, O_WRONLY|O_NONBLOCK);
```

Until the reading process stops, this application is limited of running an infinity loop, after the initial settings.

At every sequence, we then read a data of each of the cameras, as shown in the code below:

```
            // get block from Pixy cameras
            blocks_R = pixy_R.getBlocks();
            blocks_L = pixy_L.getBlocks();
```

According to Pixy camera configuration used in this project, the camera has the capacity of returning data from several object colors signature placed in front of it, that is, multiple points. Therefore, to achieve 3D processing positioning, it is required to use the exact number of points for each of the cameras. Thus, the lowest number of detected points between the two cameras was chosen, as shown below:

```
            // get the number of detected objects (camera with the minimum
detected)
            if (blocks_R < blocks_L)
                  num_blocks = blocks_R;
            else
                  num_blocks = blocks_L;
```

When there are signatures detected for one object or more by the cameras, it then writes it to individual pipes location. Though there is a possibility that the camera is unable to detect an object, for instance, no object to be detected in the field view of the cameras. The code below handles the individual process as explained in this paragraph:

```
if (blocks_R && blocks_L) // print the detected objects
 {
    if ((i%1)==0)  // Pixy outputs data 50 times per second
    {
       i = 1;
       std::cout << "Left number of blocks: " << blocks_L << std::endl;
       std::cout << "Right number of blocks: " << blocks_R << std::endl;

       std::cout << "Number of detected objects: " << num_blocks << 
std::endl;
       std::cout << "R: ";
       for(int i = 0; i < num_blocks; i++)
       {
           //pixy_R.blocks[i].print();
           blob_temp << pixy_R.blocks[i].x << "\t" << pixy_R.blocks[i].y 
<< std::endl;
           blob_R = blob_temp.str();
           write(fd_R, blob_R.c_str(), (unsigned)strlen(blob_R.c_str()));
           blob_temp.clear();
           blob_temp.str("");
           std::cout << blob_R.c_str() << std::endl;
        }
        std::cout << "L: ";
        for(int i = 0; i < num_blocks; i++)
        {
           //pixy_L.blocks[0].print();
           blob_temp << pixy_L.blocks[i].x << "\t" << pixy_L.blocks[i].y 
<< std::endl;
            blob_L = blob_temp.str();
            write(fd_L, blob_L.c_str(), (unsigned)strlen(blob_L.c_str()))
            blob_temp.clear();
            blob_temp.str("");
            std::cout << blob_L.c_str() << std::endl;
         }
      }
      i++;
   }
   else
   {
      std::cout << "No blocks detected" << std::endl;
   }
```

Lastly, when there is a hit on the keyboard to stop the processes, that is exit the infinity loop, the program called the function below, and the file descriptor used to create pipes are closed and the processes ends.

```
close(fd_R);
close(fd_L);
```

6.6 Compute 3D Marker Position using TIY

The Track It Yourself provides the possibilities of building your own 3D marker object tracking system, given that it is an open source code.

Originally the software was designed to 3D marker tracking system using videos from two cameras, as input. If the two cameras can view a marker at the same time, then the related 3D position to the used cameras will be calculated by triangulation, if the view objects maker are see by cameras from different angle and position.

In this project, data input to TIY software is culled from two text files, so it was required to edit the TIY source code and some code segments previously used for video signal input were removed.

Therefore, some of the unneeded code were eliminated, to simplify the remaining code, then to meet the requirement of this work, additional code was developed and added to the source code.

In the TIY library, there are multiple source files, among others, the useful one for the sake of this project are explained as follow.

6.6.1 Server

The server is parts of TIY source files, primarily it has some settings, and it is edited the xml file that is present in TIY root folder. The server calls the function responsible for data collection from cameras, for instance, from images, point files, or video files. The method used for this project, since it uses pipes for inter processing communications, then data acquisition is done from two text files using two dimensional points.

Basically, two vectors were created, one vector for one camera and another for second cameras, like:

```
// ----------------------------------------------------------------------
---------------
      // Extract (or read from file) 2D points
// ----------------------------------------------------------------------
---------------
  // Point vectors with values from Pixy's
  std::vector< cv::Point2f > points_2D_left, points_2D_right;
```

Then, the function to load the two vectors using 2D points from text file needs to be called.

```
          m_track.get2DPointsFromFile('l', &points_2D_left);
          m_track.get2DPointsFromFile('r', &points_2D_right);
```

The 3D points are then calculated from the 2D points, after the functions called, with this function:

```
// ----------------------------------------------------------------
// Compute 3D points from 2D points
// ---------------------------------------------------------------
cv::Mat points_3D = m_track.get3DPointsFrom2DPoints(points_2D_left,
points_2D_right);
```

The server application produces the desired data, after the 3D points have been calculated, and the edited xml files are affected due to the outputs of the data by server application.

6.6.2 MarkerTracking Application

The MarkerTracking is one of the source code of TIY, this source code includes the most valuable operation function by TIY software.

In this source file, it required first to create two file descriptors for our temporary data storage. As explain before, the named pipes were used, one for each camera to store the 2D points of the two cameras, as shown below:

```
          // FIFO between Pixy's and TIY
          #define MAX_BUF 256
          const char* fifo_R = "/tmp/fifo_R";
          const char* fifo_L = "/tmp/fifo_L";
          char buf_R[MAX_BUF];
          char buf_L[MAX_BUF];

          // open the FIFO
          int fd_R = open(fifo_R, O_RDONLY);
          int fd_L = open(fifo_L, O_RDONLY);
```

Note, the two files descriptor created are only permit reading of the 2D points.

For this purpose, the get2DPointFromFile function was used. It is the major target with regards to reading 2D points from the corresponding pipes. This function takes a character variable to determine the left, or right camera, with a reference parameter to the vector position of the 2D points defined in the server file which is a vector to either the left, or the right camera. Primarily, the 2D point can be read from named pipes and store in a buffer, after, it then sends to a stringstream simplifying the operations of the read data. After, it read the points into a temporary variable, then, it placed it in 2D points vector of the individual camera. The left camera code is shown below, while the right camera process is in analog form.

```
    MarkerTracking::get2DPointsFromFile(const char camera, std::vector<
cv::Point2f > *points_2D)
    {
        cv::Point2f L_2D_point, R_2D_point;
        points_2D->clear();
    if(camera == 'r')
    {
       read(fd_R, buf_R, MAX_BUF);
       std::stringstream linestream_R(buf_R);
       while (linestream_R >> R_2D_point.x)
       {
           if (linestream_R >> R_2D_point.y)
           {
              points_2D->push_back(R_2D_point);
           }
           else
                 break;
        }
     }
```

The get3DPointsFrom2DPoints function is also important function; it permits the marking of 3D points from 2D vector points, for each of the two cameras. Additionally, the function receives two points; one for either camera, and an array with 3D points will be returned, computed from 2D points with the use of correspondence triangulation and optimization.

After these processes, the obtained 3D marker points, then, fit3DPointsToObjectTemplate function will be calculate the detected object position in 3D space.

6.6.3 Client

This application was provided by Track It Yourself developer, the purpose of client is to permit the extraction of needed data from the 3D tracking application. In this project, it was utilised in communicating with the other systems. Additionally, the application contains an infinite cycle, which produces pointing device positioning.

6.7 Chapter Conclusion

In this chapter, the details of the implementation of the project was presented from both a software and hardware point of view.

It starts with the description of the hardware, primarily; the emphasis is on the communication and integration of diverse component used. It then proceeds to the description of the software which is inter processes overview, which is the most important part, because of the low processing power of the used computer, this method provides the best possible performance for the project. In addition, the chapter also tells about modification of TIY source code which enables the performance of 3D marker tracking from 2D points, with the use of Pixy cameras.

Finally, the chapter also describes how an application has been developed to handle receiving data from the used cameras and give them to TIY application.

7. SYSTEM TESTING

The testing result of the application developed in the previous chapter is presented here.

7.1 Implementation and testing

The tests were carried out with Pixy cameras lenses which have 45-degree vertical field view, a 75-degree horizontal view and M12 mounting and focal length of 2.8 mm. in addition, the sensor used is an Ominivision OV9715, which has 1/ 4 size also with a resolution of 1280 x 800 pixels /28/.

The tests were made with Pixy tracking a small blue rectangle object 3cm length and 2cm breadth. The object distance to the camera on the workstation is about 1m. remember that for data output, the resolution provided by the pixy camera is 320 x 200 pixels. The rectangle object is detected by the tracking system with distance of 1m. But with 50cm distance the object is perfectly detected by the tracking system without false negatives.

Furthermore, in case there will be a change of distance where the camera is being mounted to the workstation. For example, assuming the user want to permanently mount the camera on 1.5m distance to the workstation, due to the resolution of 320 x 200 pixels for data output provided by pixy developer with the original lenses, the result might not be satisfactory.

To improve the detection of object in the workstation with distance of 1.5 meter. This then call for a need to replace the camera lenses of both camera. To know the right lenses little calculation, needs to be done.

7.2 Choosing the Right Lenses

Taken into account that, the field of view is 800 x 600, the distance of where the camera is mounted to the work station is 1.5 m and a sensor size of 1/3, then focal length needs to be known and it can be calculated using the following method /41/.

$$focal\ length \cong \frac{sensor\_size \times working\_distance}{field\ of\ view + sensor\_size} \qquad \textit{Equation 17}$$

Given that, 1/3 sensor has a 4.8 mm width, 3.6 mm height and 6 mm diagonal, according to /41/, therefore:

$$focal\ length \cong \frac{4.8 \times 1500}{800 + 4.8} \cong 8.9 \qquad \textit{Equation 18}$$

Thus, to replace the original lens of Pixy mounted with 1.5m distance, the 1/3 sensor size with 8mm lens is recommended and is available in the market. That is, Focal Length 8.0mm, F2.0, M12*0.5 Board Lens. Designed for 1/3 Image Sensor. Angle Of View 42° /2/. In addition, the instruction for lens and filter replacement can be found here /19/. Warning! when a new lens and filter are replaced with the original lens on Pixy camera, a new camera calibration is required and the instruction is provided in the Annex D on this report.

For the case of this project, the original lens on the Pixy cameras were used for the calibration, the results below shown the calibration results of the right camera:

Intrinsic parameters:

```
Focal Length:          fc = [ 252.77930   252.80260 ] +/-
[ 4.40510   4.08791 ]
Principal point:       cc = [ 166.25039   97.33874 ] +/- [ 5.95531
4.40161 ]
Skew:             alpha_c = [ 0.00000 ] +/- [ 0.00000  ]   =>
angle of pixel axes = 90.00000 +/- 0.00000 degrees
Distortion:            kc = [ -0.43376   0.24373   0.00162   -
0.01308  0.00000 ] +/- [ 0.03641   0.06219   0.00329   0.00558
0.00000 ]
Pixel error:          err = [ 1.01679   0.63203 ]
```

Extrinsic parameters:

```
Translation vector: Tc_ext = [ -17.187892      -156.666642
467.686318 ]
Rotation vector:   omc_ext = [ -1.478283       -1.755087
0.830990 ]
Rotation matrix:    Rc_ext = [ -0.116891       0.548837           -
0.827716
0.988096  0.148201          -0.041271
0.100017  -0.822687         -0.559627 ]
Pixel error:           err = [ 4.28220         5.34302 ]
```

Calibration results of the left camera:

```
Intrinsic parameters

Focal Length:          fc = [ 331.53583   258.65272 ] +/-
[ 13.13245   9.08643 ]
Principal point:       cc = [ 177.91841   108.00549 ] +/-
[ 7.38701   6.33456 ]
Skew:             alpha_c = [ 0.00000 ] +/- [ 0.00000  ]   =>
angle of pixel axes = 90.00000 +/- 0.00000 degrees
Distortion:            kc = [ 0.45986   -2.10565   0.01556   -
0.03507  0.00000 ] +/- [ 0.17710   0.88614   0.01429   0.01178
0.00000 ]
Pixel error:          err = [ 1.95485   2.17360 ]

Extrinsic parameters:

Translation vector: Tc_ext = [ 1.843724       -177.904158
617.244803 ]
Rotation vector:   omc_ext = [ -1.439188      -1.782910
0.834178 ]
Rotation matrix:    Rc_ext = [ -0.148720      0.539517          -
0.828736
                               0.981948       0.179625          -
0.059276
                               0.116881       -0.822591         -
0.556492 ]
Pixel error:           err = [ 3.62934        4.97299 ]
```

7.3 Testing the Tracking System

The two cameras were placed on a working table and the object to be detected is placed in front of the cameras with distance of 1m. then, the test was carried out by running the tracking system application in two used computers, that is, starting the developed Rasp_Pixy and server TIY application at the same time to obtain the 2D object coordinates detected by the cameras.

In addition, the purpose of the tracking system is determining the rate which object is being detected. With low refresh rate, if the object is in motion on high speed, detection might not be 100% correct. But, in the case of this project an object at a position was tracked and it uses the maximum rate provided by the Pixy cameras, (the highest possible rate, which is 50 Hz).

Two different Raspberries were used for the tests, the Raspberry Pi 2 B+ and Raspberry Pi 3 B+. after several tests, it was concluded the best highest possible rate which tracking system can perform well. The Pi 2 B+ with CPU speed of 900 MHz is approximately 16 sample per second is the advised to use, and Pi 3 B+ with CPU speed of 1.2GHz, the rate of the data provided will not be able to control by the tracking system if there is an increased in the camera sampling rate.

For Raspberry Pi B+, the table below shows the estimate time required to carried out the TIY tracking system at overlocked CPU speed of 900 MHz.

Table 3: Model B processing at 5Hz.

Test	Undistortion lens (µs)	3D correspondence (µs)	3D triangulation (µs)	Time total (µs)
1	299	2880	12868	16047
2	295	1134	2166	3595
3	302	997	22105	23404
4	298	1012	9431	10741
Average	298	1505	11642	13445

For Raspberry Pi 2 B+, the table below shows the estimate time required to carried out the TIY tracking system at 900MHz CPU speed.

Table 4: Pi 2 Model B running at 16.67Hz.

Test	Undistortion Lens (µs)	3D correspondence (µs)	3D triangulation (µs)	Time total (µs)
1	252	1441	2102	3795

2	344	2375	3699	5418
3	198	2270	3341	5809
4	204	1032	2992	4228
Average	249	1779	3033	5061

7.4 Chapter 7: Conclusion

From the several tests carried out this chapter, it was shown that the implemented tracking system was successful. And then it was concluded that the Raspberry 2 B+ computer was a better choice of computer to use in processes the main TIY tasks in this project, as the refresh rate is approximately 17 Hz, compare to Model B which is 5Hz

8. CONCLUSIONS AND PROSPECTIVE WORK

In this dissertation, 3D marker tracking system was developed to provide solution for the proposed thesis provided by VAMK, a department of mechanical engineering. The goal was to develop a robotic vision system that can be used for teaching in Technobothnia, VAMK, in which the vision system should use low cost hardware, be easy to operate and must be simple to program. The task was to develop a system that can read the coordinate (x and y) position of a color object(s), extract data and calculate 3D position from the 2D received data and the result should be ready to use to teach industrial robot what to do.

The developed system consists of a stereo vision system. The system uses two cameras, and the developed application in the calculation of 3D position from the detected object. In addition, the detection algorithm based on color differences was used by the cameras which enable 2D object tracking and outputted data coordinates of object being detected. Then, the 3D object position is produced through the calculated 2D object data coordinates, and the outputs are presented by the client application for robot teaching.

Finally, the tests confirm that the systems were successfully built, but because of the low resolution of the used Pixy cameras and the Raspberry pi computer, the functionality of the developed system are imperfect.

8.1 Prospective Work

Developing an object tracking system via color, is the main objectives of this thesis work and that was achieved. Thus, the system is limited and some areas need to be improved. However, some of them have been figured out, but because of the time schedule for this work, they cannot be implemented.

The cameras used, processes and outputted data with 320 x 200 pixels resolution, for an industrial solution, the problem of low resolution might occur when there is a need to cover the whole workstation because the field of view may be too large to be cover by the (320 x 200) resolution.

Secondly, when testing, it was discovered that the original lens captures the object to be detected clearly on 1m distance from where the camera is mounted to the work bench where the object is placed, if the distance increases, like 1.5m, the lens cannot detect the object correctly. Changing of the camera lens and the use of IR pointing device to detect where the object is place might be a solution to this in future.

Last but not the least, in real time, the used Raspberry Pi 2 gives the require processing power to implement the tracking system. But while testing the real implementation with robot, if the tracking system is unable to track the movement of the object, then the use of Raspberry pi with greater processing power is required.

References

/1/ 4-channel I2C-safe Bi-directional Logic Level Converter [BSS138] ID: 757 - $3.95 : Adafruit Industries, Unique & fun DIY electronics and kits. (n.d.). Retrieved November 26, 2017, from https://www.adafruit.com/product/757

/2/ 8.0mm, F2.0 Board Lens. (n.d.). Retrieved January 17, 2018, from http://www.m12lenses.com//ProductDetails.asp?ProductCode=PT%2D0820

/3/ Aguilar-Torres, M. A., Argüelles-Cruz, A. J., & Yánez-Márquez, C. (2008). A Real Time Artificial Vision Implementation for Quality Inspection of Industrial Products. In *Electronics, Robotics and Automotive Mechanics Conference, 2008. CERMA '08* (pp. 277–282). https://doi.org/10.1109/CERMA.2008.75

/4/ alp.pdf. (n.d.). Retrieved from http://www.cse.hcmut.edu.vn/~hungnq/courses/nap/alp.pdf

/5/ AN10441.pdf. (n.d.). Retrieved from https://cdn-shop.adafruit.com/datasheets/AN10441.pdf

/6/ Background Subtraction Website. (n.d.). Retrieved November 26, 2017, from https://sites.google.com/site/backgroundsubtraction/

/7/ Bülthoff, I., Bülthoff, H., & Sinha, P. (1998). Top-down influences on stereoscopic depth-perception. *Nature Neuroscience, 1*(3), 254–257. https://doi.org/10.1038/699

/8/ Camera Calibration Toolbox for Matlab. (n.d.-a). Retrieved December 5, 2017, from http://www.vision.caltech.edu/bouguetj/calib_doc/

/9/ Camera Calibration Toolbox for Matlab. (n.d.-b). Retrieved December 5, 2017, from http://www.vision.caltech.edu/bouguetj/calib_doc/htmls/example.html

/10/ Canny Edge Detector — OpenCV 2.4.13.5 documentation. (n.d.). Retrieved January 28, 2018, from https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html

/11/ Color-tracking Explanation - CMUcam4 - CMUcam: Open Source Programmable Embedded Color Vision Sensors. (n.d.). Retrieved November 26, 2017, from http://www.cmucam.org/projects/cmucam4/wiki/Color-tracking_Explanation

/12/ File:Drawing of Raspberry Pi model B rev2.svg. (n.d.). In *Wikipedia*. Retrieved from https://en.wikipedia.org/wiki/File:Drawing_of_Raspberry_Pi_model_B_rev2.svg

/13/ Fundamental Guide for Stereo Vision Cameras in Robotics - Tutorials and Resources. (n.d.). Retrieved November 26, 2017, from https://www.intorobotics.com/fundamental-guide-for-stereo-vision-cameras-in-robotics-tutorials-and-resources/

/14/ gaschler. (2017). *tiy: NO LONGER MAINTAINED*. C++. Retrieved from https://github.com/gaschler/tiy (Original work published 2015)

/15/ Google Code Archive - Long-term storage for Google Code Project Hosting. (n.d.). Retrieved November 26, 2017, from https://code.google.com/archive/p/tiy/

/16/ Have You Had A Raspberry Pi Before? (2016, July 4). Retrieved January 29, 2018, from https://www.theodysseyonline.com/raspberry-pi

/17/ I2C Library | Wiring Pi. (n.d.). Retrieved December 2, 2017, from http://wiringpi.com/reference/i2c-library/

/18/ Image Formation by Lenses and the Eye. (n.d.). Retrieved November 26, 2017, from http://hyperphysics.phy-astr.gsu.edu/hbase/Class/PhSciLab/imagei.html

/19/ Installing the IR-LOCK Filter. (n.d.). Retrieved January 17, 2018, from https://ir-lock.com/pages/installing-ir-lock-filter-onto-pixy

/20/ Introduction — OpenCV 2.4.13.4 documentation. (n.d.). Retrieved November 26, 2017, from https://docs.opencv.org/2.4/modules/core/doc/intro.html

/21/ Kim, H.-B., & Sim, K.-B. (2010). A particular object tracking in an environment of multiple moving objects. In *ICCAS 2010 - International Conference on Control, Automation and Systems* (pp. 1053–1056).

/22/ lect5.pdf. (n.d.). Retrieved from https://courses.cs.washington.edu/courses/cse455/09wi/Lects/lect5.pdf

/23/ machine-vision-handbook.pdf. (n.d.). Retrieved from http://www.ukiva.org/pdf/machine-vision-handbook.pdf

/24/ McDuffie, J. (2015). *McDuffie, James: Pixy Camera Raspberry Pi Interface*. Python. Retrieved from https://github.com/omwah/pixy_rpi (Original work published 2014)

/25/ OReilly Learning OpenCV.pdf. (n.d.). Retrieved from http://www.bogotobogo.com/cplusplus/files/OReilly%20Learning%20OpenCV.pdf

/28/ Overview - CMUcam5 Pixy - CMUcam: Open Source Programmable Embedded Color Vision Sensors. (n.d.). Retrieved November 26, 2017, from http://cmucam.org/projects/cmucam5

/29/ paper101.pdf. (n.d.). Retrieved from http://www.bmva.org/bmvc/2011/proceedings/paper101/paper101.pdf

/30/ Peyman Alizadeh MSc. Thesis Corrected_2_2.pdf. (n.d.).

/31/ Pololu - Logic Level Shifter, 4-Channel, Bidirectional. (n.d.). Retrieved November 28, 2017, from https://www.pololu.com/product/2595

/32/ Porting Guide - CMUcam5 Pixy - CMUcam: Open Source Programmable Embedded Color Vision Sensors. (n.d.). Retrieved November 26, 2017, from http://cmucam.org/projects/cmucam5/wiki/Porting_Guide?version=35

/33/ Poudel, P., & Shirvaikar, M. (2011). Optimization of Image Processing Algorithms on Mobile Platforms, *7871*. https://doi.org/10.1117/12.876520

/34/ Powering Pixy - CMUcam5 Pixy - CMUcam: Open Source Programmable Embedded Color Vision Sensors. (n.d.). Retrieved November 26, 2017, from http://www.cmucam.org/projects/cmucam5/wiki/powering_Pixy

/35/ Raspberry Pi in Spanish. (n.d.). Retrieved January 29, 2018, from https://www.raspberryshop.es/

/36/ RPi Hub - eLinux.org. (n.d.). Retrieved November 26, 2017, from https://elinux.org/RPi_Hub

/37/ RPi Low-level peripherals - eLinux.org. (n.d.). Retrieved November 26, 2017, from https://elinux.org/RPi_Low-level_peripherals

/38/ spry300.pdf. (n.d.). Retrieved from http://www.ti.com/lit/wp/spry300/spry300.pdf

/39/ Stereo Vision: Facing the Challenges and Seeing the Opportunities for ADAS Applications. (n.d.). Retrieved January 7, 2018, from https://www.embedded-vision.com/platinum-members/texas-instruments/embedded-vision-training/documents/pages/stereo-vision-adas

/40/ Sylvain Calinon. (n.d.). Retrieved February 9, 2018, from http://www.calinon.ch/research.php

/41/ Technical Application Notes. (n.d.). Retrieved January 17, 2018, from https://eu.ptgrey.com/tan/10694

/42/ The Boost C++ Libraries. (n.d.). Retrieved November 26, 2017, from https://theboostcpplibraries.com/

/43/ tutorialspoint.com. (n.d.-a). Image Formation on Camera. Retrieved November 26, 2017, from https://www.tutorialspoint.com/dip/image_formation_on_camera.htm

/44/ tutorialspoint.com. (n.d.-b). Perspective Transformation. Retrieved November 26, 2017, from https://www.tutorialspoint.com/dip/perspective_transformation.htm

/45/ Vision Sensors Information | Engineering360. (n.d.). Retrieved December 11, 2017, from http://www.globalspec.com/learnmore/video_imaging_equipment/machine_vision_inspection_equipment/vision_sensors

/46/ Welch, G., & Foxlin, E. (2002). Motion tracking: no silver bullet, but a respectable arsenal. *IEEE Computer Graphics and Applications*, *22*(6), 24–38. https://doi.org/10.1109/MCG.2002.1046626

/47/ WiringPi. (n.d.). Retrieved November 28, 2017, from http://wiringpi.com/

/48/ Zhang, Z. (2000). A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *22*(11), 1330–1334. https://doi.org/10.1109/34.888718

/49/ DM_AlexandreAbreu_2015_MEEC.pdf, http://recipp.ipp.pt/bit stream/10400.22/7991/1/DM_AlexandreAbreu_2015_MEEC.pdf

Annex A. Pixy library and WiringPi

The WiringPi is a PIN built access GPIO library. It is written in C programming language, it is released under the GNU Lesser Public License version 3 (LGPLv3), used in the Raspberry Pi. This is library is used in this project to enable the use of I2C communication interface on Raspberry Pi computer and Pixy devices.

The WiringPi library includes also a command line utility gpio which make possible the programming and setup of the GPIO pins. In addition, with gpio command one can read and write the input and output pins.

The library is also extendable and modules are provided to support external GPIO interfaces and among these module is Devlib, which is a set of library routines implemented with the use of wiringPi to provide easy access to some common peripherals /47/.

A.1. Installation Guild

The wiringPi library is maintained under GIT repository. The instruction below is only applied to library installation in the Raspbian OS.

Step 1:

Check that there has not been any wiringPi installation, go to terminal window on your system, then execute command

```
gpio -v
```

if something shows up, then mean it has been installed. Next is to check if the installation is through source or standard package. If the installation is from the source then you can proceed but if otherwise, it need to be removed by execute this command:

```
sudo apt-get purge wiringpi
hash -r
```

Step 2:

If GIT is not installed yet, it is recommended to install it, to make possible the downloading of wiringPi, with this command:

```
sudo apt-get install git-core
```

In case of any error after the above step, ensure that latest version of Raspbian is installed:

```
sudo apt-get update
sudo apt-get upgrade
```

Step 3:

Download WiringPi from GIT by execute command:

```
git clone git://git.drogon.net/wiringPi
```

Step 4:

After the above step, the next is to build and install the library on the operating system using the command:

```
cd ~/wiringPi
./build
```

Step 5

Lastly, to check if the library is successfully installed, execute `gpio` command:

```
gpio -v
gpio readall
```

A.2. I2C Library

The includes wiringPi library make it easy to use raspberry Pi on board I2C interface.

It is required to load the I2C driver in the kernel before using the I2C interface /17/

```
gpio load i2c
```

With the use of load gpio command as above, the default baud rate is 100Kbps, but if one need another speed other than the default, for example, a 1000Kbps transmission speed is required, it can be selected using this command

```
gpio load i2c 1000
```

To use wiringPi library, it is required to include the header file on the developed source code program like:

```
#include <wiringPi.h>
```

In addition, with the use of compile line, like Linux command line, it is required to include the parameter below on your command line, to pass an instruction to the compiler to load the library:

```
-lwiringPi
```

For example, for the project it was used as

```
g++ raspixy.cpp -ggdb -lwiringPi -I ./ -o raspixy
```

lastly, the use of I2C library can be made possible in the program, by include the following statement in the developed source code

```
#include <wiringPiI2C.h>
```

A.3. TPixy Libraries and Pixy

The use of Pixy.h library is to permits some basic operations on Pixy cameras. Its operation relied on wiringPiI2C.h and TPixy.h libraries. In this project, the creation of TPixy.h and Pixy.h were achieved using the basis instruction provided by the Pixy developers in /40/ also with a project created but using Pixy camera with SPI communication interfaces in /24/

Therefore, to obtains 2D data from the two cameras, the following are the selected functions, the `(void init (uint8_t addr))` function is a place where I2C address must assigned, to enable the initialization of pixy camera. While, `(uint16_t getWord ())` function is responsible for returning a data block of 16-bit length from the camera and `(uint16_t getByte ())` function returning a data block with 8-bit size. Lastly, `(int8_t send (uint8_t *data, uint8_t len))` handle the operation of sending a data block with 8-bit size, which can also be used to send commands to the camera.

The TPixy.h library enable the system to get a block data, and in each block, it includes data of each object that is being detected, while all the data of each block received from the detected object is related to (x, y) 2D positioning of each object that is being detected.

Annex B. Track It Yourself (TIY) Installation and Configuration

Note: the instructions in this annex is only for Linux based system on how TIY can be installed and configured. For window based system it is required to follow the online documentation of TIY in /14/.

It is required to install the dependencies. This varies on the version of library:

- **OpenCV (≥ 2.2):**
  `sudo apt-get install libopencv-dev`
- **Boost (≥ 1.46):**
  `sudo apt-get install libboost-dev libboost-filesystem-dev libboost-system-dev libboost-date-time-dev lib-boost-thread-dev`
- **CMake:** (necessary because the TIY is built in this project, optional if otherwise):
  `sudo apt-get install cmake build-essential cmake-curses-gui`

There are two options available to install TIY on system, a), using Pre-build package: one single installation file, no building necessary and b), build from source (based on CMake) which permits further customization. Thus, in this project the b) method was chosen and the steps are shown as follow:

B. Build from source

1. Download and unzip the newest `tiy-x.x..zip` file from the download section in /14/ /49/.
2. In the terminal under tiy directory, go to Release folder using `cd` command, and use `cmake` to build the make file:
   `ccmake ../src`
3. Press `C key` from the keyboard to change the building options:
   `BUILD_client` ***ON***: the client will also be build.

`BUILD_server` ***ON***: the server will also be build.

`CMAKE_BUILD_TYPE` (set this option ***Release***) by pressing the **`Enter`** `key` on your keyboard and then type Release, this is for a test version

`USE_ARAVIS` ***OFF***: this option is set to OFF because we do not use Aravis cameras.

4. Then, Press `C key` few times and the `G key` to generate makefile
5. Use make command to make the project.
   ```
   make
   ```
6. Lastly, install the TIY library with:
   ```
   make install
   ```

after successfully done all these steps above without an error, the TIY is install in the Linux system and is ready to be used.

Annex C. Electronics Circuit Schematic Diagram

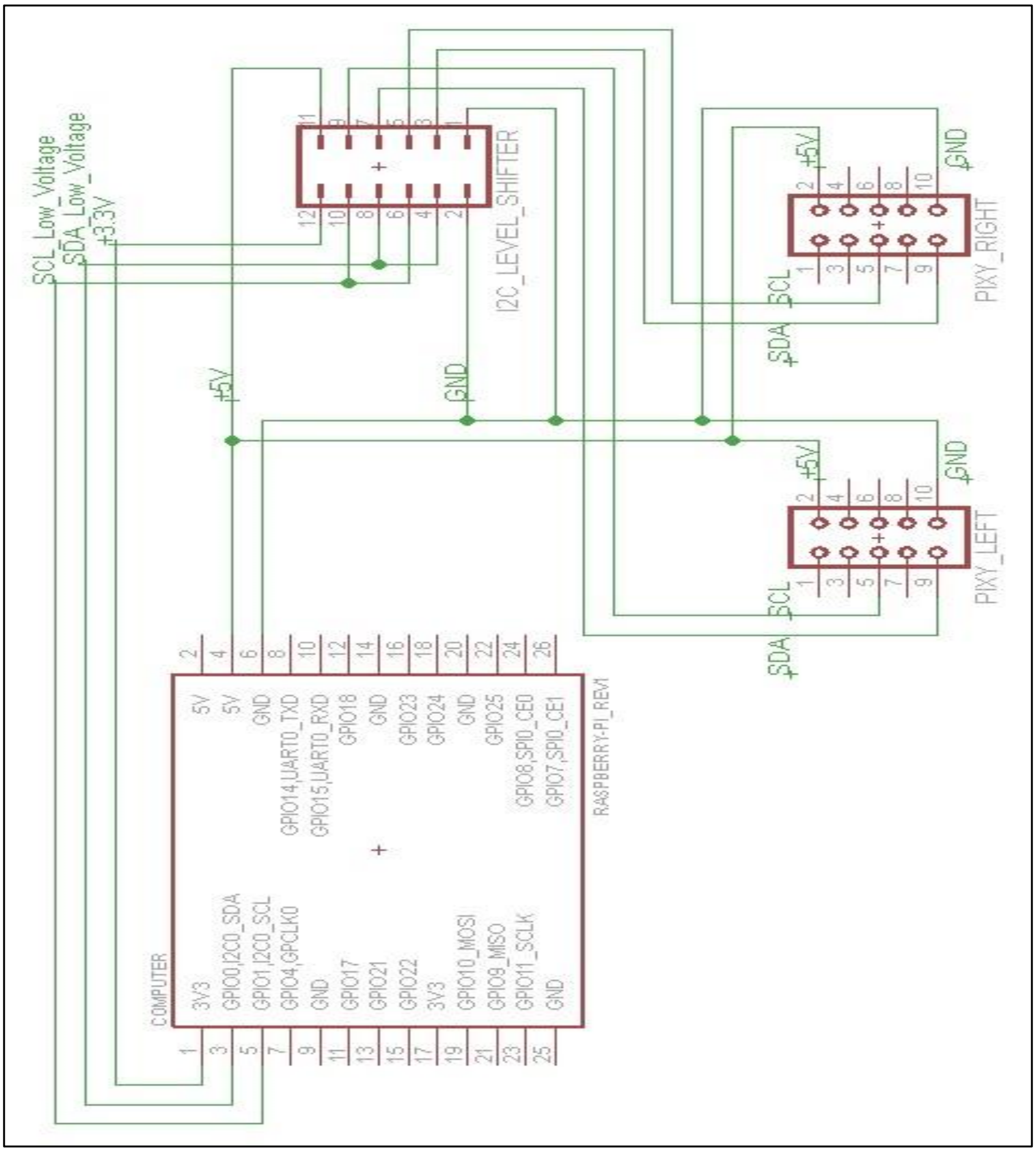


Figure 27:Schematic diagram of the electronics connection of the project

Annex D. Pixy Camera Calibration with Matlab

To use the two cameras as one stereo camera, it is required both intrinsic and extrinsic camera parameters, that is camera calibration is required. This calibration can be achieved with the use of a Matlab camera calibration toolbox (http://www.vision.caltech.edu/bouguetj/calib_doc/) /9/ and TIY calibration toolbox in which, this is recommended by the developer (gaschler, 2015/2017).

Thus, the camera calibration using TIY calibration toolbox and Matlab calibration toolbox can be achieved by following the instruction below:

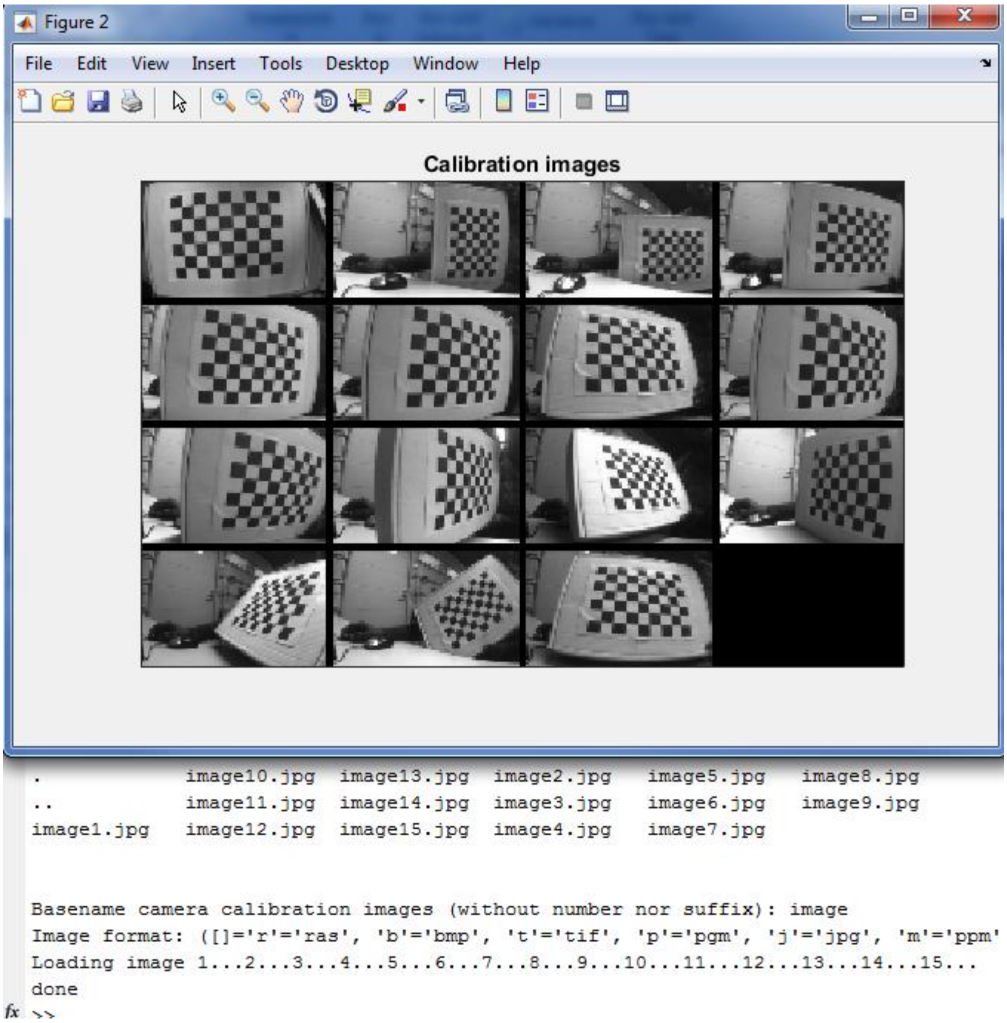
1. Firstly, download and unzip the `tiy_calibration.zip` file from TIY release section /14/ in a separate folder.
2. Download and unzip the Matlab toolbox from the vision Caltech download section /9//49/.
3. Then, copy all the MAT files from the extracted `tiy_calibration/camera_calibration/toolbox_calib` folder.
4. Also, it is required a checkboard pattern to carry out the camera calibration. The sample of the pattern can be printed out from `tiy_calibration/camera_calibration/perttern.pdf` as large as possible and fixed into a completely flat plate /49/.
5. Then, take at least `10-15 snapshots` of the checkboard pattern filling the screen in different angles for both left and right cameras using the used camera for the project.
6. Then, there is need for stereo snapshots, this can be taken by TIY server example, when activating the `<do_log_frame>` option in the config_run_parameters.xml file configuration by pressing the `SPACE` on the keyboard for each snapshot. The files are stored in the `tiy_log` folder, in home directory or TIY bin. Note: snapshots are overwritten when the server application is being restarted!
7. Then, take a stereo snapshot of the checkboard pattern lying preferably flat on a defined reproducible position in the sight of both cameras.

D.1. Main Calibration Processes

With the image data from both cameras and the taken stereo snapshot, then one can proceed with the calibration process in /9/. Open your Matlab application software, browse to the directory where you have your downloaded file. Then select the example folder `camera_calibration/toolbox_calib`, which contains the images from one the cameras (firstly, start with the right camera snapshots) by copying folder containing this images to this folder.

Then, carry out the steps below:

- Run the main matlab calibration `calib_gui` script
- On the mode selection window appears on the screen, select the first choice, Standard.
- The `Calibration Toolbox Window` appears, click on `Image names` button in the Camera Calibration Toolbox window. Enter the base name of the calibration images as (`image`) and the image format (`jpeg`). Then all the images are loaded in the memory (through the command `Read images` that is automatically executed) as shown in figure [28]

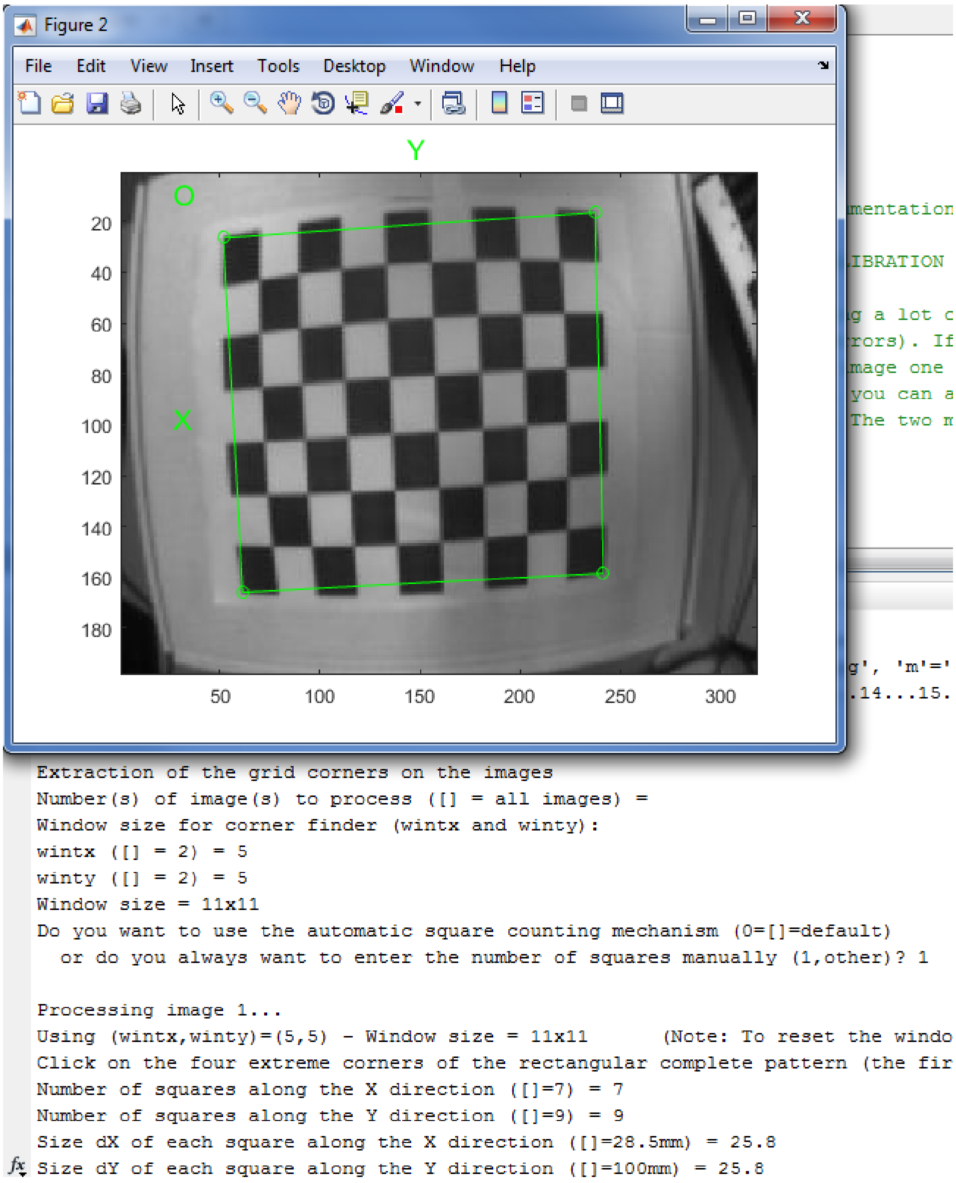


```
.            image10.jpg  image13.jpg  image2.jpg   image5.jpg   image8.jpg
..           image11.jpg  image14.jpg  image3.jpg   image6.jpg   image9.jpg
image1.jpg   image12.jpg  image15.jpg  image4.jpg   image7.jpg


Basename camera calibration images (without number nor suffix): image
Image format: ([]='r'='ras', 'b'='bmp', 't'='tif', 'p'='pgm', 'j'='jpg', 'm'='ppm'
Loading image 1...2...3...4...5...6...7...8...9...10...11...12...13...14...15...
done
fx >>
```

Figure 28: Capture image from the calibration toolbox after loaded all images

- Click the `Extract grid corners` button in the Camera `Calibration Toolbox Window.`
- Press `Enter` key (with an empty argument) to select all the images. After, select the default window size of the corner finder: `wintx = winty = 5` by pressing the Enter key with argument 5 for both wintx and winty questions. This lead to effective window of size 11x11 pixels.
- In the next question, manual enter the number of squares most be chosen, this is because the used camera (PIXY) has low resolution of (300x200) and it will be had for toolbox to automatically find the squares. Thus, enter 1 [figure 29].



```
Extraction of the grid corners on the images
Number(s) of image(s) to process ([] = all images) =
Window size for corner finder (wintx and winty):
wintx ([] = 2) = 5
winty ([] = 2) = 5
Window size = 11x11
Do you want to use the automatic square counting mechanism (0=[]=default)
  or do you always want to enter the number of squares manually (1,other)? 1

Processing image 1...
Using (wintx,winty)=(5,5) - Window size = 11x11      (Note: To reset the windo
Click on the four extreme corners of the rectangular complete pattern (the fir
Number of squares along the X direction ([]=7) = 7
Number of squares along the Y direction ([]=9) = 9
Size dX of each square along the X direction ([]=28.5mm) = 25.8
Size dY of each square along the Y direction ([]=100mm) = 25.8
```

Figure 29: Snapshot, setting number and size of the checkboard pattern

- Then provide the number of square in the checkerboard pattern: in the X direction put 7 squares and 9 squares in the Y direction. While for each size of the square put 25.8 mm as shown in [figure 29]
- Click on the four extreme corners on the rectangular checkerboard pattern. Ensure that you follow the clicking rules, in this project, it is selected first the left-up corner, next, the right up corner, followed by the right down corner and lastly, the left down corner [figure 29].
- After the grid corner is being selected, also it is possible to reduce the image distortion by adjusting the radial distortion for kc. Note: The Pixy camera has high distortion and low resolution, therefore it is required to adjust the kc for all the images, for a better result. It is advisable to first try the initial guess for kc for each image and then refine it to the best corner extraction as much as you can. The figure [30] is the extraction corner for image [1] of the right camera after the distortion has being adjusted.

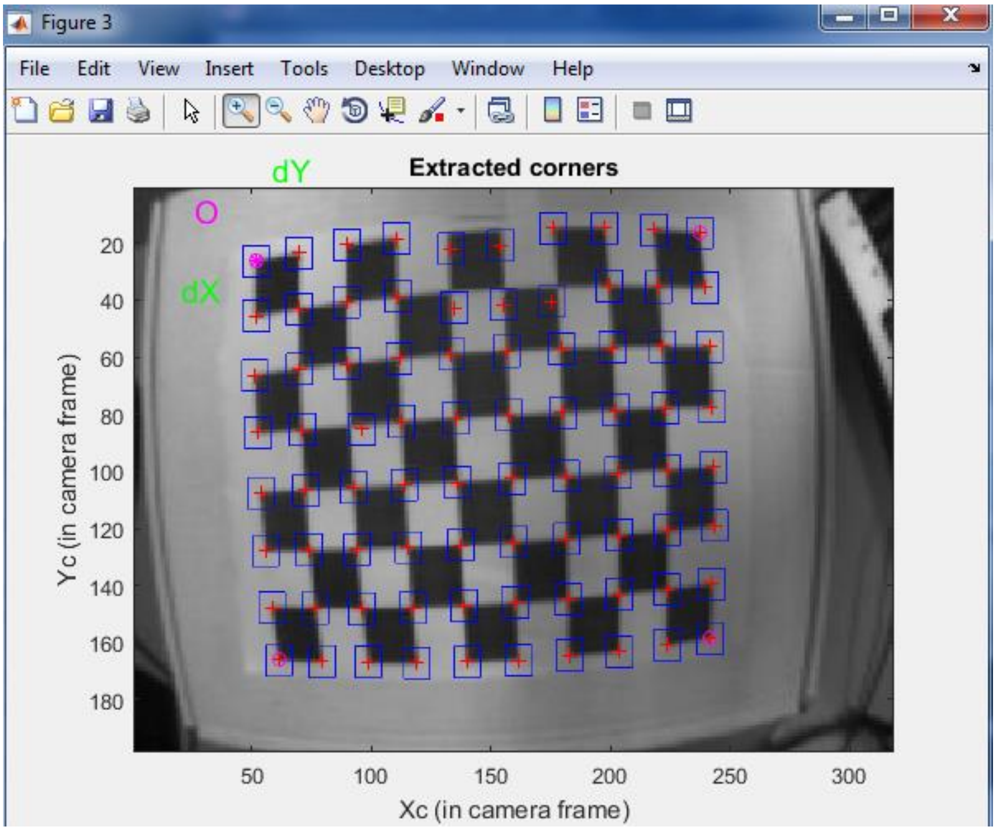


Figure 30: snapshot, Extract grid corner from image 1

- After the corner extraction, the matlab data file `calib_data.mat` is automatically generated. This file contains all the information gathered all through the corner extraction stage that is (image coordinates, corresponding 3D grid coordinates, grid sizes and so on.)
- **Main calibration step:**
- Then, go to `Calibration Toolbox Window`, and click on the `Calibration`, to run the main camera calibration procedure. Calibration is done in two steps: first initialization and then nonlinear optimization.

The initial step computes a close form solution of the calibration parameters based, which does not include any lens distortion. while nonlinear minimizes the total projection error.

- The result below shows the calibration parameters after initialization.

```
Calibration parameters after initialization:

Focal Length:          fc = [ 213.69467   213.69467 ]
Principal point:       cc = [ 158.50000   98.50000 ]
Skew:             alpha_c = [ 0.00000 ]   => angle of pixel =
90.00000 degrees
Distortion:            kc = [ 0.00000   0.00000   0.00000
0.00000   0.00000 ]

Main calibration optimization procedure - Number of images: 15
Gradient descent iterations:
1...2...3...4...5...6...7...8...9...10...11...12...13...14...15...
16...17...18...19...20...21...22...23...24...25...26...27...done
Estimation of uncertainties...done
```

- And after optimization, are shown the calibration result below:

```
Calibration results after optimization (with uncertainties):

Focal Length:          fc = [ 267.69700   271.53472 ] +/-
[ 8.88192   8.30409 ]
Principal point:       cc = [ 194.98816   98.20146 ] +/- [ 9.36895
8.46102 ]
Skew:             alpha_c = [ 0.00000 ] +/- [ 0.00000  ]   =>
angle of pixel axes = 90.00000 +/- 0.00000 degrees
Distortion:            kc = [ -0.14593   -0.32441   0.00724   -
0.02545  0.00000 ] +/- [ 0.10484   0.31321   0.00702   0.00784
0.00000 ]
Pixel error:          err = [ 1.67563   1.49509 ]

Note: The numerical errors are approximately three times the
standard deviations (for reference).
```

- To display the projections of the grid onto the original images, in the camera `Calibration Toolbox Window`, click on `Reproject on images`. These projections are computed based on the current intrinsic and extrinsic camera parameters in [figure 31]. Also, the projection error is shown in [figure 32].

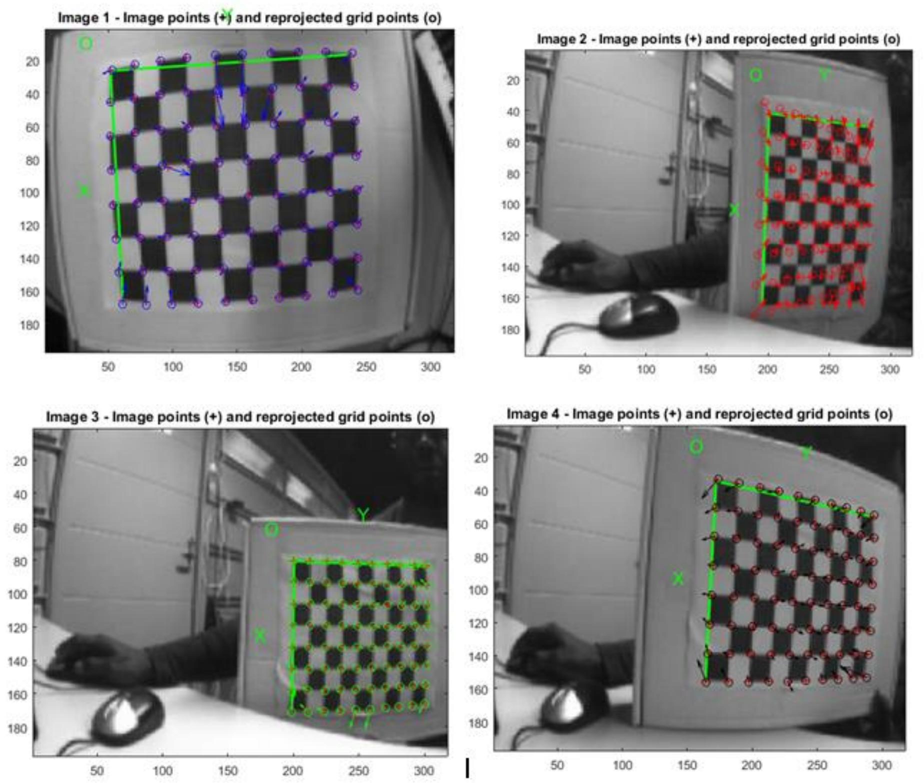


Figure 31: Snapshot, Reprojection images to image 1, image 2, image 3 and image 4

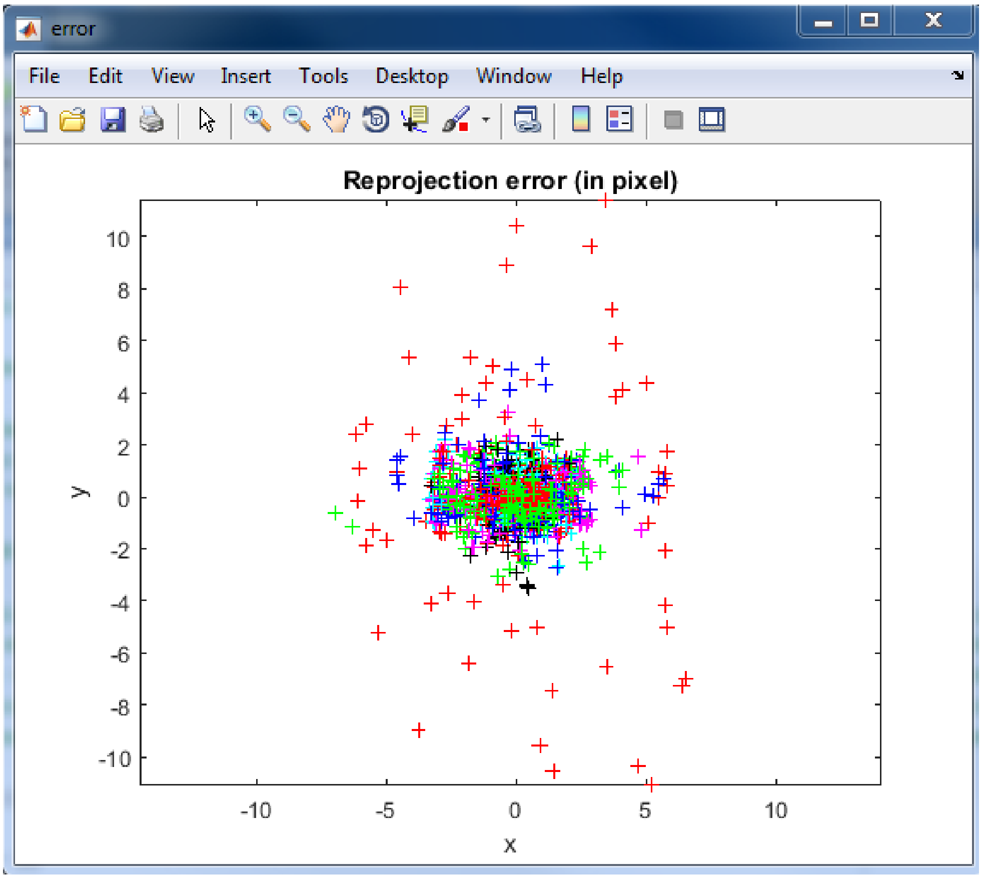


Figure 32: Snapshot, the reprojection Error

Click on `Show Extrinsic` in the Camera `Calibration Toolbox Window`, to view the extrinsic parameters. The extrinsic parameters (relative positions of the grids with respect to the camera) are then shown in form of a 3D plot showed in figure [33]

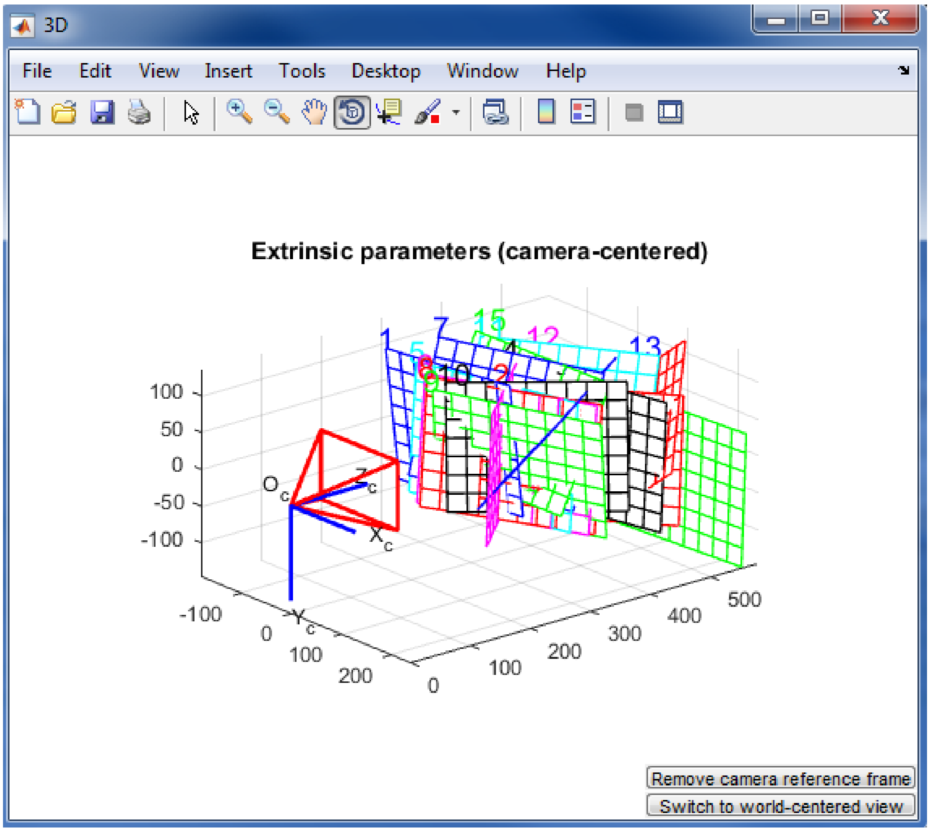


Figure 33: Snapshot, Extrinsic parameter (camera-centered) view

In figure [34] it shows the world-centered view, in this case every camera position and orientation is presented by green pyramid

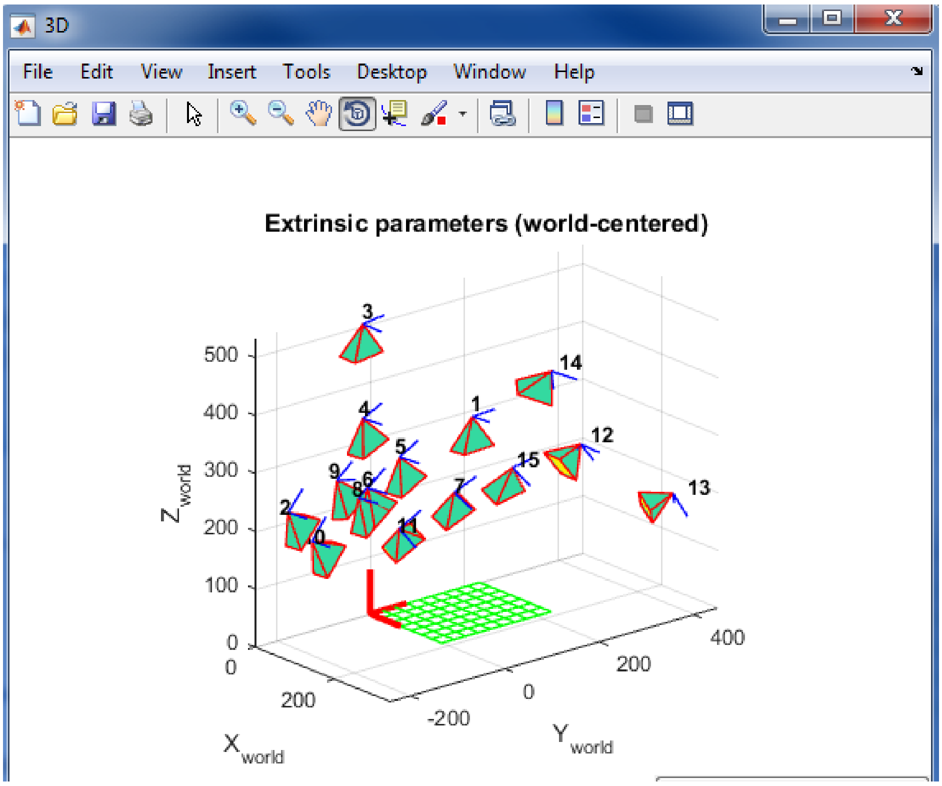


Figure 34: Snapshot, Extrinsic parameters (world-centered) view

From the figure [32], one can view the reprojection error with sparse area due to high distorted images. The used calibration toolbox provides the possibility of reducing the projection error, re-computing the image corners on all the images automatically. This can be done by click on `Recomp. Corners` in the calibration toolbox window and select a corner finder window size of `wintx` and `winty` to be 5 (`wintx and winty = 5`), which the other option unaltered. Then, it is required to perform another calibration by clicking the `Calibration` in the `Calibration Toolbox Window`. The results after calibration are the following:

Calibration results after optimization (with uncertainties):

Focal Length: fc = [252.77930 252.80260] +/- [4.40510 4.08791]
Principal point: cc = [166.25039 97.33874] +/- [5.95531 4.40161]
Skew: alpha_c = [0.00000] +/- [0.00000] => angle of pixel axes = 90.00000 +/- 0.00000 degrees
Distortion: kc = [-0.43376 0.24373 0.00162 -0.01308 0.00000] +/- [0.03641 0.06219 0.00329 0.00558 0.00000]
Pixel error: err = [1.01679 0.63203]

After optimization, in the Calibration toolbox window, click on `save` button to save the calibration results (intrinsic and extrinsic) in the Matlab file `Calb_Results.mat`.

Again, in the `Calibration Toolbox Window`, click on the `Reproject on image`, to re-project the grids onto the original calibration images, the first six images are shown in figure [number: reprojection grid onto first six images]

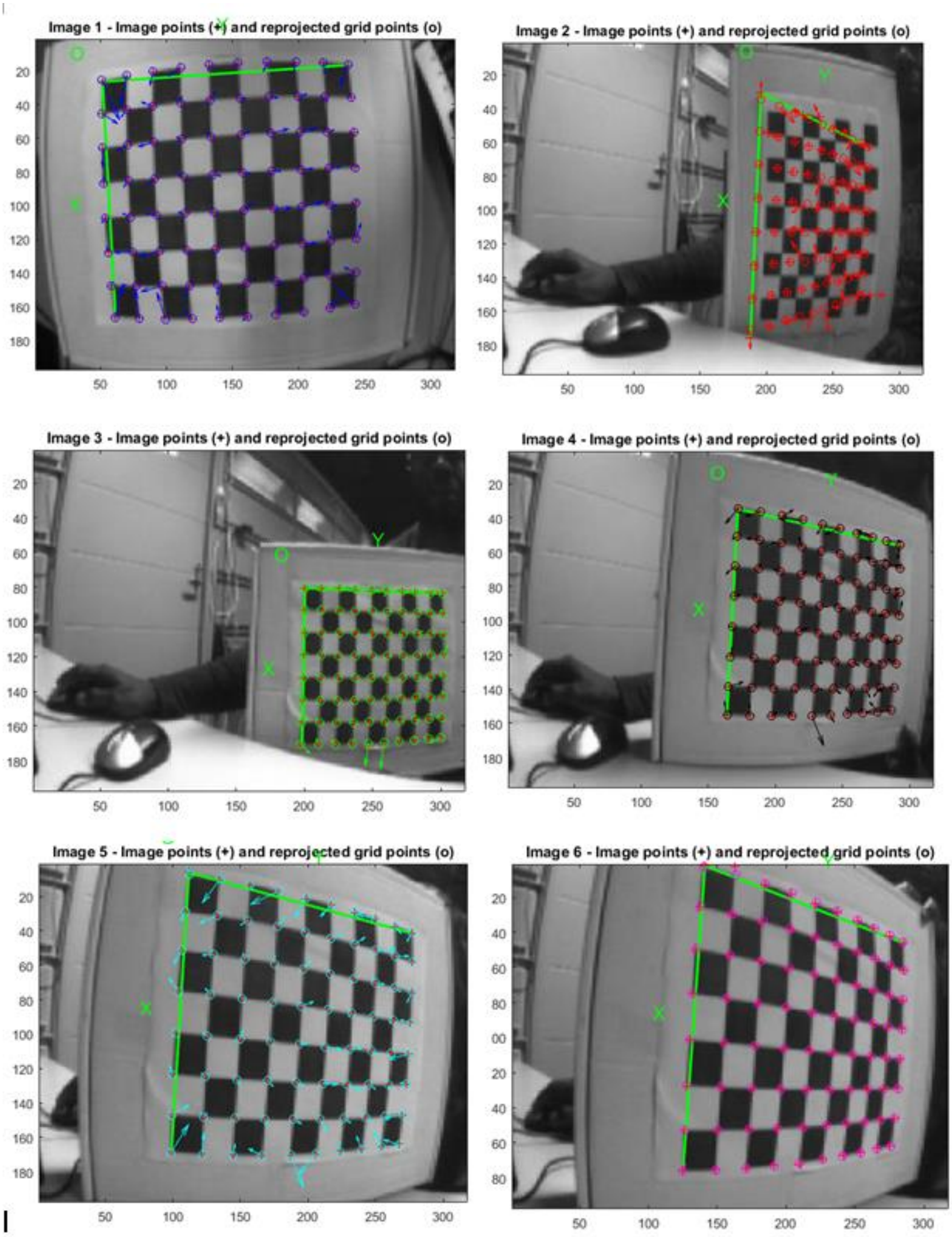


Image 1 - Image points (+) and reprojected grid points (o)

Image 2 - Image points (+) and reprojected grid points (o)

Image 3 - Image points (+) and reprojected grid points (o)

Image 4 - Image points (+) and reprojected grid points (o)

Image 5 - Image points (+) and reprojected grid points (o)

Image 6 - Image points (+) and reprojected grid points (o)

Figure 35: grid reprojection on the first six images

Click on `Analyze error` button in the `Calibration Toolbox Window` to view the new reprojection error (observed that the error is smaller than before) as illustrated in figure [36].

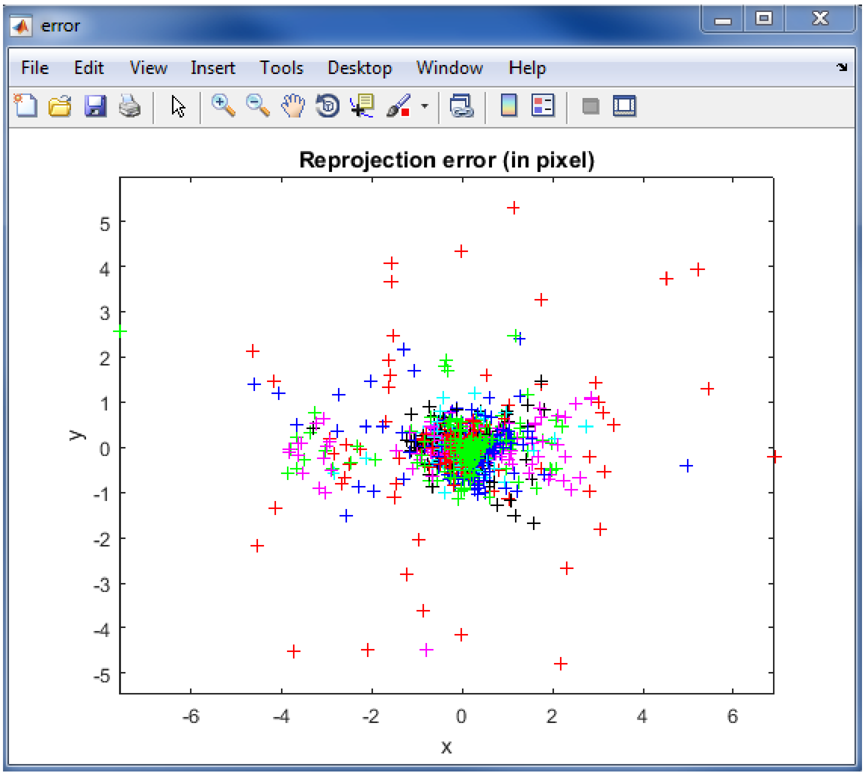

error
File Edit View Insert Tools Desktop Window Help
Reprojection error (in pixel)
5
4
3
2
1
0
-1
-2
-3
-4
-5
y
-6
-4
-2
0
2
4
6
x

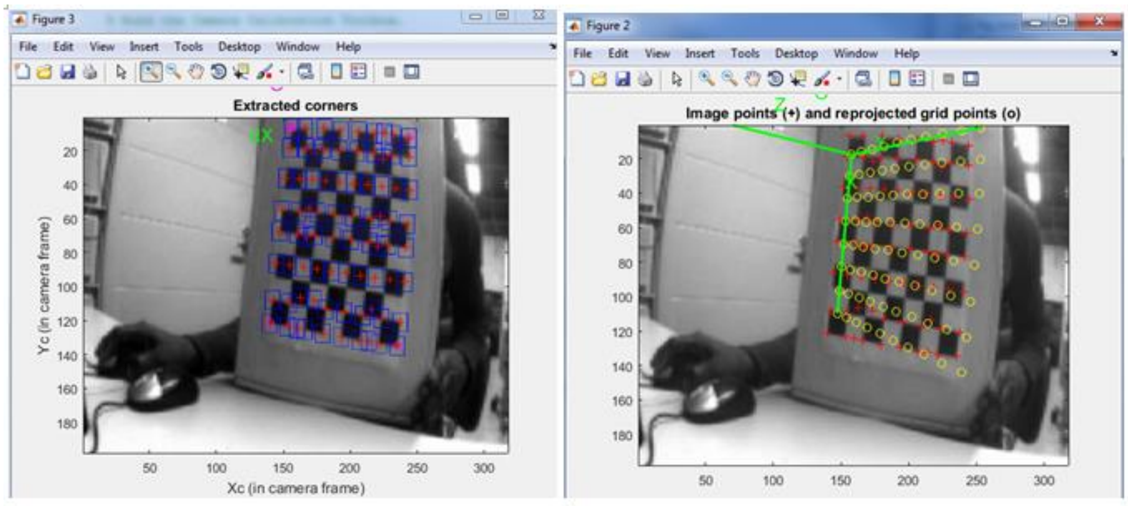
Figure 36: Snapshot, reprojection error after optimization

For the left camera calibration, repeat the steps as illustrated in the calibration procedure.

D.2. Computation of Extrinsic Parameters only

To compute extrinsic parameters, it is required additional image of the same calibration grid. Note: this image must be an image that has not be used in the main calibration procedure that is new image. The goal of this procedure is to compute the extrinsic parameters attached to this image given the intrinsic camera parameters previously computed /9//49/.

To do this, go to the `Calibration Toolbox Window` and click on `Comp Extrinsic` and then enter the image name (`image_extr`) (without an extension), then the image type (`jpeg`) and extract the grid corners (following the same procedure as previously presented, but remember that the first clicked point is the origin of that pattern reference frame). Then, the extrinsic parameter (3D location of the grid in the camera reference frame) is then computed.



The Extracted corner of the Extrinsic Calibration

projected grid point of the extrinsic calibration

Figure 37: Snapshot, Extracted corner (left image) and projected grid point (right image) of the extrinsic calibration

- In this project, the right camera extrinsic result information is showed below:

```
Extrinsic parameters:

Translation vector: Tc_ext = [ -17.187892          -156.666642          467.686318 ]
Rotation vector:   omc_ext = [ -1.478283          -1.755087    0.830990 ]
Rotation matrix:    Rc_ext = [ -0.116891           0.548837    -0.827716
                    0.988096          0.148201    -0.041271
                    0.100017         -0.822687    -0.559627 ]
Pixel error:           err = [ 4.28220          5.34302 ]
```

- The left camera extrinsic result information is shown below:

```
Extrinsic parameters:

Translation vector: Tc_ext = [ 43.019691          -195.971296          711.167992 ]
Rotation vector:   omc_ext = [ -1.729022          -2.306097    0.910322 ]
Rotation matrix:    Rc_ext = [ -0.340818           0.834019    -0.433884
0.905507    0.167121    -0.390036
-0.252786    -0.525816    -0.812168 ]
Pixel error:           err = [ 2.59343          2.87859 ]
```

D.3. Final Calibration Steps

Lastly it is needed to update the real camera data parameters with the calibrated result parameters. This can be done in the procedure as following /49/:

- At the end of each intrinsic calibration parameter, replace the calib_Results.m and Calib_Results.mat files in the specific tiy_calibrationcamera_calobration/left or right folder. Then at the after extrinsic calibrations, replace the data in the Calib_Results_extrinsic.m files.
- Replace also the config_camera.xml file in the tiy_calibration-camera_calibration folder with the file from the TIY folder. Run the make_camera_parameters.m script, to write the calibration data into the config_camera.xml file and copy the change configuration file back to the TIY folder