Detection of circular bounding box in video streams

Master Thesis

For

the fulfillment of the academic degree

M.Sc. in Automotive Software Engineering

Faculty of Computer Science

Professorship of Computer Engineering

February 2016

Submitted by : Md Abul Hasnat  
Matr. Nr: 306715

Supervisors : Prof. Dr. Wolfram Hardt  
Dipl.-Inf. Daniel Reißner
Declaration

I Md Abul Hasnat, Matric No. 306715, Professorship of Computer Engineering, Faculty of Computer Science, hereby declare that the work entitled “Detection of circular bounding box in video streams” is my original work. I have not copied from any other students’ work or from any other sources except where due reference or acknowledgement is made explicitly in the text, nor has any part been written for me by another person.

________________________
Date submitted

________________________
Signature of the student

Md Abul Hasnat (md.-abul.hasnat@s2012.tu-chemnitz.de)
Detection of circular bounding box in video streams
Master Thesis, Technische Universität Chemnitz, February 2016
Acknowledgment

At this point I would like to thank all the people who helped me to be success with this thesis. First I would like to thank Prof. Dr. Wolfram Hardt, Vice Dean, Faculty of Computer Science Chemnitz University of Technology, for giving me the opportunity to do this thesis in the University with this title “Detection of circular bounding box in video streams”. It was a unique experience for me to work with images. The topic was very interesting and I enjoyed it lot. It also helped me to get back my interest in image processing research and to have new idea for my future career. I also would like to thank my supervisor Dipl.-Inf. Daniel Reißner, Researcher in Chemnitz University of Technology for his useful ideas, suggestions and contribution. This research work wouldn't have completed on time without him. At the beginning of this thesis I did not know anything about OpenCV and Blender. He inspired me and helped me to grow interest for this topic. Finally, I have completed this thesis with his help. I would like to thank Dr. Ariane Haller, Faculty of Computer Science, who helped me to select such a good topic for the thesis.
# Table of Content

1 Introduction .......................................................................................................................... 8  
1.1 Motivation ........................................................................................................................ 8  
1.2 Outline of thesis ................................................................................................................ 10  
1.3 Problem statement ........................................................................................................... 12  

2 State of the art ...................................................................................................................... 13  
2.1 Digital image processing basics ........................................................................................ 13  
2.1.1 Image .......................................................................................................................... 13  
2.1.2 Image capturing .......................................................................................................... 16  
2.1.3 Data Stream Visualization .......................................................................................... 16  
2.1.4 Image Processing ........................................................................................................ 17  
2.1.5 Computer Vision ........................................................................................................ 17  
2.1.6 Codec .......................................................................................................................... 21  
2.1.7 IR based Proximity sensor in android .......................................................................... 29  
2.1.8 The robot .................................................................................................................... 31  
2.1.9 Image processing tools ............................................................................................... 35  
2.1.10 Blender (Modeling) .................................................................................................... 38  
2.2 Hough transform .............................................................................................................. 39  
2.3 Line detection .................................................................................................................. 40  
2.4 Circle detection ............................................................................................................... 42  
2.5 Bounding box Detection .................................................................................................. 47  

3 Concept ............................................................................................................................... 48  
3.1 Parameters to detect circular bounding box ..................................................................... 48  
3.1.1 Edge detection ............................................................................................................ 48  
3.1.2 Size of circles and bounding boxes .......................................................................... 49  
3.1.3 Distance between segment center to bounding box center ....................................... 50  
3.2 Hardware and Software development concept ................................................................... 51  
3.2.1 Hardware layer .......................................................................................................... 51  
3.2.2 Software layer ............................................................................................................ 52  

4 Realization and Implementation ........................................................................................... 59  
4.1: Realization of Hardware and Software .......................................................................... 59
4.2: Implementation .................................................................................................................. 62
  4.2.1: Implementation of Peripheral module ........................................................................ 63
  4.2.2: Implementation of Observation control module ...................................................... 65
  4.2.3: Implementation of Image processing pipeline module ............................................. 68
5  Result Analysis ..................................................................................................................... 82
6  Summary and outlook .......................................................................................................... 86
Appendix A .............................................................................................................................. 88
Appendix B .............................................................................................................................. 89
Bibliography ............................................................................................................................ 90
List of Figures:

Figure 1.1: Robot arm for industrial use ................................................................. 8
Figure 1.2: Sample robot arm .................................................................................. 11
Figure 1.3: Model robot arm ................................................................................... 11
Figure 2.1: Camera view of image in pixel values .................................................... 14
Figure 2.2: image co-ordinates convention for two dimensional image .................... 15
Figure 2.3: Pixel neighborhood ............................................................................. 15
Figure 2.4: The circular joint of robot arm is just a grid of numbers to a computer, ...... 19
Figure 2.5: The ill-posed nature of vision: the 2D appearance of objects can change radically with viewpoint .......................................................... 19
Figure 2.6: Encoding and Decoding(a) .................................................................... 26
Figure 2.7: Encoding and Decoding(b) .................................................................... 26
Figure 2.8: Decoder looking for specific Codec ...................................................... 27
Figure 2.9: Decoder looking for specific Codec, is not available ............................. 28
Figure 2.10: Proximity sensor ................................................................................ 31
Figure 2.11: Robot arm and its degree of freedom .................................................. 32
Figure 2.12: Stepper motor (a) (b) ......................................................................... 33
Figure 2.13: Stepper motor in Robot arm ............................................................... 34
Figure 2.14(a): OpenCV logo .............................................................................. 36
Figure 2.14(b): MATLAB SIMULINK logo ............................................................ 37
Figure 2.15: Blender logo ..................................................................................... 38
Figure 2.16: Side view of robot arm ...................................................................... 40
Figure 2.17: Edge pixels obtained from Sobel or gauss for a line ......................... 41
Figure 2.18: Three edge points considered to detect line ...................................... 41
Figure 2.19: The set of circles in $R^2$ plane ........................................................... 43
Figure 2.20: Mapping of circles with equal radius from $R^2$ to $R^3$ plane .......... 44
Figure 2.21: Mapping of circles touching origin $R^2$ to $R^3$ plane by keeping origin point constant ................................................................. 44
Figure 2.22: Mapping of circles touching origin $R^2$ to $R^3$ plane representing centers .......... 45
Figure 2.23: Mapping of three edge points to the parameter space and obtain intersection point of three right circular cone ......................................................... 45
Figure 2.24: Tangent direction at the edge point of a circle ................................... 46
Figure 2.25: Bounding Box external points of a region ........................................ 47
Figure 3.1: number of circles and segments .......................................................... 49
Figure 3.2: Circles and bounding boxes found in the image .................................. 50
Figure 3.3: Distance between connection line to bounding box center .................. 50
Figure 3.4: System architecture to detect circular bounding box ........................... 51
Figure 3.5: Colour space conversion (RGB to Intensity) ....................................... 54
Figure 3.6: Circle bounding segment .................................................................... 56
Figure 3.7: Image of real robot arm and model of a robot arm ............................... 58
Figure 4.1(a): System architecture for realization of Hardware and Software ............................................. 61
Figure 4.1(b): System architecture for implementation ................................................................................. 62
Figure 4.2: Sample image frames of robot arm .......................................................................................... 63
Figure 4.3: Sample image frames of robot arm (Segments are very close) ................................................. 64
Figure 4.4: Robot arm in two different positions ....................................................................................... 66
Figure 4.5(a): Sub-modules of image processing pipeline module of software layer ......................... 69
Figure 4.5(b): Flow chart to detect robot arm and its chance of collision .............................................. 70
Figure 4.6: False circles (red circles in image) detected by HoughCircles ................................................. 71
Figure 4.7: Circular joints detected by HoughCircles() ........................................................................... 73
Figure 4.8: Distance between segment circle centers ............................................................................... 75
Figure 4.9: Difference between function Rect boundingRect(a) and RotatedRect minAreaRect(b) ................................................................. 77
Figure 5.1: Robot arm detection (1) ......................................................................................................... 82
Figure 5.2: Robot arm detection (2) ......................................................................................................... 83
Figure 5.3: Robot arm collision detection (1) ............................................................................................. 84
Figure 5.4: Robot arm collision detection (2) ............................................................................................. 84
Figure 5.5: Robot arm collision detection (3) ............................................................................................. 85
1 Introduction

1.1 Motivation

The production line of industries are getting more efficient and having very high throughput. Different kinds of machineries are being used to make the production safe, fast, precise and reliable. Robot arm is such a machine which helps the production line to be more efficient and productive. Nowadays, many manufacturing industries are using robot-arms to get a competitive edge in manufacturing and can be outfitted for multiple applications like welding, material handling, thermal spraying, painting, drilling and so on. They are widely used to increase product quality and production demand and over all, to ensure safer, faster and efficient production. It is very important to control and maintain these machines very accurately. As a simple mistake of robot arm can cause excessive destructions and bring financial losses to the industries, the robot-arms must be very accurate when they are functioning in their production settings.

It is very important to control and monitor the robot arm with accuracy. Digital image processing can be very useful for this task. Monitoring with image processing is flexible to allow for alerts in case of changing programs in dynamic factory interactions. It allows for adaptation to scenario allowing new applications not possible before synchronization and considering safety issues. It will be easier in case we make an interface so that one can move into the simulation and generate program to do so on robot arm. It will be cheaper in one can connect with various robot arms in a factory from one computer and monitor them. Thus it will increase the efficiency. It can be done by capturing video of moving robot arm and checking for any Discrepancies. So many robot arms are being used in different production industries now.

Figure 1.1: Robot arm for industrial use [i]
One of these robot arms is shown in the **figure 1.1**. This type of robots can be used for different tasks like welding, material handling, thermal spraying, painting, drilling and so on. These robots are very strong and of different sizes. They need more space for their free movement in their industrial settings. So, it is very important that the Robot-arms are maneuvered, monitored and operated appropriately to maximize space utilization and prevent accidents and thus, it will bring financial gains to the companies. So it is very important to be very efficient and accurate to maintain the robot arms in the industries to utilize the space and prevent accidents. It will be very helpful for the industries to prevent financial loss.

In such cases, Computer Vision Techniques can be useful to monitor and maintain the robot arms. Computer Vision or Image Processing techniques have high tech process to monitor robot arms in runtime. The proper use of those techniques can inform or alert the maintenance personnel well ahead before an accident takes place and consequently prevent a mishap. So, there arises an immense need of developing a cost effective, user friendly and flawless Image Processing System to be installed in the operational features of the Robot-arms. In improvising such Digital Image Processing system, as good as human eyes, an automated system with very high computation capability is required where exact mathematical calculation of the observed image is needed. This system can be effectively used to develop the system in order to monitor and maintain robot arms suitably. Robot arm can be installed with fix program for it along with different sensors. Each of the movement of the robot arm has to be measured exactly to avoid any kind of collision. If there is any change or inaccuracy in measuring the movements, the program and sensors have to be rearranged. Moreover, the maintenance technician has to go to each of the Robot-arms exclusively to troubleshoot the problem, to change anything or to do any repair. Surely, all these are time consuming, require more workforce and cost sensitive. To resolve these problems, Digital Image Processing or Computer Vision can be a smart solution as it can give metaphorical eyes to the robots to function precisely in real-time situations. This intelligent sensing system can be used to detect locations, identify items and even to measure the dimensions of process-able targets.

There is a need to develop a system with the image processing techniques with low cost and user convenient. Tough human eyes are more powerful to detect most of the features from the image; an automated system with very high computation capability is required where exact mathematical calculation of the observed image is needed. Digital image processing can be useful to develop the system to monitor and maintain robot arms. Robot arm can be installed with fixed program for it along with different sensors. Each of the movement of the robot arm has to be measured exactly to avoid any kind of collision which is very difficult. If there is any
mistake or change, the program and sensors have to be re-arranged. A simple mistake can cause disaster to make great financial loss to the company. Again the maintenance person has to go to each robot arm individually if there is any problem or anything has to be changed. This is time consuming and requires more people. Digital image processing or computer vision can provide robots with eyes. The intelligent sensing system can be used to detect locations, identify items and even to measure the dimensions of process-able targets.

1.2 Outline of thesis

It is important to locate the segments of the Robot-arms and other objects in the industrial settings where they are moving. Then the distance between the segments of the Robot-arms and other objects are important as well. In this way, it is possible to identify if there is any chance of collision caused by the Robot-arm. As the problem of collision detection and prevention with the machine has become a significant problem, the main goal of this thesis is to design such a monitoring system to detect the Robot-arms, products are being made or other objects, and if there is any chance of collision among them.

One popular approach to collision avoidance is to model the robots workspace in terms of the robot-arm’s configuration space. Unfortunately, the computational complexity of this approach grows rapidly with the number of robot arms and the number of joints making up the robot arms. In addition, this approach is not suitable for dynamic real time environment. In another approach, a plan to perform a given task is generated by the planning system. To ensure that no collision occurs, the planned motions are sampled at discrete times and inter-object interference is checked for. A region octree is designed and constructed based on static objects in the environment. At each sample, an octree is constructed for each moving object and a set of static intersection test are performed by parallel traversal of the octree. Yu and Khalil present a system for collision detection of a robot working in a fixed environment. The robot and the environment are modeled by means of simple primitives (i.e. spheres, cylinders, parallelepipeds, cones and planes). The authors observe that in spite of the simple methods used for modeling, an online application based on testing the intersection of the robot link with all obstacles at each control is not practical. In order to accelerate the calculation of the collision detection algorithm, a table look-up procedure is used. Free space is represented by discretizing joint space and is stored in a table structure. This table is used to map the position of a robot link to the obstacles that lie close to the link. Thus the number of intersection tests performed at each sample is reduced [1].
By now, many different approaches have been presented for the general problem of collision detection. Some of them are used in conjunction with planning systems to decide if pre-planned robot motions are collision free. Some propose the use of custom hardware to efficiently detect collision. Moreover, some systems are targeted for off-line applications where system performance is not a major factor and operator interaction is possible. This thesis work desires to initiate a monitoring system which will provide real time response with little or no interaction with the operator and prior knowledge of robot arm movements is not required. Above all, this research endeavors on evolving a low cost effective and user friendly image processing tool.

There are several systems developed for detecting collision of robot arms. In this thesis, a robot arm shown in figure 1.2 for laboratory use is taken as a sample. This robot arm is available in the laboratory of Computer Engineering Department at Technische Universität Chemnitz. A model of this Robot-arm is made in Blender (Open source 3D computer graphics software) shown in figure 1.3.

![Figure 1.2: Sample robot arm](image1)

![Figure 1.3: Model robot arm](image2)

Image of this model Robot-arm is fed as input to the computer for analysis. Image can be taken from Blender or captured by webcam or any imaging device. As a single image is not sufficient for the analysis, this input is extended to video as an input. OpenCV (Open Source Computer Vision is a library of programming functions for computer vision) in C++ platform is used for image analysis. There are other programming platforms which can be used for the analysis.

A Robot-arm can be in any position while its images are being captured but the best option is to record while it is moving. The video can either be recorded and sent for analysis or make real time video input to the system. There can be other moving objects or products in the
environment. The output will be the detected robot arm and other objects. Distance between robot arm with other objects and the segments of the robot arm itself will be measured. The output will highlight a message if there is any chance of collision with given minimum distance.

1.3 Problem statement

Feature extraction is very important in digital image processing. There are many feature extraction techniques available in image processing. Some of these feature extraction techniques can be useful for detection of robot arm and other objects. Combination of different image processing algorithms is required to detect the robot arm, other objects and the required distances. Image processing is capable to analyze each of the pixels in each of the frames. The criterion upon which the detection process should be done is that some parameters should be selected beforehand. Then the hardware and software system based on these parameters should be designed. The system should be cost effective and user friendly.
2 State of the art

Image is very simple for human eyes but it is important to know how it is to computer. There are many image processing algorithms available now. Basic ideas of image processing and useful algorithms are discussed in this chapter. This is important for understanding technical details.

2.1 Digital image processing basics

2.1.1 Image

According to Oxford English dictionary an image is a representation of the external form of a person or thing in art. Image is a visible impression obtained by a camera, telescope, scanner or any other imaging device or displayed on a computer or video screen. Image is an optical appearance produced by light from an object reflected through lens.

“A digital image is a set of bits (zeroes and ones) that represents an image” [2]. Computer can process digital image as computer also use zeroes and ones as its language. Images are matrices of numbers to computer. Two main categories of digital images are Vector images and Raster images. Vector images are formed with mathematical equations like points, lines and polygons where Raster images are represented by pixels. “A Pixel is called a picture element, is the smallest measurable unit in raster images” [2]. The number of pixels are used to represent the image is called Resolution.

There are different types of images according to color:

1. Binary Image
   Only two colors are used for the image; one color represent ‘1’ and other one is for ‘0’.
2. Grayscale Image
   Grayscale image store the information of the intensity of light in it not the color. There are many ways to generate grayscale image. Such as 256 shades between ‘0’ (Total absences of light) to ‘1’ (Total presence of light).
3. True color
   True color mans the natural color images. There are different color spaces like RGB (Red, Green, Blue), CMYK (Cyan, Magenta, Yellow and Black).

Digital images can be stored in different file formats. Some popular formats are given below:

1. JPG or JPEG (Joint Photographic Experts Group)
   It supports color depth of 24 bits (3 color channels of 8 bits each).
2. GIF (Graphic Interchange Format)
It supports color depth of 8 bits only.

3. PNG (Portable Network Graphics)
   It supports color depth of 48 bits (3 color channels of 16 bits each).

4. TIFF (Tagged Image File Format)
   It supports color depth of 48 bits (3 color channels of 16 bits each).

5. OpenEXR
   This is an open standard.

![Figure 2.1: Camera view of image in pixel values](image)

**Figure 2.1** shows how the camera sees the part (segment joint) of a robot arm. The numeric values in the table rows and columns are representing pixels. Compare to human eyes image processing techniques can observe the minute parts where human eyes can observe the big parts.

An image is captured by the imaging devices like camera, scanner. The image is then sent to the computer or any other processor for processing. This image is represented as a two dimensional array or matrix using a pair of co-ordinates (X,Y). The pixel represented by these co-ordinates depends upon the convention followed. Here, X represents horizontal axis and Y is vertical axis as shown in figure 2.2. Each pixel location \((i,j)\) contains the intensity value.
Figure 2.2 shows the co-ordinate convention followed to display digital image. X axis is representing height of an image and Y axis is representing width of an image. Image quality improves when the image processing techniques can analyze higher number of pixels of an image. The resolution of an image can be measured in two ways. Spatial resolution is measure of number of lines per millimeter or the number of dots per inch which are visible to human eye. Intensity resolution is the measure of smallest possible distinguishable intensity change of an image[6]. Generally it is $2^8 = 256$. Intensity resolution increases with an increase intensity range. As in figure 6 each pixel can presented by (X,Y) co-ordinate system.

Figure 2.3 shows how to manipulate the neighbors of a desired pixel in an image. Pixel P(X,Y) can have three type of neighbors. They are four, eight or diagonal neighborhood. Using these neighborhoods image processing can manipulate any range within an image.


2.1.2 Image capturing
Image Capture is the use of an imaging device, such as Digital Camera to create a digital representation of an image. Then the image can be stored in memory for further processing.

An image must be captured by a camera and converted to a manageable entity before image processing. This process is called image acquisition. There are three steps in image acquisition:

1. Energy reflected from the object of interest
2. An optical system which focuses the energy
3. Sensor which measures the amount of energy.

Some kind of energy is needed to illuminate the scene to capture the image. Most often visual lights are used as the energy.

Image sensor is used to record the image after capturing the image. The sensor consists of a 2D array where each of the array elements represents a pixel. The pixel is capable to measure the amount incident light and convert into voltage. Then the voltage is converted into digital numbers. The digital number is bigger when the voltage is higher and the voltage is higher when the incident light is higher. All the elements in the array must be empty before a camera can capture an image. When the camera is going to capture an image the light is allowed to enter and charges start accumulating the cells. The accumulated charges are converted to digital form by analog to digital converter. Each cell content is converted into a pixel value in the range of 0 – 255 to transform the information from the sensor into image. Such a value is called Intensity of a pixel which interpret the amount of light hitting a cell during the exposure time.

Video stream is the sequence of still images. The number of still pictures per unit of time of video is Frame rate. At least sixteen frame per second is required for comfortable illusion of a moving image. Streaming video is a one way transmission over a data network.

2.1.3 Data Stream Visualization
Data Stream is uninterrupted flow of long sequence of data like video files. Visualization is a process to visualize the data to the user in a meaningful format from the data source. Streaming is a technique where a receiver is allowed to process some data while the transfer is not completed by the sender. The data has been increasing at a very high rate and it becomes impossible to process the whole data in main memory due to size limitation. A good way to improve this situation is to apply streaming to the visualization. The main idea is to exploit data
locality in relationships to produce constant and continuous streams of data flowing through the visualization process. Data dependency, Stream re-arranging, progressive algorithms etc. should be considered here. Almost invariably a localized portion of data is used in visualization during data analysis, cleaning, filtering, attribute selection etc.

There are some challenges for structuring the visualization pipeline for streaming. It is important to arrange the data in order to send the data to the next stage. Data dependency has to be analyzed and solved to process related data items together. It is very important to have the required data before the stage’s process begins to reduce amount of memory required. The requirement for progressive algorithms that could perform operations step by step over the data set.

In data stream visualization it is necessary to have all related items available to start processing of each step. Relationships between items have to be analyzed to identify subsets of information able to be processed locally. Thus data dependency is identified. The data flow between processing steps has to be arranged in such a way that related items be transferred together from one step to the next. Having all the data items together allows the beginning of their processing and the amount of storage for pending items can be consequently reduced. [5]

### 2.1.4 Image Processing

Image Processing is the analysis and manipulation of a digitized image, especially in order to improve its quality. The main objective is to make the image meaningful or to achieve the required information from it. ‘Image Processing is a technique to enhance raw images received from cameras/sensors placed on satellites, space probes and aircrafts, Cars or pictures taken in normal day-to-day life for various applications’[3]. Image processing includes improvement in image appearance and efficient representation. So the feature extraction, analysis and recognition of images, filtering, enhancement and restoration can be done. ‘’The principle advantage of Digital Image Processing methods is its versatility, repeatability and the preservation of original data precision’’ [3].

### 2.1.5 Computer Vision

The transformation of data from a still or video camera into either a decision or a new representation is widely known as Computer Vision [CV]. The transformations are performed to attain some particular objectives. The input data may be comprised of some contextual
information such as “the camera is mounted on an industrial robot arm” or “laser range finder indicates an object is 1 meter away from the robot arm”. The robot arm might decide that “there is a human in the scene” or “there are some balls in the scene”. A new representation of the image might mean converting a color image into a grayscale image or taking away camera motion from an image sequence.

As human beings, we might generally think that vision and vision related features are very easy, but in reality they are very complex. On the other hand, Computer Vision in terms of machines are far more sophisticated and complicated. The human brain segregates the vision signal into many channels that stream diverse kinds of information into our brain. For us, our brains have attention systems that identify, in a task-dependent way, important parts of an image to examine while suppressing examination of other areas. Although there is massive feedback in the visual stream, as yet, it is little understood. There are extensive associative inputs from muscle control sensors and all of the other senses that let the brain to draw on cross-associations made from years of living in the world. The response loops in the brain go back to all phases of processing including the hardware sensors themselves (the eyes), which mechanically control lighting via the iris and adjust the reception on the surface of the retina.

In a machine vision system, a computer receives a grid or matrix of numbers from the imaging device or from disk, and that’s it. Each of the numbers represent a pixel, smallest unit of the image. For the most part, there is no built-in pattern recognition, no automatic control of focus and aperture, no cross-associations with years of experience. For the most part, vision systems are still fairly native. **Figure 2.4** shows a picture of robot arm. In that picture we see three joints of robot arm. What the computer “sees” is just a grid of numbers. Any given number within that grid has a rather large noise component and so by itself gives us little information, but this grid of numbers is all the computer “sees”. Our task then becomes to turn this noisy grid of numbers into the perception: “joints”. **Figure 2.5** gives some more insight into why computer vision is so hard. **Figure 2.5** shows how the view point can change the vision. Different viewpoints can generate different images.
In fact, the problem, as we have posed it thus far, is worse than hard; it is formally impossible to solve. Given a two-dimensional (2D) view of a 3D world, there is no unique way to reconstruct the 3D signal. Formally, such an ill-posed problem has no unique or definitive solution. The same 2D image could represent any of an infinite combination of 3D scenes, even if the data were perfect. However, the data is corrupted by noise and distortions.

Such corruptions can be caused by imperfections in the lens and mechanical setup, finite integration time on the sensor (motion blur), electrical noise in the sensor or other electronics, compression artifacts after image capture and from variations in the world (weather, lighting, reflections, movements).

In designing a practical system, advanced contextual knowledge can often be used to work around the obligatory limitations caused by visual sensors. Consider the example of a mobile
robot that must find and pick up staplers in a building. The robot might use the facts and figures that a desk is an item found inside offices and that staplers are mostly found on desks. This gives an imbedded size reference; staplers must be able to fit on desks. It also helps to disregard falsely “recognizing” staplers in unusual places (e.g., on the ceiling or at a window). In cases like this, as an advertising blimp lacks the prerequisite wood-grained background of a desk, the robot can certainly ignore a 200-foot advertising blimp shaped like a stapler. In contrast, with tasks such as image retrieval, all stapler images in a database may be of real staplers and so large sizes and other unusual configurations may have been implicitly excluded by the assumptions of those who captured the photographs. That is, the photographers possibly took pictures only of real, normal-sized staplers. In general, people have a tendency to center objects when taking pictures and tend to put them in characteristic orientations. So consequently, there is often quite a bit of unintended implicit information within photos taken by the people.

With these of machine learning techniques, contextual information can also be modeled and demonstrated explicitly. Concealed variables such as orientation to gravity, size and so on can then be associated and interrelated with their values in a labeled training set. In another way, one may attempt to measure hidden bias variables by using additional sensors. The use of a laser range detector to measure depth allows us to accurately scale the size of an object.

Another vital challenge for Computer Vision is noise. Typically, we deal with noise by using statistical methods. For example, it may be unmanageable to detect an edge in an image merely by relating or associating a point to its immediate neighbors. But if we look at the statistics over a local region, edge detection is much easier. In edge detection, a real edge should appear as a string of such immediate neighbor responses over a local region, each of whose alignment or orientation is consistent with its neighbors. Although it is also manageable to compensate noise by taking statistics over time, still other techniques account for noise or distortions by building explicit models learned directly from the available data. For example, as lens distortions are well understood so far, one requires only to learn the parameters for a simple polynomial model in order to define or describe and thus correct almost completely such distortions.

The decisions or actions that Computer Vision attempts to make counting or calculating on camera data are executed in the context of a specific purpose or task. We may want to exclude noise or damage from an image in order to ensure that our security system will issue an alert if someone tries to intrude or climb a fence. Or else, because we need a monitoring system that is capable of counting how many people cross through an area in an amusement park or a supermarket. Definitely, Vision software for robots that wander through office buildings will adopt different strategies than vision software for stationary security cameras because the two systems
have considerably different contexts and objectives. As a general understanding: the more constrained a computer vision context is, the more we can depend on those constraints to simplify the problem and the more reliable our final solution will be. [13]

2.1.6 Codec

The term codec is derived from the terms encoding/decoding and compression/decompression. According to the dictionary, encoding is defined as “The conversion of information into code”. Codec is defined as “The symbolic arrangement of data or instructions in a computer program or set of such instructions” in Computer science. A more general definition of code is a “System used for transmitting messages requiring brevity or secrecy” [14]. A video codec is a device or software that enables compression or decompression of digital video; the format of the compressed data adheres to a video compression specification. The compression is usually lossy. Historically, video was stored as an analog signal on magnetic tape.

As early as 1929, Ray Davis Kell described a form of video compression and was granted a patent for it. He wrote, “It has been customary in the past to transmit successive complete images of the transmitted picture. In accordance with this invention, this difficulty is avoided by transmitting only the difference between successive images of the object.” The first video systems evolved from oscilloscopes. They were essentially oscilloscopes with a second dimension, able to display multiple lines on the screen. The systems were analog and used cathode ray tubes (CRTs). Engineers soon realized that this CRT-based display device could present a signal modulated on a transmitted radio frequency carrier. Thus television was born. A notable early form of compression was interlacing, where fields of even lines and odd lines are transmitted alternately. Interlacing is useful in two ways: it can either halve the required bandwidth or double the vertical resolution. Video was first recorded on open-reel magnetic tape by the BBC in 1955. Consumer-friendly cassettes were standardized in the 1970s and adopted in home entertainment in the 1980s. Digital video processing has been developed to handle each of the techniques of broadcast, interlacing, and recording as well as Mr. Kell’s technique of predictive compression. So many standards for digital video compression are in wide use today that television and digital video device makers find it necessary and preferable to use programmable digital video processors. [15]

In 1984, the International Telecommunication Union (ITU) standardized recommendation H.120, the first digital video coding technology standard. H.120 used differential pulse code modulation (DPCM), scalar quantization and variable-length coding techniques to transmit NTSC or PAL video over dedicated point-to-point data communications lines. Practical digital video compression started with the ITU H.261 standard in 1990. The target was to transmit video over
ISDN lines, with multiples of 64 kbit/s data rates and CIF (352x288-pixel) or QCIF (176x144-pixel) resolution. The standard was a pioneering effort and used a hybrid video coding scheme that is still the basis for many video coding standards today. JPEG is a widely used method for compression of still images (photographs), standardized in 1992. JPEG stands for Joint Photographic Experts Group. The standard is widely used in digital still cameras, mobile phones and the World Wide Web. Motion JPEG encodes video data as a sequence of independently coded JPEG images. The MPEG-1 standard (1992) was designed to achieve acceptable video quality at 1.5Mbit/s and 352x288/240-pixel resolution. MPEG-1 is playable in almost all PCs, VCD players and DVD players. The MPEG-2 / H.262 standard was jointly developed by the ISO and ITU standardization organizations and approved in 1993. The standard supports standard definition (SD, 720x576/480-pixel) resolutions as well as high-definition (HD) video signals with a pixel resolution of 1920x1080. The DV video coding standard developed by the IEC and standardized in 1994 targets primarily camcorders that store the video bitstream data on tape. DV does not use motion compensation, but instead encodes the individual frames at a fixed bitrate of 25Mbit/s. Combined with the audio and error detection and correction data, this results in a very high 36Mbit/s data rate. DV efficiency is better than Motion JPEG and is similar to I-frameonly MPEG-2 / H.262. The H.263 standard (1995) developed by the ITU was a big step forward and is today’s dominant video conferencing and cell phone codec. The primary design target was video conferencing at low bit rates for mobile wireless communications. Especially for progressive video, H.263 quality is superior to all prior standards at all bit rates. RealNetworks was one of the first successful commercial companies to sell tools for streaming digital audio and video over the Internet. Version 1 of the RealVideo codec was introduced in 1997. This first version was based on the H.263 codec but versions 8 and higher use a proprietary video codec. Version 10 was introduced in 2004 and has been in use since. MPEG-4 standardization began in 1995 and has continually been enhanced with new profiles including many novel coding concepts such as interactive graphics, object and shape coding, wavelet-based still image coding, face modeling, scalable coding and 3D graphics. Very few of these techniques have found their way into commercial products. Later standardization efforts have focused more narrowly on compression of regular video sequences. The MPEG-4 video standard has been designed to allow a wide range of compression quality vs bit rate trade-offs. The MPEG-4 Simple Profile is very similar to H.263. On2 Technologies is a publicly traded company that designs proprietary video codec technology, which it licenses to its customers. The On2 codec designs are known as VP3 through VP7. In 2004, Macromedia (now Adobe) selected On2's VP6 for use in the Flash 8 video codec. Adobe’s Flash 4 Video is frequently used as the vehicle for presenting video in web pages. The popular YouTube and Google Video websites use this technology. Ebay’s Skype has licensed the On2 video codec for video conferencing over IP, and XM has licensed On2’s coding
standard for satellite broadcasts. On2 claims its technology carries no third-party patent licensing fees. Microsoft developed the Windows Media Video version 9 video codec. The codec was initially proprietary, but was later standardized by the SMPTE organization. SMPTE announced the formal release of the SMPTE 421M video codec standard in 2006. VC-1 is the standard’s informal name. WMV9 / VC-1 is a hybrid video codec that uses conventional motion compensation, the DCT transform and Huffman coding, similar to H.263 and MPEG-4. However, WMV9 / VC-1 uses 4x4-pixel blocks, which are smaller than the 8x8-pixel blocks found in previous standards. VC-1 is characterized as an alternative to the latest ITU-T and MPEG video codec standard known as H.264/MPEG-4 AVC; however, studies have shown VC-1 to compress less well than H.264. The codec is computationally less demanding than H.264 and has been adopted by both HD DVD and the Blu-ray Disc Association as a mandatory video standard for players and optional codec for video disc sellers. The codec is also frequently used on the Internet and in the Xbox 360 gaming console Today’s high-end PCs can decode high-definition WMV9 / VC-1 bitstreams in real time in software. Where the original MPEG-4 standardization committees put a lot of effort into novel media coding techniques, H.264 took a “back to basics” approach. The goal was to compress video at twice the rate of previous video standards while retaining the same picture quality. The standard was originally known as H.26L or JVT, for the Joint Video Team, in which the ISO and ITU organizations worked together to complete the standardization. H.264 is the ITU name for the standard; MPEG-4 part 10, Advanced Video Coding (AVC) is the ISO name. Due to its improved compression quality, H.264 is quickly becoming the leading standard; it has been adopted in many video coding applications such as the iPod and the Playstation Portable, as well as in TV broadcasting standards such as DVB-H and DMB. Portable applications primarily use the Baseline Profile up to SD resolutions, while high-end video coding applications such as set-top boxes, Blu-ray and HD-DVD use the Main or High Profile at HD resolutions. The Baseline Profile does not support interlaced content; the higher profiles do. The government of the People’s Republic of China initiated development of the Audio Video Standard, or AVS. The compression codec for digital audio and video was standardized in 2005. One of the key factors driving development was to remove the need for Chinese consumer electronics manufacturers to pay royalties to companies that hold patents on internationally standardized compression technology. AVS coding efficiency is comparable to H.264, while its computational complexity is lower. The AVS standard will likely be used in some set-top box applications for IPTV and mobile broadcast TV within mainland China. Major semiconductor companies have announced support for the standard, but today it’s unclear whether AVS will succeed in becoming a widely adopted video coding standard. [15]
CODEC is a Java package for encoding and decoding data structures defined by the Abstract Syntax Notation One (ASN.1), the standard notation of the Open System Interconnection (OSI) to specify any kind of messages or data structures. The name of the package is an abbreviation for COoder/DECoder, which represents the main part of CODEC. Besides there are Java classes representing some complex data structures according to the family of Public Key Cryptography Standards (PKCS). CODEC is written in Java to make it easier to run the same code on different platforms and is distributed as open source to allow as many developers as possible to use it without having to buy expensive APIs. [16]

CODEC was developed as part of the SeMoA project (Secure Mobile Agents, http://www.semoa.org) at the Fraunhofer Institute for Computer Graphics. In this project was a mobile agent platform developed with a focus on security that resulted in the need for strong cryptographic functions. The first version of CODEC was ready at the end of 1999 as a well modularized and structured package. The CDC department of the Technical University of Darmstadt contributed to CODEC by implementing CODEC packages for X.509 and PKCS#12 and there is an ongoing collaboration for maintenance, enhancement and extension of CODEC. CODEC is distributed as open source software to make it available to as many users as possible. [16]

Compression basically means reducing image data. A digitized analog video sequence can comprise of up to 165 Mbps of data. To reduce the media overheads for distributing these sequences, techniques are commonly employed to achieve desirable reductions in image data. Compression reduces color nuances within the image. It reduces the color resolution with respect to the prevailing light intensity. It can remove small and invisible parts of the picture. Compression compares adjacent images and removes details that are unchanged between two images. The first three are image based compression techniques, where only one frame is evaluated and compressed at a time. The last one is video compression technique where different adjacent frames are compared as a way to further reduced the image data. All of these techniques are based on an accurate understanding of how the human brain and eyes work together to form a complex visual system. As a result of these subtle reductions, a significant reduction in the resultant file size for the image sequences is achievable with little or no adverse effect in their visual quality. The extent, to which these image modifications are humanly visible, is typically dependent upon the degree to which the chosen compression technique is used. Often 50% to 90% compression can be achieved with no visible difference, and in some scenarios even beyond 95%. [17]
There are two basic categories of compression; lossless and lossy. Lossless compression is a class of algorithms that will allow for the exact original data to be reconstructed from the compressed data. That means that a limited amount of techniques are made available for the data reduction, and the result is limited reduction of data. GIF is an example of lossless images compression, but is because of its limited abilities not relevant in video surveillance. Lossy compression on the contrary means that through the compression data is reduced to an extent where the original information cannot be obtained when the video is decompressed. The difference is called the artifacts. [17]

Compression involves one or several mathematical algorithms that remove image data. When the video is to be viewed other algorithms are applied to interpret the data and view it on the monitor. Those steps will take a certain amount of time. That delay is called compression latency. The more advanced compression algorithm, the higher the latency. When using video compression and several adjacent frames are being compared in the compression algorithm, more latency is introduced. For some applications, like compression of studio movies, compression latency is irrelevant since the video is not watched live. In surveillance and security using live monitoring, especially when PTZ (Pan/Tilt/Zoom) and dome cameras are being used, low latency is essential. [17]

**How codec works:** An encoder is responsible for packaging data according to certain rules. For instance a digital video stream may consist of samples of luma (represented here by ‘y’) and chroma (represented here by ‘u’ and ‘v’) which can be packaged inside a wrapper in variable formations, such as ordering the code as ‘yuvyvyuv’ as opposed to ‘yyyuuuvvv’. The set of rules which defines how the data is encoded and packaged is what we refer to when we speak about a ‘codec’, and each codec has a name and an identifier. This identifier consists of four characters, such as ‘yuv2’, and thus is called a four character code (FourCC). ‘4cc1’ and ‘4cc2’ are used at the left to represent four character codes for hypothetical codecs. This 4cc is embedded in with the packaged data, either as part of the essence (audio and video) or as part of the container (file or wrapper). In **Figure 2.6 and 2.7**, we see that the same video stream can be encoded in a different order by the differently defined codecs. This is not a problem as long as the same codec is used to decode the data and unpacks it in the correct order.
As the bottom decode in each diagram shows, it is possible for a different codec to access and render the video essence (it, too, understands Y, U, V) and then decode it in the wrong order. Think of the human ‘encoder’ placing a code before encoding English text saying ‘EN-US’ to
identify it as English US so that the information receiver knows an English ‘decoder’ is needed to understand it, but also that ‘color’ is not spelled with a ‘u’ or that ‘football’ does not mean soccer. In both cases, the essence of the data is decoded, but it is not rendered faithfully to the original. It is okay to use a different tool to encode and decode the same data (QuickTime/Windows Media, American/British English speaker) as long as the same codec is employed to instruct the tool on proper rendering.

Compression and encoding have been used on our video content for years in the form of interlace scanning and the conversion of RGB to YUV for transmission, processing and storage. Just as we need the correct, well-maintained tools to encode/decode this information in order to properly access video, we need to understand codecs and be well aware of their associated best practices for use in order to preserve and maintain access to digital materials.

When a computer decoder is presented with data one of the things it looks for is the four character code. It then looks to see if it has the associated specification or codec available for application. Figure 2.8 shows the decoder is looking for the system codec 4cc2 in the system codec library to decode the information.

![Figure 2.8: Decoder looking for specific Codec](image)

If software/hardware does possess the required codec it follows the associated rules to unpack or decode the data. If it doesn’t you either get nothing or a not-so-helpful message telling you the system doesn’t have the required codec and can’t play the file back. Figure 2.9 shows the system codec 4cc1 is not available in the library. So, it is showing the system message.[14]
Many video coding standards have been developed and are in use today. Which video codecs to support in a particular device depends on the exact application, but tomorrow’s connected and open devices must all be able to decode many standards and simply play any bitstream. The following issues arise during the design of the video subsystem:

**Coding standard:** The subsystem must be able to encode or decode the applicable video coding standards. There are many standards, and within each standard there are often variations (different versions and profiles).

**File format:** video coding bitstreams are encapsulated in different file formats, which the video subsystem must be able to read and decode.

**Image resolution:** video bitstreams contain compressed images with a particular display resolution. Standard definition TV signals for instance have 720 pixels per line, while portable media player displays often have 320 pixels per line.

**Frame-rate:** the number of frames per second stored in the bitstream differs per application. Low-bandwidth wireless video conferencing for instance often uses 15 frames per second, while good quality video requires 30 frames per second.

**Bitrate:** MPEG-2 based DVDs use up to 10Mbit/s while H.264 standard definition material is often stored at 1.5Mbit/s. Higher bitrates typically require more processing power to decode.

**Latency:** some applications, like video conferencing, require very low latency transmission. A video recording application can have a much longer latency before the compressed images are stored in the memory system.
**Pre/post processing:** before or after the coding of the video, high quality algorithms for image filtering, scaling and frame rate conversion can greatly improve the quality of the images presented on the display.

**Processing power:** different video standards and processing techniques require different amounts of computational resources. The video subsystem should be capable of supporting the most demanding use case. [15]

### 2.1.7 InfraRed(IR) based Proximity sensor in android

"Proximity Sensor" includes all sensors that perform non-contact detection in comparison to sensors, such as limit switches, that detect objects by physically contacting them. Proximity Sensors convert information on the movement or presence of an object into an electrical signal. The proximity sensor is common on most smart-phones, the ones that have a touchscreen. This is because the primary function of a proximity sensor is to disable accidental touch events. The most common scenario being- The ear coming in contact with the screen and generating touch events, while on a call. To solve the "I didn't take that stupid picture, my ear did" issue, device manufacturers came up with the idea of placing a proximity sensor close to the speaker, which will then detect any object in the vicinity of the speaker. If any object is present (for example, user's ear), then the touch events can be assumed to be accidental and ignored. Now there are various technologies for proximity sensing:

- Electrical (Inductive, Capacitive)
- Optical. (IR, Laser)
- Magnetic.
- Sonar.

Of all these, the most non-intrusive and low-cost modules are the optical proximity sensors. These can detect bodies in the vicinity of the device up to 5cm. This is perfect for use on smartphones. Coming to Sensors on Android-Gingerbread, the proximity sensor is often implemented using a light sensor chip. Common ones are ISL29003/23 & GP2A by Intersil & Sharp respectively. Both these sensor-chips are primarily active light sensors, which provide the ambient light intensity in LUX units. The distance value is measured in centimeters. Note that some proximity sensors only support a binary "close" or "far" measurement. In this case, the sensor should report its max Range value in the "far" state and a value less than max Range in the "near" state.
For example, device uses a GP2A chip placed close to the speaker facing the user. The normal light response of the GP2A is such that it triggers a proximity-detect interrupt at a distance of approximately 5cms in normal lighting. This means that when the user receives a call and brings the phone closer to his ear, the ambient-light around the light/proximity-sensor slowly drops down below the threshold & the sensor detects this and switches the state from FAR to NEAR. This event can be detected by any android application which has registered a SensorEventListener.

In most Android phones, the proximity sensor is implemented as a boolean-sensor. Its returns just two values "NEAR" & "FAR". Thresholding is done on the LUX value i.e. the LUX value of the light sensor is compared with a threshold. A LUX-value more than threshold means the proximity sensor returns "FAR". Anything less than the threshold value and the sensor returns "NEAR". The actual value of threshold is custom-defined depending on the sensor-chip in use and its light-response, coupled with the location & orientation of the chip on the smart-phone body. The proximity sensor is implemented very much like the other sensors except on one critical point - Proximity sensor is interrupt-based (not Poll-based). All the other sensors are polled for at regular intervals, selected by the DELAY parameter used while registering a SensorEventListener. Proximity Sensor is interrupt based. This means that we get a proximity event only when the proximity changes (either NEAR to FAR or FAR to NEAR).[35]

Now a day, touchscreens have become widespread and are progressively replacing mechanical buttons in applications where an interactive, easy-to-use interface is desirable. In the automotive sector, these applications include center stack radio consoles, navigation systems and backseat entertainment units. While it is becoming increasingly natural for people to interact with a system via touch screen, there is still considerable advancements that can be made to make touch interfaces even more intuitive, comfortable and safe in an automotive environment. This application requires that the driver’s attention be focused on the road with minimal interaction and distraction. Proximity detection allows the system to know when the user wants (or doesn’t want) to interact with the screen. This information can then be used to control LCD backlight intensity, system wake-up, and button/menu visibility thereby. Today, mobile phones use IR-based proximity sensors to detect the presence of a human ear. This sensing is done for two purposes: Reduce display power consumption by turning off the LCD backlight and to disable the touch screen to avoid inadvertent touches by the cheek. [19]

Any proximity sensor has two essential part, first one is Transmitter and the other one is Receiver. The Transmitter emits a typical wave e.g. Infrared (IR), ultrasonic, supersonic. Range of this wave is varied according to the transmitter. If any obstacle comes in the transmitted beam,
some part of the wave is reflected back from the obstacle and it is received by the receiver and the receiver generates some indicating signal according to that. This functionality is shown in figure 2.10.

![Proximity sensor diagram](image)

**Figure 2.10: Proximity sensor [ii]**

Proximity sensors have different features. Proximity Sensors detect an object without touching it, and they therefore do not cause abrasion or damage to the object. No contacts are used for output, so the Sensor has a longer service life (excluding sensors that use magnets). Unlike optical detection methods, Proximity Sensors are suitable for use in locations where water or oil is used. Proximity Sensors provide high-speed response, compared with switches that require physical contact. Proximity Sensors can be used in a wide temperature range. Proximity Sensors are not affected by colors. [20]

### 2.1.8 The robot

In 1973, world’s first industrial robot with six electromechanically driven axes, known as FAMULUS was made by KUKA AG, a German company. An industrial robot is defined as an automatically controlled, reprogrammable, multipurpose manipulator programmable in three or more axes by ISO 8373.

The robot arm in **figure 2.11(a)**, is available for academic use. **Figure 2.11(b)** shows this robot has five degrees of freedom for positioning in space and a degree of freedom for the opening width of the gripper. The axes are the following are numbered from zero to five, zero, what the degree of freedom for the Rotation of the entire robot around the vertical axis describes to five, which the opening represents the gripper. The axis is zero, one, two and five respectively with a own independent motor driven. The axes three and four share two motors. A motion to axis three
is realized by two motors at the same speed rotate in the same direction. For the motion to be four axis have the two Engines opposite move. The motors on the robot are equipped with sensors, which enable the adjusting angle of engines with a high accuracy capture. [21]

Stepper motor

A stepper motor is an electromechanical device which converts electrical pulses into discrete mechanical movements. The shaft or spindle of a stepper motor rotates in discrete step increments when electrical command pulses are applied to it in the proper sequence. The motors rotation has several direct relationships to these applied input pulses. The sequence of the applied pulses is directly related to the direction of motor shafts rotation. The speed of the motor shafts rotation is directly related to the frequency of the input pulses and the length of rotation is directly related to the number of input pulses applied. A stepper motor and its inside view is shown in figure 2.12.
There are several advantages of stepper motor. The rotation angle of the motor is proportional to the input pulse. The motor has full torque at standstill (if the windings are energized). Precise positioning and repeatability of movement since good stepper motors have an accuracy of 3 – 5% of a step and this error is non cumulative from one step to the next. The motor has an excellent response to starting/stopping/reversing. Stepper motor is very reliable since there are no contact brushes in the motor. Therefore the life of the motor is simply dependant on the life of the bearing. The motors response to digital input pulses provides open-loop control, making the motor simpler and less costly to control. It is possible to achieve very low speed synchronous rotation with a load that is directly coupled to the shaft. A wide range of rotational speeds can be realized as the speed is proportional to the frequency of the input pulses.

Stepper motor has some limitations also. Low Efficiency is one of the limitations. Unlike DC motors, stepper motor current consumption is independent of load. They draw the most current when they are doing no work at all. Because of this, they tend to run hot. Limited High Speed Torque - In general, stepper motors have less torque at high speeds than at low speeds. Some steppers are optimized for better high-speed performance, but they need to be paired with an appropriate driver to achieve that performance. No Feedback – Unlike servo motors, most steppers do not have integral feedback for position. Although great precision can be achieved running ‘open loop’. Limit switches or ‘home’ detectors are typically required for safety and/or to establish a reference position. There are three basic types of stepper motor. They are:

- Variable-reluctance
- Permanent-magnet
- Hybrid
Variable-reluctance (VR): This type of stepper motor has been around for a long time. It is probably the easiest to understand from a structural point of view. Figure 1 shows a cross section of a typical V.R. stepper motor. This type of motor consists of a soft iron multi-toothed rotor and a wound stator. When the stator windings are energized with DC current the poles become magnetized. Rotation occurs when the rotor teeth are attracted to the energized stator poles.

Permanent Magnet (PM): Often referred to as a “tin can” or “canstock” motor the permanent magnet step motor is a low cost and low resolution type motor with typical step angles of 7.5° to 15°. (48 – 24 steps/revolution) PM motors as the name implies have permanent magnets added to the motor structure. The rotor no longer has teeth as with the VR motor. Instead the rotor is magnetized with alternating north and south poles situated in a straight line parallel to the rotor shaft. These magnetized rotor poles provide an increased magnetic flux intensity and because of this the PM motor exhibits improved torque characteristics when compared with the VR type.

Hybrid (HB): The hybrid stepper motor is more expensive than the PM stepper motor but provides better performance with respect to step resolution, torque and speed. Typical step angles for HB stepper motor range from 3.6° to 0.9° (100 – 400 steps per revolution). The hybrid stepper motor combines the best features of both the PM and VR type stepper motors. The rotor is multi-toothed like the VR motor and contains an axially magnetized concentric magnet around its shaft. The teeth on the rotor provide an even better path which helps guide the magnetic flux to preferred locations in the airgap. This further increases the detent, holding and dynamic torque characteristics of the motor when compared with both the VR and PM types [22]. Stepper motor used in robot arm is visible in figure 2.13.
2.1.9 Image processing tools

The next is to select the software platform for our architecture. As discussed in concept chapter, image processing is the most suitable technique for evaluating chain elongation. It is a advance technique of computer vision and Artificial intelligence. The image processing software developed on hardware with high computational capacity can analyze even minute part of the moving chain in industry. There are several tools available to develop image processing software. The selection of proper tool which can deliver high performance with low cost is one of the most important agenda of this thesis. Some of the tools for developing image processing software are Open Source Computer Vision(OpenCV), Matlab, ToolIP-Image Processing from Fraunhofer ITWM [23], LEADTools Image processing SDK [24]. There are also some other tools available but most of them are application specific. The above four tools are general purpose tools. Any image processing software can be developed using these tools. The comparison has been done among these four tools for our software development purpose. The selection is done on the basis of low cost, good development support and cross platform availability. The following paragraph discusses the merits and demerits of all these four tools.

ToolIP is a programming tool that supports in image processing and its analysis [23]. This tool is used to solve complex image processing algorithm and has also many additional features such as graph descriptive, highly modular and reusable, and rapid prototyping is allowed. The graph descriptive feature allows the user to create complex image processing algorithms with the help of nodes and edges. The nodes represent the components and the edges describe the signal flow among nodes. The development with the help of nodes and edges is also called prototyping phase and it can be exported as XML format.

The other tool is LEADTools. This tool has also most of the feature for image processing. The SDK is available for many programming languages. ToolIP and LEADTools, both are well capable for thesis task. But, one of the purpose of thesis is to develop very cheap system. These two tools are not open Source [24]. They have less development support and enough amount of supporting documentation is limited compared with open source tools like OpenCV. Now the choice remains in between OpenCV and Matlab. Though Matlab is also not an open source software but it has all the necessary facilities, very good development support and enough documentation for the concept developed. In latter section, the comparison between OpenCV and Matlab is done.
The open source computer vision library, OpenCV, began as a research project at Intel in 1998. It has been available since 2000 under the BSD open source license. The library is written in C and C++ and runs under Linux, Windows and Mac OS X. There is active development on interfaces for Python, Ruby, Matlab, and other languages. OpenCV was designed for computational efficiency and with a strong focus on realtime applications. [25]. It is a collection of commonly used functions that perform operations related to computer vision. OpenCV has been natively written in C/C++, but has wrappers for Python, Java, and any JVM language, which is designed to create the Java byte code, such as Scala and Clojure. Since most of the Android app development is done in C++/Java, OpenCV has also been ported as an SDK that developers can use to implement it in their apps and make them vision enabled.

Initially developed by Intel, OpenCV is a free cross-platform library for real-time image processing that has become a de facto standard tool for all things related to Computer Vision. OpenCV is aimed at providing the tools needed to solve computer-vision problems. It contains a mix of low-level image-processing functions and high-level algorithms such as face detection, pedestrian detection, feature matching, and tracking. The library has been downloaded more than three million times [26]. In 2010 a new module that provides GPU acceleration was added to OpenCV. The GPU module covers a significant part of the library’s functionality and is still in active development. It is implemented using CUDA and therefore benefits from the CUDA ecosystem, including libraries such as NVIDIA Performance Primitives (NPP). The GPU module allows users to benefit from GPU acceleration without requiring training in GPU programming. The module is consistent with the CPU version of OpenCV, which makes adoption easy. There are differences, however, the most important of which is the memory model. OpenCV implements a container for images called cv::Mat that exposes access to image raw data. In the GPU module the container cv::gpu::GpuMat stores the image data in the GPU memory and does not provide direct access to the data. If users want to modify the pixel data in the main program...
running on the GPU, they first need to copy the data from GpuMat to Mat[26]. In 2012, the nonprofit foundation OpenCV.org took on the task of maintaining a support site for developers and users.

One of OpenCV’s goals is to provide a simple-to-use computer vision infrastructure that helps people build fairly sophisticated vision applications quickly. The OpenCV library contains over 500 functions that span many areas in vision, including factory product inspection, medical imaging, security, user interface, camera calibration, stereo vision, and robotics. Because computer vision and machine learning often go hand-in-hand, OpenCV also contains a full, general-purpose Machine Learning Library (MLL). This sublibrary is focused on statistical pattern recognition and clustering. The MLL is highly useful for the vision tasks that are at the core of OpenCV’s mission, but it is general enough to be used for any machine learning problem.

MATLAB and SIMULINK

Figure 2.14(b): MATLAB SIMULINK logo [vi]

MATLAB is a commercial software package for high performance numerical computation and visualization. The combination of analysis capabilities, flexibility, reliability and powerful graphics makes it the premier software package for electrical engineers [4]. It contains thousands of functions in its library to do mathematics. It can be used to graph functions (Both 2-D and 3-D graphs are possible), solve equations, and perform statistical tests, signal processing and much more. There are some toolboxes for special applications such as Signal processing toolbox, Image Acquisition Toolbox, Computer Vision System Toolbox, Control System Design Toolbox, Vehicle Network Toolbox and more. The MATLAB help documentation is very rich and helpful for the users. Even it is possible to do wonderful things knowing a little about MATLAB [18]. It provides an excellent introductory language with built in graphical, mathematical and graphical user interface to help the users to visualize the behaviour of the model. Data structure is very important in the field of Programming. The main advantage of MATLAB is the data structure used in it. Only one data structure is present in MATLAB and
that is what the name ‘‘MATrixLABratory’’ stands for is matrix. If one limits oneself to the basic data structure of the matrix, then MATLAB syntax remains very simple and programs can be written far more easily than programs in other programming languages. It does purely numerical calculations. Nevertheless, Computer algebra functionality can be achieved within its environment using the so called ‘‘Symbolics’’ toolbox [27]. MATLAB can communicate with other high level programming language like C and FORTRAN. ‘C’ functions can be called from MATLAB. Since the MATLAB command interface involves so–called interpreter and it is an interpreter language, all commands can be carried out directly. This makes the testing of particular programs much easier. SIMULINK is an accessory from MATLAB which can help to make the modeling and simulation of a system. It has a very good Graphical User Interface. There are many functional blocks in Simulink Library and easy to use. This is a great advantage of MATLAB i.e. interaction with the special toolbox Simulink. The simulation runs under MATLAB and an easy interconnection between MATLAB and Simulink is ensured [27].

Simulink is very valuable for some reasons. As Simulink has very good GUI (Graphical User Interface) to interact with the user, it is very interesting for the users mainly the students. User can easily explore the behaviour of the model by manipulating sliders, buttons and checkboxes etc. Running the model many times with differing inputs, the user can start to see the characteristic behaviour of physical system represented by the model [Craig S. Lent, 2013]. Simulink HDL Coder can generate code for VHDL, Verilog. Automatic code generation makes code safer, also. MATLAB syntax is different from the other programming languages but it is easier for the beginner. It is interpreted rather than compiled; variable types and array sizes need not be declared in advance; it is not strongly typed; vector, matrix, multidimensional array and complex numbers are basic data types [28].

2.1.10 Blender (Modeling)

Figure 2.15: Blender logo [29]

Ton Roosendaal co-founded the Dutch animation studio NeoGeo in 1988. This studio quickly became the largest 3D animation house in the Netherlands. Ton was responsible for both art
direction and internal software development at NeoGeo. After a careful deliberation it was decided that the current in-house 3D toolset needed to be rewritten from scratch. In 1995 this rewrite began and was destined to become the 3D software tool which is known as Blender now.

Blender is released under the GNU General Public License (GPL, or “free software”). This license grants people a number of freedoms. The user is free to use Blender, for any purpose. User is free to distribute Blender. Anyone can study how Blender works and change it. People can distribute changed versions of Blender. The GPL strictly aims at protecting these freedoms, requiring everyone to share their modifications when they also share the software in public. That aspect is commonly referred to as Copyleft.

Blender is an open source 3D creation suite and it is free of cost. It supports the entirety of the 3D pipeline—modeling, rigging, animation, simulation, rendering, compositing and motion tracking, even video editing and game creation. Advanced users employ Blender’s API for Python scripting to customize the application and write specialized tools; often these are included in Blender’s future releases. Blender is well suited to individuals and small studios who benefit from its unified pipeline and responsive development process. Examples from many Blender-based projects are available in the showcase. Blender is cross-platform and runs equally well on Linux, Windows and Macintosh computers. Its interface uses OpenGL to provide a consistent experience. To confirm specific compatibility, the list of supported platforms indicates those regularly tested by the development team.

As a community-driven project under the GNU General Public License (GPL), the public is empowered to make small and large changes to the code base, which leads to new features, responsive bug fixes, and better usability. Blender has no price tag, but one can invest, participate, and help to advance a powerful collaborative tool: Blender is people’s own 3D software. [29]

### 2.2 Hough transform

The study of the features of the robot arm helps us to evaluate its movements. The features can be decided on the basis of the structure of the robot arm. Figure 2.17 shows the side view of the robot arm which has been used for collision detection for this thesis. The image of side view has circular joints in its arm segments. Circular joints are well defined geometric shape and easy for mathematical analysis in image processing techniques.
The method available for detecting circle is discussed in this section. The Hough transform is very popular features extraction technique to analyze the different types of curves in digital image processing [7]. There is no research paper on it. Rather, a person named Paul V.C. Hough has patent on "Methods and means recognizing complex pattern" [8]. This method was originally found to detect lines in an image using voting procedure in parametric space [8]. Here, voting procedure means to calculate the maximum value of local maxima for a particular point. Later Hough transform was generalized to detect circle. It is easy to understand Hough circle detection following the concept of line detection.

2.3 Line detection

Line detection is discussed first and further it is generalized for detecting circle. The very first thing to detect line or any other arbitrary shape is to discover its edges. Edges are found by high discontinuity in the value of pixel in an image. The featured image can be determined on the basis of discovered edges. There are various ways to find edges, some of them are Sobel, Robert, Prewit filter, Gaussian, Canny edge detector Blur filter [10,11,12]. Let us consider an image as in Figure 2.17, we have obtained some edge points by edge detection methods. Now we have to fit a line to that with equation of line: (2.1).

\[ y = mx + c \] (2.1)
So, the standard way to fit a line or circle is **Least Square Fit**. In equation 2.1 we have two unknowns for a line.

![Graph showing least square fit](image)

**Figure 2.17**: Edge pixels obtained from Sobel or gauss for a line

We can use standard linear solution to find two unknowns. Let us consider only three points found by edge detector as in **figure 2.18** and try to fit a line with these points. We can first assume that all these three points lie on a straight line. Next, we try to minimize the difference between them to find the unknowns. This can be done by taking derivatives with respect to \( m \) and \( c \), and set it equal to zero as shown in equation (2.3)

![Graph showing three edge points](image)

**Figure 2.18**: Three edge points considered to detect line [vii]

\[
y = mx + c = f(x, m, c) 
\]

Minimize \( E = \sum [y_i - f(x_i, m, c)]^2 \)  \hspace{1cm} (2.3)

This equation can find unknowns and fit a line. This method has limitation with more than one lines in an image as it is over constraint. This problem was solved by Paul Hough by considering
under constraint and voting concept. Hough transform solved segmentation as well as fitting a line in an image with multi-lines. Hough transform can be shortly described as:

write a line equation \( y = mx + c \), here, \( m \) is slope and \( c \) is intercept and then rewrite this equation in the plane of \( m \) and \( c \) as

\[
c = m(-x) + y
\]

For a particular edge point \( i \) this equation becomes

\[
c = m(-x_i) + y_i
\]  \( (2.4) \)

Equation 2.4 is an equation of line for a particular point in \((c,m)\) space. The value of slope for vertical line is \(90^\circ\). This can be a problem because \(\tan 90^\circ\) equals to indefinite term. So, we consider polar form of equation (2.5) to avoid this problem.

\[
r = x \cos(\theta) + y \sin(\theta)
\]  \( (2.5) \)

Here, \( r \) is the perpendicular distance from origin to straight line and \( \theta \) is an angle made by perpendicular to horizontal. Pseudo code for Hough transform using polar form equation of a straight line can be written as:

- Quantize the value of \( r \) and \( \theta \) in parametric space.
- for all points \( x_i, y_i \) draw straight lines in by varying \( m \) and \( c \)
- if \( x_i \) and \( y_i \) is an edge point
- for all \( \theta \)
  \[
r = x \cos(\theta) + y \sin(\theta)
\]
- increment the vote count \( P[\theta,r] = P[\theta,r] + 1 \)
- local maxima in parameter space gives an information of correct point on line to be drawn

### 2.4 Circle detection

This similar concept was used to generalize Hough transform to detect arbitrary shapes and geometric curves by Richard Duda and Peter Hart [9],[10]. The concept of circular Hough transform becomes clear with the understanding of Circle-point mapping from image plane to parameter plane [11].

Let us consider some edge points for circles obtained from xy-image plane \((R^2)\)
\[ I = (x_1, y_1), (x_2, y_2), (x_3, y_3), \ldots, (x_n, y_n) \]  
(2.6)

Consider 3-parameter of a circle with \((a, b)\) as center and \(r\) as a radius.
\[ S^1_{(a, b)}(r): (x - a)^2 + (y - b)^2 = r^2, r \geq 0 \]  
(2.7)

In equation (2.7), different value of \((a, b, r)\) generates different circle and its unique value gives a unique circle. The set of all different value of \((a, b, r)\): \(r \geq 0\), gives 3- dimensional parametric space denoted by \(R^3^+\). The parametric space \(R^3^+\) with \(S^1_{(a, b)}(r)\) can be mapped to all circles in image plane \(R^2\). [11] as follows:
\[ S^2_{(a, b)}(r) \subset R^2 \iff P \equiv (a, b, r) \]  
(2.8)

The relation (2.8) describes an one to one mapping , also called bijective mapping for all the circles in \(R^2\) to \(R^3\). this is called Circle point mapping and the inverse is known as point circle mapping [11] as in figure:

![Figure 2.19: The set of circles in \(R^2\) plane [11]](image)

**Figure 2.19** shows that any circle regardless of its position can be mapped from parameter space to image space. The process is described below. First we can see the mapping of circles with equal radius to parameter plane, see **figure 2.20**. Here, we have mapped the centers and radius of circles to parameter space. We can see the plane passing to \((a, b)\) plane with equal radius.
According to Generalized Hough Transform, an edge point of circle is mapped to the parameter space [10]. We consider the circles touching the origin in image plane $aR^2$ for simplicity. These circles generates right circular cone $a^2 + b^2 - r^2 \geq 0$ in $R^3$ plane with its vertex at origin, see figure 2.21.

In the next figure 2.22, we consider only two circles to make the mapping more visible. Here, we can see the edge point $(x,y)$ is fixed in parameter plane and centers of circles lie on the surface of right circular cone.
Mapping procedure described above shows that, a right circular cone is obtained with a single edge point of a circle in parametric space. If we consider three edge points of a circle in an image plane and map them in parameter space. We obtain tree right circular cone intersecting at a single point. This point signifies that all three point lies on the same circle. In this way circle can be drawn back to the image plane.

In the figure 2.23, \((p1,p2,p3)\) are edge points of a circle obtained by edge detection technique. For each point, right circular cone is drawn in \(R^3\) plane. The cones are intersected at a particular point ‘p’. This method is used for voting procedure for developing software. This intersection point is makes sure that the edge points used to obtain right circular cone are from one circle. There can be a problem with lots of noise in an image. We can get the false edge points , which in turn gives us false intersection point. It can be difficult to distinguish the correct intersection. This is one of the demerit of Hough transform.
For the sake of software development, we consider polar form of circular equation.

\[
\begin{align*}
    x_0 &= x - r \cos(\theta) \\
    y_0 &= y - r \sin(\theta)
\end{align*}
\] (2.9) (2.10)

Here, \(x_0\) and \(y_0\) are centers of a circle, \(r\) is radius of a circle. \(\theta\) is tangent direction at the edge point \((x, y)\) as in **figure 2.24**.

![Figure 2.24: Tangent direction at the edge point of a circle](image)

Pseudo code can be developed with the help of above mathematical concept. This is more or less similar to standard Hough transform for line detection.

- Quantize the terms \(x_0, y_0, r\) in the parameter space
- for all edge point \(x_i\) and \(y_i\) do
  - Loop from \(r_{\text{min}}\) to \(r_{\text{max}}\)
    \[
    \begin{align*}
    x_0 &= x - r \cos(\theta) \\
    y_0 &= y - r \sin(\theta)
    \end{align*}
    \]
- vote for each intersection \(P[x_0, y_0, r] = P[x_0, y_0, r] + 1\)
- Determine the local maxima in parameter space

In this way we can develop the software for circle detection using generalized Hough transform. Nevertheless, we have some demerit of this algorithm as follow: Hough transform outcome completely depends upon edge detection technique. It generates many false circles. It needs very high computation power, which slows down the processing of frame per second in video processing.
2.5 Bounding box Detection

It is often useful to have a rough idea of where a region is in an image. One useful concept is its bounding box, which is a rectangle with horizontal and vertical sides that encloses the region and touches its topmost, bottommost, leftmost, and rightmost points. As shown in figure 3.25, there can be as many as eight distinct extremal pixels to a region: topmost right, rightmost top, rightmost bottom, bottommost right, bottommost left, leftmost bottom, leftmost top, and topmost left. Each extremal point has an extremal coordinate value in either its row or column coordinate position. Each extremal point lies on the bounding box of the region.

Figure 2.25: Bounding Box external points of a region [viii]

Figure 2.24 shows eight extremal points of a region and the normally oriented bounding box that encloses the region. The dotted lines pair together opposite extremal points and form the extremal point axes of the shape.
3 Concept

It is an attractive application of machine vision to control the motion of a robot-arm with camera. Many methods have been proposed but they are to control them in a static scene where only the robot-arm position causes the change of the image. The main disadvantage is that the movement of the robot arm is pre planned. They are not user friendly and time consuming. Thus development of a low cost user friendly maintenance tool is necessary for the robot-arm where there movements are not fixed or pre planned.

The collision can be detected either by measurement sensors or by image processing techniques. The collision can be avoided by fix program for the robot arms. But this is user inconvenient and not dynamic. Image processing is supported by several hardware platforms and open source software is available for its development. This is why image processing is proffered. The collision of robot arm can occur in two ways. First, collision between the robot arm own segments and the second is the collision with the other objects around it.

There are different kinds of robot arms are being used in the industries. Many of them have common features like the sample robot arm in TU-Chemnitz laboratory. Such a robot arm with circular joints and segments has been considered for evaluation. When the distance of the robot arm with other objects or the distance between its segments is smaller, than there is a chance of collision.

3.1 Parameters to detect circular bounding box

The parameters discussed here have been used to analyze the image frames of the video. The developed software has considered all these parameters for detecting circular bounding box.

3.1.1 Edge detection

Edge detection is very important in image processing. This parameter makes the object more visible in image and helps to determine the shape of the object i.e. helps to recognize the objects in the image. This parameter is required for Hough circle detection function. In this thesis the video of a robot arm is captured when it is moving. Line function draw the lines in there threshold points. Thus, the shape of the circular joints, segments and other parts of the robot arms are sketched. A fix number is considered to evaluate the robot arm. A robot arm used for
this thesis has three circular joints and two segments. There are some other issues which will be explained later.

![Figure 3.1: number of circles and segments](image)

In the **figure 3.1 (b)** shows the edges detected from the original image 3.1(a). Three circular joints and two segments are clearly visible in **figure 3.1(b)**.

### 3.1.2 Size of circles and bounding boxes

Specific size of circle and bounding box can be helpful to determine specific objects in the image. An image can contain many noises. So it is important to know the size of the objects so that the expected objects can be separated from the other objects. The shapes of the objects are more visible when the edge detection is done. The robot arm has three equal size circular joints and the segments are fix size also. Some circles and bounding boxes are found in **figure 3.2**. circle detected for the circular joints are connected by a line while other circles are false circles.
3.1.3 Distance between segment center to bounding box center

This parameter is to determine the distance between bounding box center (containing the robot arm segment) and segment center i.e. the line connecting two of the circular joints center. The robot arm has two segments and three joints. Each segments has two joints at their (two) end points. The segments are rectangular and the size is fixed. The joints are circular. Other circles can be found with line function due to other objects or noise. The distance between the bounding box center and the segment center helps to determine if the circles are circular joints and the bounding box is containing the segment. If the distance is less than a given number then robot arm segments are detected.
Figure 3.3 shows the distance between bounding box center M to connection line PQ (P and Q are the center of circular joints). Figure 3.3(a) shows the perfect image when the bounding box center is on the connection line. Figure 3.3 (b) is used for an example.

3.2 Hardware and Software development concept

All the parameters discussed in previous section are helpful to determine circular bounding box. The concept for implementation, integrate all the points to develop cheap and user convenient system for detection of circular bounding box. The efficiency of the system is also considered. Concept of system architecture is shown in figure 3.4. The system architecture has two layers: Hardware and Software layer.

![System Architecture](image)

**Figure 3.4: System architecture to detect circular bounding box**

3.2.1 Hardware layer

Hardware layer is in the bottom in architecture as shown in figure 3.4. Hardware in the architecture needs to be realize in such a way that the system will be very low cost with high performance. Several options are available for image processing using a standalone hardware
system. For example, embedded hardware devices are particularly suitable in a big factory for high performance and heavy duty. On the other hand customized micro-controller based systems combined with ASIC, FPGA or DSP are also advantageous for this application. In this thesis project selection of hardware is done by making trade off between cost and performance. Figure 3.4 shows the architecture for development where hardware layer is in the bottom. In the hardware layer computer, smart phone, video camera and robot arm are shown. Computer and smart phone are connected each other as the software developed for computer will be ported to android with the help of JAVA Native Interface. A video can be made first and then send to computer. That is why, an external video camera is there in hardware layer. A model robot arm is shown to use as model object to detect. The robot arm has five degree of freedom with three circular joints and two segments. The size of the segments and the circular joints are known. So that it will be easier for the system to detect the segments with circular joints i.e. circular bounding box.

3.2.2 Software layer

Software layer is the top layer in the architecture shown in figure 3.4. There are three main modules in this layer: Peripheral module, Image processing and Observation control. The peripheral module is used to read and write the images. Observation control module is implemented to observe the output and adjusting the parameters. Image processing module is the most important part where different image processing algorithms are used to detect circular bounding box. Beside these three modules there is a module in Software layer named Robot arm simulation. This module is to make a virtual simulation of real robot arm. All these modules are explained in details in the next section. The software layer works together with the Hardware layer. The Hard ware layer passes the information to software layer for analysis and the software layer processes it. For example, the image or video captured by video camera in Hardware layer send to software layer for processing.

Different options are available to develop the image processing software. MATLAB and OpenCV are very popular for computer vision and widely used. Other computer vision libraries are also available i.e, Lead Tools, Aforge.net, VXI (Vision-something Library). This can be chosen on the basis of the applicability of the software so that it supports the developed concept. Three modules in the Software layer shown in figure 3.4 are discussed in the following subsection.
**Peripheral module**
Peripheral module will be used to read or write image and video. This module takes the image as input and sends it to image processing module for processing. In opposite it receives the output image and display on the screen. The input images can be taken in two different ways. A static image of video can be captured and store it to the computer and then use it as input to the system. The static image will be evaluated first for the proof of concept in this thesis. Then the recorded video will be used as input. The other way is the real time i.e. capturing the image or video while the application is running. First one is good during the development process and the second approach is better for performance measurement.

Peripheral module contains two functionalities in the **figure 3.4**. Image read means loading an image from the specified file and returns it. If the image cannot be read the function will return an empty matrix (OpenCV docs). Different file formats are supported i.e. bmp, jpeg, jp2, png, pbm, tif etc. the function determine the type of an image by the content not by the file extension. Image write in this module saves the image to the specified file. The image format is chosen based on the filename extension.

**Image processing pipeline module**
Image processing module is will be responsible to analyze the images or videos. This module will contain different image processing algorithms to process the image. Thus the module is named Image processing module. Edge detection, Hough circle detection, Bounding box detection and the distance measurement will be done here. Edge detection is required for circle detection. This is useful for some other algorithms also. Circle detection will extract the circular objects from the image and bounding box will enclose other. For example, circle detection will detect the circular joints and bounding box will enclose robot arm segments. Then the distance will be measured from bounding box center to line between circular joints with that segment. The detection of circle and comparison of different parameters between image frames increases the computation complexity. In order to decrease the complexity color input video is converted to grayscale image and then to binary image, if required.

Colour space conversion:
Colour space of the image will be changed to Greyscale/Intensity **figure 3.5**. Colour space conversion block is used to convert the input image from a colour space to Intensity image.
RGB (Red, Green and Blue) image is a type of true colour image and Intensity image is the image where pixels contain the colour of shades from white to black (0 to 255) that is ‘Greyscale’ image. RGB values are converted to Greyscale by forming a weighted sum of the R, G and B components.

\[ 0.2989 \times R + 0.5870 \times G + 0.1140 \times B \ [\text{www.mathworks.com}] \]

Parameters initialization:
We will initialize some parameters for our system to develop. We will define the range of the circle size by minimum and maximum radius. Canny edge detection is required for Hough circle detection. So minimum and maximum threshold values are required for canny edge detection. Height and width ranges will be used for defining bounding boxes. We will define the maximum distance (distance between bounding box center and segment circle connection line) which is acceptable to detect the right segments.

Image filtering:
Normally images contain many noises. Image filtering is the process to reduce noises from the image. There are different filters available to reduce noise. Some of these filters are discussed below.
Sobel Derivative: Irwin Sobel gave the idea of “Isotropic 3*3 Image gradient operator”[12]. Sobel concept uses two 3*3 Kernels for convolution of input image. This calculates the approximations of the derivatives in both horizontal and vertical direction.

Let us consider that the input image to be operated is $Img$, $Grad_x$ and $Grad_y$ as two images with horizontal and vertical approximation which at each point,

$$Grad_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * Img$$

$$Grad_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * Img$$

Here, * sign represents for convolution and the kernel size is 3. The outcome of the gradient approximation is used to calculate the magnitude, using

$$Grad = \sqrt{Grad_x^2 + Grad_y^2}$$

The $Grad$ value is used to calculate the gradient directions.

Hough circle:
Hough circle is derived from Hough line transform introduced by Paul Hough in 1962[8]. Hough transform as it is used today was invented by Richard Duda and Peter Hart in 1972, who called it a “Generalized Hough Transform”[10]. Hough circle finds the circles from a grayscale image using a modification of the Hough transform. This function desire edge detection of the image. It is better to provide a range the size of circle to extract circles otherwise all the circles are extracted. Three parameters are required to define a circle.

$$C : (x_{center}, y_{center}, r)$$

Here ($x_{center}, y_{center}$) is co-ordinates of the center position and $r$ is the radius.
Bounding box:
Minimum bounding box is the co-ordinates of the rectangular border to enclose an object with minimum area. Let us consider an object with height ‘H’ and width ‘W’. The upper right co-ordinate of the object is (x, y). So the Bounding box is represented as,

\[ BB : (x, y, H, W) \]

Circle Bounding Segment (CBS) preference:
The robot arm has three circles (circular joints) and to segments. The minimum area that contain the circles and the segments are named as circle bounding segment, shown in figure 3.6.

Figure 3.6: Circle bounding segment

Image processing is very complex and the calculation takes time and energy for the computer. In terms of video processing it is more complex now a day. For increasing complexity of hierarchical data sources and evaluation proceedings, a describing approach is required to describe single cases in a group way in order to make them easier to handle and extendable and to define weighting adaption on the system[30]. A flexible data structure is introduced by Prof. Dr. Wolfram Hardt and Dipl. Ing. Daniel Reißner, which allowed for energy efficient calculation of preference by short message and direct data access. In their work they present a architecture which allows for description of complex systems by grouping preferences and defining observer-rules on top of it. This allows by a compact group-value format energy-saving
short messages. Furthermore, the group-value format allows longer sleep times of system by faster response by combination of iterative stored procedure with index lifting. Hierarchical preference integrations and observer rules allow direct data access (low interface overhead). This way Tools only need browser on client side and routing application can work on low power microcontrollers [30]. In this thesis the “group-value format” is used to detect Circle Bounding Segment.

**Observation control module**

This module will display the update version of the image or video after analyzing them. There will be slider bar to adjust the parameters i.e. size of the circles (minimum, maximum), size of the bounding box (minimum, maximum), minimum distance for collision etc. The current situation will be observed here while the application is running and the parameters will be adjusted with the slider. Distance between found segments and the distance between dummy objects will be checked and value of required distance will be adjusted. Red or green line between the circular joints will be drawn to show the result. The status of the other objects will be visible here in this module also.

As the length of the robot arm segments and the size of the circular joints are known some conditions are helpful to detect the robot arm segments.

- Biggest two segments in acceptable tolerances are robot arm segments.
- Reject if one segment is smaller than other two segments
- Accept the segment if it is bigger than possible smallest segment

**Robot arm simulation**

The purpose of simulation is to create a model of robot arm. The movements will be modeled as the real robot arm. A video of moving robot arm will be made to test it in the designed system. Simulation video is helpful to develop the system. The real scenario of the robot arm can have much noise and the problem of light effect. These problems can be maintained in simulation. Moreover it is better to have a test for collision detection in simulation video rather than a collision in real.
Figure 3.7: Image of real robot arm and model of a robot arm

**Figure 3.7(a)** shows the image of a real robot arm available in the Technische Universität Chemnitz computer laboratory. **Figure 3.7(b)** shows a model of that robot arm created with Blender modeling tool.
4 Realization and Implementation

The concept to implement the system is discussed in the previous chapter. The basic of image and image processing is also discussed. This chapter is to present the implementation part of the thesis. It is very important to select proper hardware and software to implement the system in the beginning. The hardware and software are selected by their costs, performance, portability and availability.

4.1: Realization of Hardware and Software

In terms of hardware selection a Personal computer is cost effective, easily available and user convenient. It has high performance and the software developed for the Personal computers are easily portable to other hardware’s such as android (in smart phones). Web camera is available and cheap for capturing image or video. That is all for the hardware selection.

Image processing is an advance technique of computer vision. There are different algorithms to detect the shape of the object in image. It is also possible to measure the distance between the objects. Many tools are available for digital image processing. Among them the selection of proper tool is which can deliver high performance with low cost. The tool should be user friendly also. Some of the image processing tools are discussed in the second chapter (2.1.9: Image processing tools). OpenCV and MATLAB are the most popular image processing tools among them.

MATLAB is very popular and widely used now a day. There are many advantages of using this tool. So many algorithms for image processing are already implemented in MATLAB and the memory management is done by itself. It is easy to use and easy to learn. But there are some limitations also. The tool is very expensive. The license cost around $2000. Again it makes the system slow taking top priority from the operating system. In terms of image processing it can process only four to five frames per second where OpenCV can process more than thirty frames. OpenCV is free and has Development Pack (Tegra Android Development Pack from NVIDIA) for android development. This Thesis is focusing on low cost and high performance. So OpenCV is selected for the implementation of the system. A comparison between OpenCV and MATLAB are given below:
<table>
<thead>
<tr>
<th>OpenCV</th>
<th>MATLAB</th>
</tr>
</thead>
<tbody>
<tr>
<td>It is faster in image processing</td>
<td>MATLAB is slow</td>
</tr>
<tr>
<td>It is specific to image processing</td>
<td>It is general purpose tool</td>
</tr>
<tr>
<td>It delivers high performance in terms of speed with the processing of 30+ frames per second</td>
<td>It can only process 4 to 5 frames per second</td>
</tr>
<tr>
<td>Different IDE can be used for OpenCV such as eclipse, visual studio, Qt and even simple editor for OpenCV. This gives more choice to developer.</td>
<td>There is no other choice other than Matlab editor</td>
</tr>
<tr>
<td>NVIDIA has developed Tegra Android Development Pack which provides complete software package for Android development with OpenCV.</td>
<td>There is not any complete package for Android development in Matlab.</td>
</tr>
<tr>
<td>In terms of cost OpenCV is BSD licensed which is free</td>
<td>Matlab license can cost around $2000</td>
</tr>
</tbody>
</table>

Table 4.1: OpenCV and MATLAB Comparison

Linux platform with Eclipse IDE (Integrated Development Environment) is selected to develop the software because they are free to use. There are some other packages which should be used in order to use OpenCV library. These packages can be installed on Ubuntu via its software center with internet connection. Some of the important packages are cmake, pkgconfig, python, python numpy, ffmpeg, gimp, libopencv, c++ plugin for eclipse. Cmake has been used because it provides cross platform build environment for open source projects. The make files and work spaces generated by it can be used in our compiler environment. A multimedia framework is required for reading and writing video. FFmpeg package has been used for this purpose. This package is capable for decode, encode, transcode, mux, demux, stream, filter and play any kind of multimedia. The package gimp which is the short form of GNU Image Manipulation Program is installed to edit the input image on if required. An OpenCV code is native developed with C++ and it is also preferred programming language for our software development. Therefore, C++ plugin has been installed in eclipse as it provides only JAVA development without any addins. The combinations of all these packages have made the OpenCV development work efficient.

Blender is selected for modeling as it is open source also. The software is free and easy to use. Blender is widely used for modeling.
Figure 4.1(a) shows the realization of Hardware and Software for the designed system. Personal Computer or laptop with webcam is required to implement the monitoring system. Robot arm is shown to capture a real video of it or to create a model in blender. Video of a robot arm will be captured by web camera and send as input. The other way is to create a model in Blender and make a video of that for input. Peripheral module will read or write the image and observation control module will display the image frames and provide graphical user interface to adjust the parameters by sliders. Image processing pipeline module will process the image using different algorithms like Sobel edge detection, Canny edge detection, Hough Circles and Bounding rectangle detection to get the required information from the image. Thus the system will work.
4.2: Implementation

The implementation process of the designed system is described in this section. Architecture of the system for concept and realization is discussed before. System architecture for the implementation is given below.

Figure 4.1(b) shows the necessary parts of the implementation. The Software layer shows the OpenCV functions used for the development part. Observation control module has two functions. Function createTrackbar() will display the slider bar on the screen so that the user can adjust different parameters while the video is running. Function namedWindow() will display the output. Other functions are described in the following sections.

Figure 4.1(b): System architecture for implementation
4.2.1: Implementation of Peripheral module

This module is responsible of reading input images or videos and displaying the output. The very first thing to start implementation is to decide the input criterion of image. Input image has been captured in two ways. The first one is recorded video or image as an input and the second one is real-time video through web-cam of PC. Three different images are shown below in Figure 4.2. In these images the robot arm is in three different positions. **Figure 4.2 (c)** shows the segments of the robot arm are very close to each other. This image is important to analyze.

![Figure 4.2: Sample image frames of robot arm](image)

Figure 4.2 shows the segments of the robot arm are so close that there a collision can occur. This is one of the worst cases for analysis. This can happen in different positions where the segments are getting closer than the minimum distance to keep the movement of robot arm safe. The concept for the software has been tested on all these kinds of frame.
There is the need of capturing different kinds of frame either as a static image or from running video. Three different type of inputs have been analyzed in our case. They are given below.

**Input as a static image:** The static input image has been analyzed at the beginning of the work for proof of concept. The starting of the work has been done by taking one single static image frame. The circular joints was successfully detected of static image frame by the use of simple Hough transform for circle detection. Some filtering has been done for static input in order to reduce the noise.

The functions of OpenCV library such as `imread()` has been used to read the input image from the project folder. At the same time detected circle on the input image by Hough transform has been displayed on the PC monitor on a separate window for user. This static image input is not suitable for real time maintenance process of industrial carrier belt. As it can only analyze very small part of the carrier belt. Therefore, this input has been only used to test the hough transform for circle detection.

**Input as recorded video:** The recorded video is an extended form of single image frame. The maintenance process in the industries can be done after some time interval as it will not directly affect the manufacturing process. Therefore, video of carrier belt can be recorded after every regular time interval and can be send for analysis. The video can also analyze the extended part of the chain. After the recorded video is analyzed, it has been saved for future reference. The analyzed output video can be seen by any user and come across the condition of chain. The function `VideoWriter()` has been used to save the output video for future reference. `VideoWriter` class has been used to display the processed video. It takes five parameters as:
Video-writer(const string& filename, int fourcc, double fps, Size framesize, bool isColor = true)

The parameter `fourcc` is for Four character code of codec used to compress the frames. 'P','I','M','1' has been used here. It means MPEG-1 codec. This has been used because MPEG-1 codec writes the output video without excessive loss of quality so that it is easier to view and analyze. Parameter `fps` is frame per second. The `fps` has been kept same as input video in order to get the real glimpse of speed of carrier belt. Frame size has also been taken from the input video stream. The last parameter `isColor` is considered true to display the output as color image for better view of user. The proper selection of parameters for the function VideoWriter() has been done to display well readable output for analysis purpose.

**Input from a webcam of PC:** The analysis of industrial robot arm with the recorded video will not be useful for the industries with huge production line where the maintenance process is important in real time. In such case they should be monitored in run time by taking their regular input. The regular monitoring can reduce any kind of hazardous event to happen in the industries production line. The system designed in this thesis for detection of robot arm is to monitor them during run time. In order to realize this, web-cam connected to computer has been placed in front of the robot arm which captures the video when it is moving and send it to the computer for analysis. The output is also displayed at the same time on the PC monitor.

These are the three methods for taking input for PC implementation. All the three methods have been tested successfully.

**4.2.2: Implementation of Observation control module**

This module has been implemented to update the value of parameters obtained from the image frames of running video. Figure 4.4 shows two image frames where robot arm segments are in different positions. Normally segment position changes from Figure 4.4(a) to 4.4(b) will take a lot of image frames in real time video but these frames are shown as example for better understanding. In Figure 4.4(b) the segments are so close that there is a chance of collision. Two of the circular joints are also very close. This frame has been considered for our evaluation of robot arm segments collision detection or detection of the distance between shapes.
An image frame with updated values of the analyzed parameters is called model or reference image frame. The central controller module updates the various parameters of this image frame such as circle and bounding box center position, size of circle and bounding box, number of circles and bounding box detected in one image frame, vector detection which tell the direction of movement of the circle, the distance between bounding box center to connection line between the circle centers at two end points of that bounding box (Robot arm segment).

These all parameters are maintained as a structure for implementation as shown in the code below.

```c
struct recircle{
    float x;
    float y;
    float r;
    float redetects;
    int typ;

    recircle(float x, float y, float r){
        this->x = x; this->y = y; this->r = r; this->redetects = 0; typ=0;
    }
};

Mat image, imagegray, canny_output;
```
int rmin = 35;
int rmax = 35;
int cannythreshmax = 110;
int centerthresh = 11;
int cthreshl = 44;
int cthreshh = 138;
int maxapproxdist = 20;
int minbbheight = 123;
int minbbwidth = 21;
int minbbheight2 = 45;
int minbbwidth2 = 8;
vector<recircle> circlespm;
int maxhredetects = 0;
int maxhredetects3 = 0;
size_t maxhidx = 0;
float seg12 = 0;
float seg23 = 0;
float seg13 = 0;

dist(Point p1, Point p2){
    int dx = p1.x-p2.x;
    int dy = p1.y-p2.y;
    return sqrt(dx*dx+dy*dy);
}

void writeText(std::ostringstream& oss, int x, int y){
    putText(image,
    oss.str(),
    Point(x, y),
    FONT_HERSHEY_SIMPLEX,
    0.6,
    Scalar(0,255,0),
    1,8,
    false);
}

int getmin(int a, int b, int c){
    if(a < b && a < c) return a;
    if(b < a && b < c) return b;
    return c;
}
Here, `recircle` is the structure data type for Vector template for C++. Vector is a kind of array which can change its size according to the need during run-time. It can create the space during run-time to append the new values. This facility of C++ is used extensively for updating the runtime values to maintain the record of parameters of image frames. This structure `recircle` has been created to update the parameters of the model frame in each iteration. In the above list $x$, $y$ and $r$ are the parameters for circle center $(x,y)$ and radius $r$, detection has been defined for increment and decrement. The `redetects` is a counter which will increase by one when it get the circle in same points with the previous frame.

$r_{min}$ and $r_{max}$ are indicating the range of the size of circle for the circular joints. $cannythresh_{max}$ and $centerthresh$ are the maximum and minimum values for canny edge detection used in the function `HoughCircles()`. $cthresh_l$ and $cthresh_h$ contain the values to use in the function `Canny()` to find the edges from the image. $maxapproxdist$ is the maximum distance between bounding box center and circular joints connection line. $minbbheight$, $minbnwidth$, $minbbheight_2$, $minbnwidth_2$ are minimum and maximum height and width of the bounding boxes enclosing the robot arm segments. All these parameters ($r_{min}$, $r_{max}$, $cannythresh_{max}$, $centerthresh$, $cthresh_l$, $cthresh_h$, $maxapproxdist$, $minbbheight$, $minbnwidth$, $minbbheight_2$, $minbnwidth_2$) can be adjusted by the user with the slider bar during the video is running.

### 4.2.3: Implementation of Image processing pipeline module

The input received by the peripheral module arrives to this module for further analysis. The main task of this module is to compare parameters of image frames of robot arm. The parameters to be compared have been discussed in observation control module. The comparison of image frame is done with the updated model frame of observation control module. This gives the current situation of the robot arm movement. Comparison of each image frame during run time can detect any discrepancies and can prevent an accident. After capturing the input video, further filtering of image has been done to reduce the computation complexity and noise. Two steps have been taken for this:

**Converting color image to gray image:** The color input image has been converted to grayscale image. This reduces the computation complexity. The color image has three channels of eight bits each whereas the grayscale image is single channel with eight bits. Thus, using grayscale
image for analysis reduces the computation complexity and makes the feature detection easier and faster.

**Filtering input image:** The next step for reducing computation complexity is to filter out the grayscale image. This reduces the noise and makes the edge detection easier. Detecting edges is one of the most important tasks as the circular joints and segments of the robot arm have sharp edges. There are several filters available for image filtering. Some of them are discussed in concept chapter. Sobel filter has been used for this task. Because, Sobel operator uses first, second or mixed image derivatives and this gives better edge detection. OpenCV has inbuilt function for Sobel filter. This function takes nine arguments as shown below,

```c
void Sobel(InputArray Source, OutputArray destination, int ddepth, int dx, int dy, int ksize, double scale, double delta, int borderType)
```

It simply takes the grayscale image and generates the separate output image which will be used for feature detection of robot arm. The argument ddepth is for depth of the output image which has been considered sixteen signed bit and this is enough for feature detection. The first order derivation has been performed for filtering with kernel size three. This gives the output image with less noise and better edge detection. The output image obtained after filtering has been used by this module for comparing the image frames with model image frame. This module has been categorized with further sub-modules as shown in the figure.

![Figure 4.5(a): Sub-modules of image processing pipeline module of software layer](image)

**Figure 4.5** shows four sub-modules of image processing pipeline module. A flowchart for implementation of image processing pipeline of the designed system is showing in **Figure 4.5(b)** to display the sequence of activities in this process.
Figure 4.5(b): Flow chart to detect robot arm and its chance of collision
Circle detection and management

This sub-module considers the image frame from observation control module as model image frame and compares all other upcoming image frames with that image frame of video. As we have seen in the peripheral module captures the input frames of images. The robot arm in the image has three joints which are circular in shape. These circular joints of robot arm have to be detected. These circles can be detected by some edge detection techniques such as canny edge detector, sobel operator or scharr filtering. These edge detection techniques need very less computation power and can increase the performance. At the same time they have some drawbacks regarding circle detection. These techniques detect any kind of edges present in an image which is not the aim of our work. We need only the circular joints and the segments to be detected. This brings more work to distinguish circles from all kind of features detected by above methods. Many false circles are also detected by these methods. Therefore, it is not suitable to use these methods for this thesis work.

OpenCV has special function for detecting circle in an image. It is Hough transform for circle detection. It has been used to detect circles. This needs very high computation power and also gives many false circles. Improving our algorithm for Hough transform and selection of proper parameters has helped to eliminate the false circle detection. The selection of parameters in hough function is very important to eliminate the false circle detection. Apart from the circular joints of the robot arm, there are many other circles which can be seen in Figure 4.6 on the robot arm image frames. These circles should not be considered for analysis purpose. Hough transform for circle detection will detect all the circular shape present on the image. The algorithm has been designed in such a way that only the circles for circular joints will be detected. Hough circle function has been discussed below along with the selection of values for the parameters.

Figure 4.6: False circles (red circles in image) detected by HoughCircles
void HoughCircles(InputArray image, OutputArray circles, int method, double dp, double minDist, double param1, double param2, int minRadius, int maxRadius)

The parameter method is the detection method for circles. CV_HOUGH_GRADIENT method has been taken in our case. This method takes finds points from the edges and determines if it is a circle. This criterion of selecting edge points to find circles differentiates HoughCircle() functions from any other Hough transform method. This is the only method present for Hough function in OpenCV till date. It has been discussed in more detail in next paragraph. Parameter dp is for selecting resolution factor of an image. The bad resolution can decrease the visibility. It can also detect some false circles. It has been tried to keep the resolution same as the input image. This has been done by taking the value of dp as '1'. In case if we take the value '2', it will increase the resolution. At the same time it will also increase the computational complexity. Parameter minDist is the minimum distance between the detected circles. The three circular joints of the robot arm are equal in size. Parameters param1 and param2 are higher threshold value passed to Canny edge detector into the method CV_HOUGH_GRADIENT. These are the maximum and minimum threshold values for detecting edges of the circles. The wrong range of these values will generate either many false circles or cannot detect the circles. Each image has different threshold values. It is generally found by testing with some random values maintaining a range. In our case we range the values 110 for higher threshold and 11 for lower threshold. The last arguments are minRadius and maxRadius. Theses radiuses are minimum and maximum circle radius of the circles. In our case both the values are 35 because we are using slider bar so we can take the value we need. The data structure for the circle is given below.

```c
struct recircle{
    float x;
    float y;
    float r;
    float redetects;
    int typ;

    recircle(float x, float y, float r){
        this->x = x; this->y = y; this->r = r; this->redetects = 0; typ=0;
    }
};
```
Here x and y are the coordinates of the center point and r is the radius. The variable redectes is how many times the circle is redected. In recircle() this->x = x, points the current object. This is same for y and z.

One can find several ways to detect circle based on Hough Transform [31, 32]. Some of them are: The GKHT with edge direction, The 2-1 Hough Transform, The Gerig and Klein HT and Fast Hough Transform. These methods have somehow better efficiency and storage capability. But, the performance decreases with the complex images. So, H.K Yuen and his team combined the merits of these methods and developed their own method called CV_HOUGH_Gradient [33]. This method has decreases the storage and computation requirement.

The proper selection of parameters as described above gives the good output image of circle detection in the case of static input image as in figure 4.7. The circles has been drawn on output image with function void circle(Mat& img, Point center, int radius, const Scalar& color, int thickness=1, int lineType=8, int shift=0). It is a OpenCV function to draw the circles. The parameter img is the image where the circle is drawn, center is the coordinates of the center of the circle, radius take the radius of the circle. Some other parameters are also there i.e. color of the circle, thickness of the circle outline, line type of the circle boundary, Number of fractional bits in the coordinates of the center and in the radius value.

The result shown in figure 4.7 is obtained when input is static. In order to deal with dynamic detection and not detection changes in image a dynamic possibility for regrouping model circles and new detected circles by current evaluation is required. Therefore homogeneous basis set of
model circles is compared with set of current frame circles and new result groups are formed depending on position, detection rate, detection rate increments and vector forming. By homogeneous grouping good model points are continuous strengthened in detection while wrong detection's are as error detection's automatically in other groups on preference adapting requirements filtered. A rate aware preference adapts and weights continuous parameters like position, detection rate and vector forming for decision for good circles. This will be described more in detail. The aim of this thesis is to monitor the industrial robot arm regularly. The input is not limited to static rather the focus has been kept for video analysis to find defect any time during the robot arm is active. The real problem occurs with the circle detection of video input which is the real time analysis of robot arm. The designed algorithm for circle detection in videos has been explained further with comparison of all the frames with model frame.

In the video input each frame of image sequence is evaluated separately. Several parameters as discussed in previous section have been studied in each frame to make comparison between the running image frames of robot arm.

The function HoughCircles() can find many circles including false circles from image. Our goal is to find the specific circles enclosing the circular joints of the robot arm. So we need filtering to detect the right circles. The circular joints of the robot are will be the most significant circles in the image frames. When the distance between the model frame circle to current frame circle center is less than a given distance (80 pixels in our case), the current frame circle center is get redetected and the variable redetects increased by one. The distance is measured by the function dist(). The function takes two points and return the point distance between them using the equation \( c = \sqrt{(X_A - X_B)^2 - (Y_A - Y_B)^2} \). Where \( c \) is the distance and A,B are two points. Most significant circles are those which have best redetection rates. Among three line distances from the circle centers (to each other) of the circular joints, two will remain same while other one can change due to the movement of the robot arm. Figure 4.8 (a) and 4.8 (b) shows the distances between the circle centers enclosing circular joints. Distance between the middle circle center A to other circle centers B, C is remaining same while distance between B and C is changing.
Figure 4.8: Distance between segment circle centers

Code section to find the Hough circles and redetection count is given below. Vector is used for the circles. Vectors are sequence containers representing arrays that can change in size. Just like arrays, vectors use contiguous storage locations for their elements, which means that their elements can also be accessed using offsets on regular pointers to its elements, and just as efficiently as in arrays. But unlike arrays, their size can change dynamically, with their storage being handled automatically by the container.

```
vector<Vec3f> circles;
    HoughCircles( imagegray, circles, CV_HOUGH_GRADIENT, 1, imagegray.rows/8,
                  cannythreshmax, centerthresh, rmin, rmax );
    for( size_t i = 0; i < circles.size(); i++ )
    {
        Point center(cvRound(circles[i][0]), cvRound(circles[i][1]));
        int radius = cvRound(circles[i][2]);
        circle( image, center, 3, Scalar(0,255,0), -1, 8, 0 );
        circle( image, center, radius, Scalar(0,0,255), 3, 8, 0 );
        int isredetect = 0;
        for( size_t j = 0; j < circlespm.size(); j++ ){

            if(dist(center,Point(circlespm[j].x, circlespm[j].y)) < 80){

                circlespm[j].redetects++;
            }
        }
    }
```
Bounding box detection and management

The main goal of this thesis is to detect the circular bounding box in video streams. Circular bounding box means the combination of circles and bounding boxes. It is important to extract important features from image to identify an object. In this experiment the robot arm has two specific features which are very useful. The first one is three equal size circular joints which is already implemented in last section (4.2.3.1 : Circle detection and management). The other one is two segments of the robot arm connected by these joints. The shape of these two segments is rectangular. These two segments are required to be detected. These segments can be detected by some edge detection techniques also. The problem is they detect any kind of edges present in the image frame. In this section we need the rectangles enclosing the segments of the robot arm. There are many other rectangles enclosing other thing in the image frames can be detected also. There are some rectangles can be detected also, where nothing is visible by human eyes. The algorithm should be designed in such a way that only the rectangles with segments will be detected.

There are different functions available in OpenCV to determine different shapes. The function Rect boundingRect(InputArray points) calculates the up-right bounding rectangle of a point set. Rect is template class for 2D rectangles, described by two parameters. Coordinates of the top-left corner, which is a default interpretation of Rect_ ::x and Rect_ ::y in OpenCV. Though, one may count x and y from the bottom-left corner. the other parameter is rectangle width and height. Parameter points is Input 2D point set, stored in
std::vector or Mat. The function calculates and returns the minimal up-right bounding rectangle for the specified point set. But this function encloses more space rather than required space when the object is not vertically or horizontally aligned. That is when the object is rotating on the plane this function is not suitable. Function RotatedRect minAreaRect(InputArray points) finds a rotated rectangle of the minimum area enclosing the input 2D point set. The class RotatedRect represents rotated (i.e. not up-right) rectangles on a plane. Each rectangle is specified by the center point (mass center), length of each side (represented by cv::Size2f structure) and the rotation angle in degrees. The function calculates and returns the minimum-area bounding rectangle (possibly rotated) for a specified point set.

![Diagram](image)

Figure 4.9: Difference between function Rect boundingRect(a) and RotatedRect minAreaRect(b)

**Figure 4.9** shows the difference between the functions Rect boundingRect in Figure 4.8(a) and RotatedRect minAreaRect in Figure 4.8(b)). Figure 4.8(a) shows the rectangular object ‘B’ is enclosed by Area ‘A’ as it calculates the up right bounding rectangle. Figure 4.8(b) shows the same object enclosed by the area ‘A’ which is rotated with the object as the function minAreaRect() finds a rotated rectangle of the minimum area. The function minAreaRect() needs an array of points as input to get the minimum area of the rectangle. OpenCV has a built in function called findContours() to get these points. Contour is the boundary line or edge of an object.

findContours(InputOutputArray image, OutputArrayOfArrays contours, OutputArray hierarchy, int mode, int method, Point offset=Point())
This function finds contours in a binary image. Parameter `image` is source, an 8-bit single-channel image. Non-zero pixels are treated as 1’s. Zero pixels remain 0’s, so the image is treated as binary. Contours are detected contours. Each contour is stored as a vector of points. Parameter `hierarchy` is an optional output vector, containing information about the image topology. It has as many elements as the number of contours. For each i-th contour contours[i], the elements hierarchy[i][0], hierarchy[i][1], hierarchy[i][2], and hierarchy[i][3] are set to 0-based indices in contours of the next and previous contours at the same hierarchical level, the first child contour and the parent contour, respectively. If for the contour i there are no next, previous, parent, or nested contours, the corresponding elements of hierarchy[i] will be negative. CV_RETR_LIST mode retrieves all of the contours without establishing any hierarchical relationships. CV_CHAIN_APPROX_SIMPLE compresses horizontal, vertical, and diagonal segments and leaves only their end points. For example, an up-right rectangular contour is encoded with 4 points.

The function `findContours()` can return many contours. So the function `minAreaRect()` can find many rotated rectangles. But we only need to find two segments i.e. only two rectangle enclosing two contours. For this task filtering is required. If the difference between rectangle height or width and minbbheight or minbbwidth is less than forty (in our case), these rectangles are considered.

**Distance measurement**

The goal of this thesis work is to detect the circular bounding box. The circles and the rectangular boxes are already extracted from the previous sections. The task now is to determine the circles and the rectangles containing the robot arm segments. Each of the image frames contains many circles and contours.

Each of the circles is compared to each other. The minimum area rectangle containing contours are compared to each other also. Some parameters of the system can be adjusted with the slider bar during the video is running. Any change in slider bar will call the function `trackbarchange()`. In the slider bar minbbheight and minbbwidth are taking minimum height and width value for the minimum area of rectangles containing contours. Others will be ignored. When the height and width of the current rectangle are bigger (in our case 30 pixels or more) than minbbheight and minbbwidth, this rectangle is considered. For each two most significant circles the midpoint of their connection line is taken as segment center.
(segcenter). If the distance between segment center and current rectangle center is less than a specific value (40 in our case) and the distance between two circle centers is in a range between 210 and 270 then the line joining two circle centers are drawn and value of redetects for these circles are increased. Code for bounding boxes is given below.

```cpp
for( size_t i = 0; i < circlespm.size(); i++ ){
    for( size_t j = 0; j < circlespm.size(); j++ ){
        Point p1(cvRound(circlespm[i].x), cvRound(circlespm[i].y));
        Point p2(cvRound(circlespm[j].x), cvRound(circlespm[j].y));
        Point segcenter( (p1.x+p2.x)/2, (p1.y+p2.y)/2 );
        for(size_t k=0; k<boundRect.size(); k++){
            RotatedRect curRectangle = boundRect[k];
            if( (abs(curRectangle.size.height - minbbheight)<30 &&
                abs(curRectangle.size.width - minbbwidth)<30 ) ||
                ( abs(curRectangle.size.width - minbbheight)<30 &&
                abs(curRectangle.size.height - minbbwidth)<30 ) ){
                if( dist(curRectangle.center, segcenter) < 40 &&
                    dist(p1,p2) < 270 && dist(p1,p2) > 210){ //werte mit scalar factor parametrisieren um zoombar zu machen wie bei Antennendetektion
                    line( image, p1, p2, Scalar(0,200,0), 3, 8 );
                    circlespm[i].redetects++;
                    circlespm[j].redetects++;
                    std::ostringstream oss;
                    oss << dist(p1,p2);
                    writeText(oss,(p1.x+p2.x)/2, (p1.y+p2.y)/2);
                }
            }
        }
    }
}
```

Collision detection

The robot arm has two segments with threes joints where one joint is common for both. A collision can occur when the other two joints are very close. Figure 4.8 (a) and 4.8 (b) shows...
the distance between joints. Joint A is keeping the same distance from joints B and C. But distance between B to C is changing due to movement of the robot arm segment. So a collision can occur when a segment is smaller than other two segments and the length of the segment is less than a given value from the slider (in our case the value is less than 237). Segments lengths are calculated as the distance between each pair of the segment circle centers (joint A, B and C). Code is given below.

```c
{
    if(seg12 < seg23 && seg12 < seg13){
        if(dist(Point(circlespm[maxhidx].x,
circlespm[maxhidx].y),Point(circlespm[maxhidx2].x,
circlespm[maxhidx2].y))>237 )
            line( image,
Point(circlespm[maxhidx].x,circlespm[maxhidx].y),
Point(circlespm[maxhidx2].x,circlespm[maxhidx2].y),
    Scalar(30,255,30), 1, 8 );
        else
            line( image,
Point(circlespm[maxhidx].x,circlespm[maxhidx].y),
Point(circlespm[maxhidx2].x,circlespm[maxhidx2].y),
    Scalar(0,0,255), 15, 8 );
    }
    if(seg23 < seg12 && seg23 < seg13){
        if(dist(Point(circlespm[maxhidx2].x,
circlespm[maxhidx2].y),Point(circlespm[maxhidx3].x,
circlespm[maxhidx3].y))>237 )
            line( image,
Point(circlespm[maxhidx2].x,circlespm[maxhidx2].y),
Point(circlespm[maxhidx3].x,circlespm[maxhidx3].y),
    Scalar(30,255,30), 1, 8 );
        else
            line( image,
Point(circlespm[maxhidx2].x,circlespm[maxhidx2].y),
Point(circlespm[maxhidx3].x,circlespm[maxhidx3].y),
    Scalar(0,0,255), 15, 8 );
    }
    if(seg13 < seg12 && seg13 < seg23){
        if(dist(Point(circlespm[maxhidx3].x,
circlespm[maxhidx3].y),Point(circlespm[maxhidx2].x,
circlespm[maxhidx2].y))>237 )
            line( image,
Point(circlespm[maxhidx3].x,circlespm[maxhidx3].y),
Point(circlespm[maxhidx2].x,circlespm[maxhidx2].y),
    Scalar(30,255,30), 1, 8 );
        else
            line( image,
Point(circlespm[maxhidx3].x,circlespm[maxhidx3].y),
Point(circlespm[maxhidx2].x,circlespm[maxhidx2].y),
    Scalar(0,0,255), 15, 8 );
    }
    if(seg13 < seg12 && seg13 < seg23){
```
if(dist(Point(circlespm[maxhidx].x, circlespm[maxhidx].y),Point(circlespm[maxhidx3].x, circlespm[maxhidx3].y))>237 )
    line( image,
    Point(circlespm[maxhidx].x,circlespm[maxhidx].y),
    Point(circlespm[maxhidx3].x,circlespm[maxhidx3].y),
    Scalar(30,255,30), 1, 8 )
else
    line( image,
    Point(circlespm[maxhidx].x,circlespm[maxhidx].y),
    Point(circlespm[maxhidx3].x,circlespm[maxhidx3].y),
    Scalar(0,0,255), 15, 8 );
}
}


5 Result Analysis

The main goal of this thesis is to detect the circular bounding box from video stream. Circular bounding box is the combination of circles and rectangles (bounding box). The robot arm in TU Chemnitz lab is used as a model for this work. A real video of the moving robot arm is taken and tested with the system implemented. The video is taken from a distance of three meters and it was moving while capturing. The video is taken to have a real scenario of its use in reality. Because, it is possible to do simulation without noise or having less noise.

Figure 5.1: Robot arm detection (1)
Length of the robot arm segments are 220 mm each, this corresponds to 124 pixels from the screen shot measurement. Therefore 85 pixels of the red collision line correspond to 150 mm with an average circle center error of the screenshot of 6 pixels corresponding to an error of 1 cm.

The robot arm is well detected with the system as shown in the figure 5.1. The track bar is at right side of the image. The lower part of the image frame is showing the edges of the image frame. The upper part is showing robot arm segment where the circular joints are detected and their center points are connected. There is a false circle in detected at the bottom of the robot arm. The false circle is shown in red color. Many bounding rectangles (blue boxes in the image) are visible in the image frame, also.

The position of the robot arm in figure 5.2 is different from the figure 5.1. The robot arm segment is well detected here in this image frame also. The false circles in red colour have the same size as the segment circles (circular joints). That is why these circles are drawn.

![Figure 5.2: Robot arm detection (2)](image)

**Figure 5.3** shows the detection of collision. A red line is visible in the image showing the robot arm segments are very close to each other. The yellow lines show the distance between the segment circles. Length of the red line is less than yellow lines is displayed here.
Figure 5.3: Robot arm collision detection (1)

Figure 5.4 shows the detection of collision also but this is from a different angle. A red line is visible in the image showing the robot arm segments are very close to each other.

Figure 5.4: Robot arm collision detection (2)
Figure 5.5 shows the detection of collision also but this is from another angle. A red line in the image frame is showing the robot arm segments are very close to each other. There are so many positions where the segments can come very close to each other. At those positions the red line will alert the user.
6 Summary and outlook

The system is designed for image based monitoring and process control of the degrees of freedom of robot arm. Personal Computer (PC) with webcam is used to develop the software. C++ with OpenCV library for image processing is used for the development of the system using Eclipse IDE (integrated Development Environment). The system is implemented in Linux (Ubuntu 13.1) platform. A model of robot arm is first created and the movements of the model like the real robot arm are done with IK solver using Blender 3D modeling software. All of these softwares are selected as they are open source and free of cost. Especially OpenCV is selected for the development as it is open-source platform dedicated for image processing.

The robot arm segment is a combination of two shapes, circles i.e. circular joints and rectangles i.e. two parts of segments. The robot arm is extracted by combining these two features. Three circular joints are available in the segment. Each part of the segment contains two circular joints at their end points where one joint is common for both of them. The joints are equal in size and the segment parts have a fixed length also. The robot arm segment is determined when three circles enclosing circular joints are found in a user defined length and connection line of the circles are very close to the segment enclosing rectangles. The false circles and rectangles are avoided by the algorithm developed. Collision is detected when two of the circular joints are very close. A red line is drawn between the joints when there is a chance of collision. The system provides a very good graphical user interface for the user. The user can change the value for different parameters through the slider to determine the robot arm from the running video. The output shows good detection of robot arm while testing with the modeling video from Blender. The system is tested with the real video of robot arm captured in laboratory at room 1/024 in Technische Universität Chemnitz. The robot arm is well detected in the video. Moreover, the designed system is very user convenient and cost effective. The system can start by turning on the eclipse and a single click to run the program. The user can observe the output on PC monitor and adjust the parameter values with the slider bar when needed. There is no need of and special training to operate it. Again, Predefined robot arm movement in its space is not required. The designed system can be used for any robot arm which has the same degree of freedom.

This system reduces the cost and ease handling process which is the main goal of this thesis work. The system is a mono camera system, so it is specific by the point of view. When a object is staying in between camera and the robot arm that can make problem for well detection. To avoid this, a research work “Image based distance preference for collision detection of industrial robot” is running by Computer Engineering Professorship at Technische Universität Chemnitz.
Kinect, a line of motion sensing input devices by Microsoft is being used to get the depth information from the image. Further customizations of output are possible, for example with big alert text in case of collision. At the same time faster detection optimization is possible with higher frame rate devices.

In this report selected algorithms are discussed which can help to find different shapes from the image. Combinations of different shapes or features help to identify an object. The segment of a robot-arm is detected here with this thesis work. The main goal for image processing is to get the required information from the image. If it is possible to do the image processing properly, it will help the machine to take the decision based on information from the environment. In medical science image processing can help to identify unexpected objects. For space science it can help to get the images more perfect. Military can identify the enemy and their places well. Automotive industry is developing different applications for driving assistance. Driving will be more comfortable and rate of accidents will decrease a lot with these applications. So many researches are going on this field of study. Image processing is giving eye to machine. To do Master Thesis at university with guidance of Professors was very helpful to me to learn solve practical tasks by own view. Comparing with start point of the thesis I collected many experiences with OpenCV. In the result of this thesis increased my knowledge on image processing algorithms and I had possibility to test with practical task with my knowledge.
Appendix A

Glossary

- **Segment Center**: It is the midpoint of two circles containing robot arm joints.
- **Bounding box Center**: Center point of the bounding box enclosing object.
- **CBS**: Circle Bounding Segment (CBS) preference
- **OpenCV**: Open Source Computer Vision. It is the open source software for developing image processing software.
- **Blender**: 3D Modeling tool
- **IDE**: Integrated Development Environment
- **BSD**: Berkeley Software Distribution. The software license of OpenCV is granted by BSD.
- **LUX**: The lux (symbol: lx) is the unit of illuminance and luminous emittance, measuring luminous flux per unit area.
- **GP2A driver**: It is a driver for proximity and light sensors
- **DIP**: Digital Image Processing.
- **Codec**: Coding/Decoding, The symbolic arrangement of data or instructions in a computer program or set of such instructions
- **JPEG**: Joint Photographic Experts Group
- **ASIC**: Application-Specific Integrated Circuit
- **FPGA**: Field Programmable Gate Array
- **DSP**: Digital signal processing
Appendix B

Useful links

- Web resource to install Ubuntu 13.04 alongside Windows7
  http://askubuntu.com/questions/324578/install-ubuntu-inside-alongside-windows-7
- Web resource to install OpenCV in Linux
  http://docs.opencv.org/2.4/doc/tutorials/introduction/linux_install/linux_install.html#linux-installation
- Using OpenCV with Eclipse (plugin CDT)
  http://docs.opencv.org/2.4/doc/tutorials/introduction/linux_eclipse/linux_eclipse.html#linux-eclipse-usage
- OpenCV image processing tutorial
  http://docs.opencv.org/2.4/doc/tutorials
Bibliography

[1] Clifford A. Shaffer, Gregory M. Herb, *A real-time robot arm collision avoidance system*, Department of Computer Science, Virginia Polytechnic Institute and state University, Virginia


[22] *Industrial Circuits Application Note Stepper Motor Basics*, Solarbotics Ltd


[29] History of Blender, available at: https://www.blender.org/foundation/history/


Image resource:

i. KUKA Industrial robot. Available at:
   http://www.kuka-robotics.com/germany/de/products/industrial_robots

ii. Proximity sensor: The way it works. Available at:
    http://www.techfunzz.science/2014/01/robotics-post1-tsop-based-proximity.html

iii. Stepper motor: cross section. Available at:

iv. Stepper motor: outlook. Available at:
    http://gemsmotor.com/stepper/nema-23-stepper-motor-.png

v. Logo of OpenCV. Available at:
   http://opencv.org/

vi. Logo of MATLAB. Available at:
    https://helpdesk.dei.uc.pt/site/assets/files/2534/matlab.png

vii. Equation of line. Available at:
     http://www-rohan.sdsu.edu/~jmahaffy/courses/f00/math536/static/lstsq/images/bestfit.jpg

viii. Bounding box and extremal points. Available at: