Automated Ripe Fruits Detection and Sorting Technique

A Project Report Submitted in partial fulfilment of the requirements for the Award of the Degree

Bachelor of Technology
Electronics and Communication Engineering

Submitted by

Preeti Vyas 131114005
Suhita Bhatia 131114004
Vibhor Dubey 131114139
Himanshu Raghuvanshi 131114021
Vrushabh Ambade 131114025
Apoorva Manwani 131114053

Automated Ripe Fruits Detection and Sorting Technique

by

Preeti Vyas
Himanshu Raghuvanshi
Suhita Bhatia
Apoorva Manwani
Vrushabh Ambade
Vibhor Dubey

Final year
Bachelor of Technology
Electronics and Communication Engineering
Maulana Azad National Institute of Technology
Bhopal, India

Under the supervision of

Dr. Jyoti Singhai
Professor
Department of Electronics and Communication
Maulana Azad National Institute of Technology
Bhopal, India

ACKNOWLEDGEMENT

This whole project work is done in Electronics and Communication Engineering Department, Maulana Azad National Institute of Technology, Bhopal, India under the supervision of Dr. Jyoti Singhai, Professor, Department of Electronics and Communication Engineering. We express our immense gratitude to our project supervisor for her invaluable support and guidance. We would also like to thank Dr. R.K.Baghel for providing us the research opportunity to explore our horizons. We convey our sincere regards for the ECE department. We would also give gratitude to our undergraduate institute that ensured the best possible help and resources availed for our research project.

Preeti Vyas
131114005

Suhita Bhatia
131114004

Vibhor Dubey
131114139

Himanshu Raghuvanshi
131114021

Apoorva Manwani
131114053

Vrushabh Ambade
131114025

Final year
Bachelor of Technology
Department of Electronics and Communication
Maulana Azad National Institute of Technology
Bhopal, India

CERTIFICATE

This is to certify that the project report entitled - "**Automated Ripe Fruit Detection and Sorting Technique**" submitted by Preeti Vyas (131114005), Himanshu Raghuvanshi (13114021), Suhita Bhatia (13114004), Vrushabh Ambade (131114025), Apoorva Manwani (131114053), Vibhor Dubey (131114139) in partial fulfilment of the requirements for the award of the Degree Bachelor of Technology in Electronics and communication Engineering is a bonafide record of the work carried out under my guidance and supervision at Maulana Azad National Institute of Technology, Bhopal, India.

Dr. Jyoti Singhai
Professor
Department of Electronics and Communication
Maulana Azad National Institute of Technology
Bhopal, India

TABLE OF CONTENTS

LIST OF FIGURES/ TABLES

Chapter 1
INTRODUCTION

Robotics is an emerging field in the research and development. Whenever we talk about the 3D's of processing involving dirty, dull and dangerous task, this comes to us as a solution. We live in an era where every possible aspect of our lives are affected by these automated machines, and researchers are further making the technology viable at every hand. Let it be industries, education system, medical implementations, aviation, marine, and space vehicles, robotics have its vast expanse of every field of technology. Automation is the need of the hour or rather we should term it as a desire to attain self-sustainability in the world of machines. Automated systems have been incorporated into production lines and machines for years. The main purpose of an automated system is to help speed up a process. Tasks that are time-consuming or inconvenient are often incorporated into systems. Some manufacturing companies will work to develop automated systems that can handle jobs that would be difficult for a human to do. Automated systems can be used to handle a wide range of tasks. Systems have key components that allow them to function properly including a control system, a way to interpret and distribute data and a human interface. High-speed robust processors allow the system to process data and control them efficiently at an ever improving speed.

1.1 Rationale of the project

Automation though prominent in major leading firms and bigger organizations lack its coverage in small scale business where workers are incorporated to do such repetitive tasks like sorting, assembly, and placement for bulk orders. These works can be easily done by automated system but it proves really expensive and non-feasible to these small scale workers. Encountering manual work done by workers at whole scale fruit retailer in sorting fruits, we identified automated detection and sorting of fruits as our project rationale. With our project we have tried to propose a technique to sort ripe and raw oranges based on color maturity. Computer vision algorithms and processes are incorporated to develop an automated sorting technique. The technique has been implemented through a 4 DOF arm and a fixed camera setup. Effort has been made to make the system feasible for all kind of industrial settings.

1.2 Need of automated object sorting system

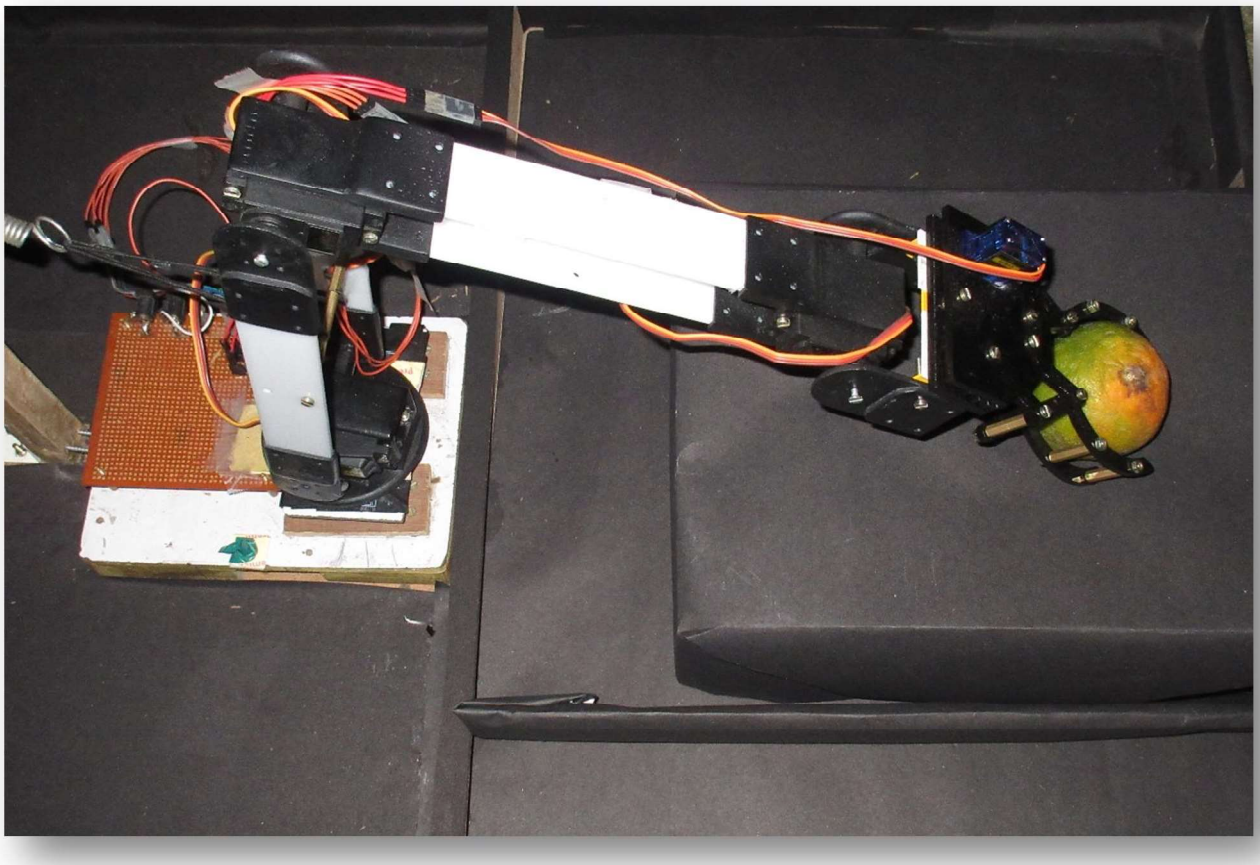
Agriculture is the largest economic sector and it plays the major role in economic development of India. The manual classification and grading techniques which are being used to distinguish between different types of fruits are completely relying on human resource. Since these techniques are guided by human intervention, they are subject to some kind of errors. Since humans are subjected to tiredness and due to the shortage of laborers, the automated system needs to be incorporated to minimize the work and the automated system also helps to reduce the time consumed by manual techniques. Automation in agriculture comes into play to increase productivity, quality and economic growth of the country. Fruit grading is an important process for producers which affects the fruits quality evaluation and export market. Although the grading and sorting can be done by the human, but it is slow labor intensive, error prone and tedious. Hence, there is a need of an intelligent fruit grading system

1.3 Existing technologies & advantages of image-guided sorting

In recent times, various sorting systems have been developed. Every sorting methodology can be classified based on the specification of two issues: (1) the form of the criteria aggregation model which is developed for sorting purposes, and (2) the methodology employed to define the parameters of the sorting model. Few types of research were also based on automatic sorting, manual sorting, and online sorting methods. For example, few researchers propose sorting system that can organize different fruits automatically without human aid, with the use of double acting pneumatic cylinder to push the material to its equivalent boxes on the conveyor belt. Other methods are the di-electrophoresis, morphological transformation of labeling of materials, magnetophoresis, fluorescence activated image segmentation. These proposed sorting methods, however have various problems attributed to them. For example, poor sorting efficiency, energy demand, multi-tasking and machine flexibility. In order to rise above the shortcomings of ever increasing sorting efficiency of materials, conserved energy and improve quality productivities, automatic image guided sorting methods are used.

Chapter 2
AUTOMATED RIPE FRUITS DETECTION AND SORTING TECHNIQUE

The project involves the collaboration of hardware and software to implement a sorting technique. Software part is handled through Computer Vision and various algorithms are used for segmentation, thresholding and contouring the images. The software concepts, algorithms and technologies used is described under Section 2.1. The hardware part is implemented by self-designed 4 DOF robotic arm controlled by servo motors actuated by Raspberry Pi processor. The whole concept of Robotics, Serial Arm Kinematics and other parameters are discussed in Section 2.2. Development of the robotic arm is done through SolidWorks software. Prototyping is done using 3D printing and laser cutting techniques details of which is given in Section 2.3.



2.1 Computer Vision

Computer Vision (CV) is the process of applying a range of technologies and methods to provide imaging-based automatic inspection, process control and robot guidance in industrial applications. While the scope of CV is broad and a comprehensive definition is difficult to distil, a generally accepted definition of computer vision is 'the analysis of images to extract data for controlling a process or activity. Computer vision is a novel technology for acquiring and analyzing an image of a real scene by computers to control machines or to process it. It includes capturing, processing and analyzing images to facilitate the objective and non-destructive assessment of visual quality characteristics in agricultural and food products. The techniques used in image analysis include image acquisition, image pre-processing and image interpretation, leading to quantification and classification of images and objects of interest within images. Images are acquired with a physical image sensor and dedicated computing hardware and software are used to analyze the images with the objective of performing a predefined visual task

2.1.1 Size, shape and color analysis

Size, which is the first parameter identified with quality, has been estimated using machine vision by measuring either projected area, perimeter or diameter. Size measurement is important for determining produce surface area. The shape is one of the important visual quality parameters of fruits, vegetables, etc. Currently, human sorters are employed to sort fruits based on shape. The shape is a feature, easily comprehended by a human but difficult to quantify or define by computer. Most of the machine vision shape detection work has been done on industrial objects, which have a definite structure. Agricultural and biological products are unique in nature and the growing environment causes various boundary irregularities which influence their shapes. Image processing offers a solution for sorting of fruits based on their shape. The color is also an important quality factor that has been widely studied. The color of an object is determined by the wavelength of light reflected from its surface. In biological materials, the light varies widely as a function of wavelength. These spectral variations provide a unique key to machine vision and image analysis. Some fruits have one color homogeneously distributed on the skin surface, which is called primary color. The averaged surface color is a good quality indicator for these fruits. However, other fruits (e.g. some varieties of peaches, apples, tomatoes) have a secondary color that can be used as a good indicator of maturity. In this case, it is not possible to rely only on the global color as a quality parameter.

2.1.2 Image Processing

Image processing in agricultural applications consists of three steps: (1) image enhancement, (2) image feature extraction and (3) image feature classification. Image enhancement is commonly applied to a digital image to correct problems such as poor contrast or noise. Image enhancement procedures such as morphological operations, filters, and pixel-to-pixel operations are generally

used to correct inconsistencies in the acquired images caused by inadequate and/or non-uniform illumination. Statistical procedures from basic image statistics such as mean, standard deviation, and variance to more complex measurement such as principle component analysis can be used to extract features from digital images. For the test fruit image, color and texture features are derived as that of the training phase and compared with corresponding feature values, stored in the feature library. The classification is done using the Minimum Distance Criterion. The image from the training set which has the minimum distance when compared with the test image says that the test image belongs to the category of that training image.

Once image features are identified, the next step is feature classification. Numerical techniques such as neural networks and fuzzy inference systems have been successfully applied to perform image feature classification. Pre-processing refers to the initial processing of the raw image. The images captured or taken are transferred onto a computer and are converted to digital images. Digital images though displayed on the screen as pictures are digits, which are readable by the computer and are converted to tiny dots or picture elements representing the real objects. In some cases pre-processing is done to improve the image quality by suppressing undesired distortions referred to as "noise" or by the enhancement of important features of interest. The intermediate-level processing involves image segmentation, image representation, and image description.

2.1.3 Image Segmentation

Image segmentation is a process of cutting, adding and feature analysis of images aimed at dividing an image into regions that have a strong correlation with objects or areas of interest using the principle of matrix analysis. Image segmentation is one of the most important steps in the entire image processing technique, as subsequently extracted data are highly dependent on the accuracy of this operation. Its main aim is to divide an image into regions that have a strong correlation with objects or areas of interest. If objects in the image cannot be segmented correctly, it is difficult for object measurement; classification and recognition, hence impact interpreting and understanding that image. Earlier studies proposed the use of a 'flooding' algorithm to segment patch-like defects (russet patch, bruise, and also stalk or calyx area). It was found that this method of feature identification is applicable to other types of produce with uniform skin color. This technique was improved by Yang and Marchant, who applied a 'snake' algorithm to closely surround the defects. Segmentation can be achieved by three different techniques: thresholding, edge-based segmentation and region-based segmentation.

Thresholding is a simple and fast technique for characterizing image regions based on constant reflectivity or light absorption of their surfaces. Edge-based segmentation relies on edge detection by edge operators. Edge operators detect discontinuities in gray level, color, texture. Region segmentation involves the grouping together of similar pixels to form regions representing single objects within the image. Raji et al. developed a program in FORTRAN. Computer vision has been used for such tasks as shape classification, defect detection, quality grading and variety classification. A color model developed was used as a standard for comparison with sample images. The developed algorithm gave satisfactory results with well-contrasted defects; however, two further enhancements following segmentation were required to improve accuracy. A novel

adaptive spherical transform was developed and applied in a machine vision defect sorting system. The transform converts a spherical object image to a planar object image allowing fast feature extraction, giving the system an inspection capacity of 3000 apples min-1 from the three cameras, each covering 24 apples in the field of view. A 94% success rate was achieved for sorting defective apples from good ones for the 600 samples tested.

2.1.4 Color Analysis

Color vision can be processed using RGB color space or HSV color space. RGB color space describes colors in terms of the amount of red, green, and blue present. HSV color space describes colors in terms of the Hue, Saturation, and Value. In situations where color description plays an integral role, the HSV color model is often preferred over the RGB model. The HSV model describes colors similarly to how the human eye tends to perceive color. RGB defines color in terms of a combination of primary colors, whereas, HSV describes a color using more familiar comparisons such as color, vibrancy, and brightness. Unlike RGB, HSV separates *luma*, or the image intensity, from *chroma* or the color information. This is very useful in many applications. For example, histogram equalization of a color image is only done on the intensity component, leaving the color components alone. In computer vision, you often want to separate color components from intensity for various reasons, such as robustness to lighting changes, or removing shadows and thus HSV model serves a great help.

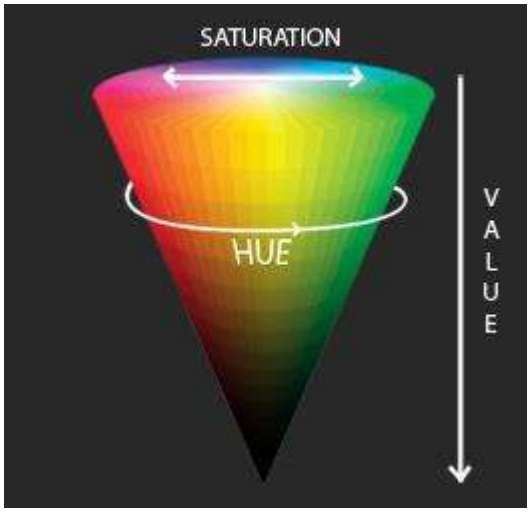


Fig. 5 HSV Loop

Figure 5 illustrates how hue, saturation, and value are defined.

• Hue represents the color type. It can be described in terms of an angle on the above circle. Although a circle contains 360 degrees of rotation, the hue value is normalized to a range from 0 to 255, with 0 being red.

• Saturation represents the vibrancy of the color. Its value ranges from 0 to 255. The lower the saturation value, the more gray is present in the color, causing it to appear faded.

• Value represents the brightness of the color. It ranges from 0 to 255, with 0 being completely dark and 255 being fully bright.

• White has an HSV value of 0-255, 0-255, 255. Black has an HSV value of 0-255, 0-255, 0. The dominant description for black and white is the term, value. The hue and saturation level do not make a difference when value is at max or min intensity level.

The RasPi camera, on the robot, uses the RGB model to determine color. Once the camera has read these values, they are converted to HSV values. The HSV values are then used in the code to determine the location of a specific object/color for which the robot is searching. The pixels are individually checked to determine if they match a predetermined color threshold.

2.1.5 Image Analysis

Image analysis is the process of distinguishing the objects (regions of interest) from the background and producing quantitative information, which is used in the subsequent control systems for decision making. Luis Gracia et al. present the flowchart of the developed application for image analysis as shown in Figure 5. The first step is image acquisition, which includes the software filtering described in subsection "Erode and Dilate process". Next, the image is binarized. After that, the opening/closing operations and the erode/ dilate algorithms are applied, hence only relevant blobs (i.e., the flowers/ fruits/vegetables) remain in the image. The "feature extraction" step obtains the characterization of each detected blobs, i.e. the position of the center of the flower/fruit/vegetable and the cutting point that allows separating the element from its peduncle. Finally, the "movement action" step sends the characterization information of each flower/ fruit/vegetable to the robotic mechanism responsible for the desired task: recollection, cutting, packaging, classification, fumigation, etc.

2.1.6 Thresholding

In OpenCV, value range for HUE, SATURATION and VALUE are respectively 0-179, 0-255 and 0-255. HUE represents the color, SATURATION represents the amount to which that respective color is mixed with white and VALUE represents the amount to which that respective color is mixed with black.
In the application, the orange object has HUE, SATURATION and VALUE in 0-22. Here the HUE is unique for that specific color distribution of that object.
But SATURATION and VALUE may be varied according to the lighting condition of that environment.

Hue values of basic colors

- Orange 0-22
- Yellow 22- 38
- Green 38-75
- Blue 75-130
- Violet 130-160
- Red 160-179

These are approximate values. Exact range of HUE values is to be found according to the color of the object. The range of 13-22 was found perfect for the range of hue values of the object through trial-error calibration. The SATURATION and VALUE depend on the lighting condition of the environment as well as the surface of the object.

After thresholding the image, you'll see small white isolated objects here and there. It may be because of noises in the image or the actual small objects which have the same color as our main object. These unnecessary small white patches can be eliminated by applying **morphological opening**. **The morphological opening** can be achieved by an **erosion**, followed by the dilation with the same structuring element.

The thresholded image may also have small white holes in the main objects here and there. It may be because of noises in the image. These unnecessary small holes in the main object can be eliminated by applying **morphological closing**. **Morphological closing** can be achieved by a **dilation**, followed by the erosion with the same structuring element.

2.1.7 Dilation

This operation consists of convoluting an image A with some kernel (B), which can have any shape or size, usually a square or circle.

- The kernel B has a defined *anchor point*, usually being the center of the kernel.
- As the kernel B is scanned over the image, we compute the maximal pixel value overlapped by B and replace the image pixel in the anchor point position with that maximal value. As you can deduce, this maximizing operation causes bright regions within an image to "grow" (therefore the name *dilation*). Take as an example the image above. Applying dilation we can get:

Fig. 6 Letter J

The background (bright) dilates around the black regions of the letter.
To better grasp the idea and avoid possible confusion, in this another example we have inverted the original image such as the object in white is now the letter. We have performed two dilatations with a rectangular structuring element of size $3x3$.

Fig. 7 Image before and after dilation respectively

The dilatation makes the object in white bigger**.**

2.1.8 Erosion

- This operation is the sister of dilation. What this does is to compute a local minimum over the area of the kernel.
- As the kernel B is scanned over the image, we compute the minimal pixel value overlapped by B and replace the image pixel under the anchor point with that minimal value.
- Analogously to the example for dilation, we can apply the erosion operator to the original image (shown above). You can see the result below that the bright areas of the image (the background, apparently), get thinner, whereas the dark zones (the "writing"(gets bigger.

Fig. 8 Letter J

In the same manner, the corresponding image resulting from the erosion operation on the inverted original image (two erosions with a rectangular structuring element of size $3x3$.

Fig. 9 Image before and after erosion respectively

The erosion makes the object in white smaller.

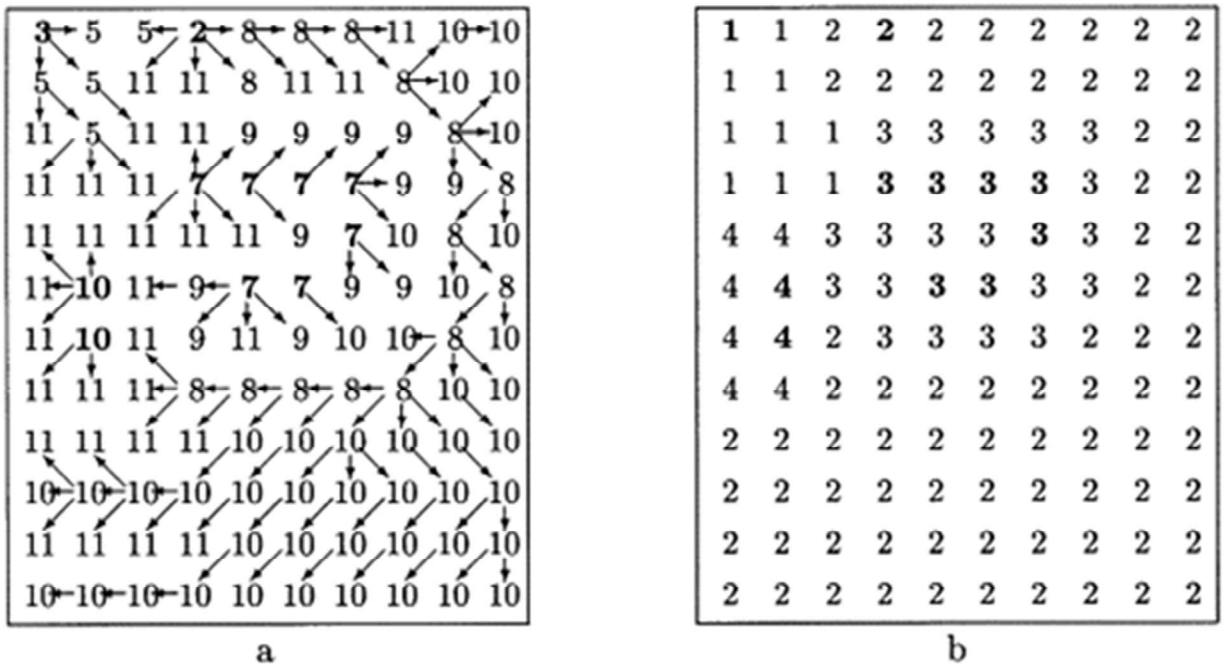
2.1.9 Watershed

The watershed transformation is a popular image segmentation algorithm for grey-scale images. The traditional watershed algorithm simulates a flooding process. Thus, an image is identified with a topographical surface, in which the altitude of every point is equal to the gray level of the corresponding pixel. Holes are then pierced in all regional minima of the relief (connected plateaus of constant altitude from which it is impossible to reach a location of lower altitude without having to climb). Sinking the whole surface slowly into a lake, water springs through the holes and progressively immerses the adjacent walls. To prevent streams of water coming from different holes to intermingle, a hinder is set up at the meeting locations. Once the relief is completely covered by water, the set of obstacles depicts the watershed image.

Various definitions of watersheds have been proposed in the literature for both digital and continuous spaces. Most algorithms label each pixel with the identifier of its catchment basin and no watershed lines are explicitly constructed. In this paper, we present a new algorithm to perform the watershed transformation which does not construct watershed lines. Let us mention that the algorithm produces the same segmentation result as the techniques in Refs. , but a simpler algorithmic construction and hence a lower complexity is issued.

The traditional implementation of the watershed segmentation algorithm simulates the flooding process over the image surface. First, regional minima are detected and uniquely labeled with integer values. Then, the algorithm simulates the flooding process using a hierarchical queue. Such a queue consists of H first-in–first-out (FIFO) queues, one queue for each of the H gray levels in the image; the size of the hth FIFO queue is given by the number of pixels in the image having the grey-level h. This data structure is used to impose the order of accessing pixels to operate on. Initially, the hierarchical queue contains the seeds for the flooding, i.e. the minima which are at the interface line between the regional minima and the non-minim

a pixels; a pixel of grey-level h is introduced into the hth FIFO queue of the hierarchical queue. The hierarchical queue is then parsed from the lowest grey level to the highest one. A pixel p, removed from the queue, propagates its label to all its neighbors which have not been already reached by flooding. The latter are introduced, at their turn, into the queue of their grey level. The FIFO order of serving the candidate pixels within the same connected plateau ensures the synchronous breadth-first propagation of labels coming from different minima inside a plateau. When all FIFO queues have been emptied, each pixel was appended to a single region and the procedure stops. The image of labeled pixels depicts the segmentation result. For a simple input image, the flooding process, illustrated by arrows, is shown in Fig. 2.



Following the flowing scheme in **Fig. 10**, we developed a formalism which allows us to determine for every pixel p a neighboring pixel q from which p will be flooded. As in other watershed formalisms, q may not be unique. In such a case, q is arbitrarily chosen among the potential pixels. Having this local "connectivity" relation, between neighboring pixels which pertain to the same catchment basin, embedded into the image (technique also known as arrowing), the result is nothing but a directed graph, for which the connected components, must be computed. The novelty of our approach is to effectively apply the connected component operator, to compute catchment basins.

2.1.10 Watershed in Open CV

Any grayscale image can be viewed as a topographic surface where high intensity denotes peaks and hills while low intensity denotes valleys. You start filling every isolated valleys (local minima) with different colored water (labels). As the water rises, depending on the peaks (gradients) nearby, water from different valleys, obviously with different colors will start to merge. To avoid that, you build barriers in the locations where water merges. You continue the work of filling water and building barriers until all the peaks are under water. Then the barriers you created gives you the segmentation result. This is the "philosophy" behind the watershed.

But this approach gives you over segmented result due to noise or any other irregularities in the image. So OpenCV implemented a marker-based watershed algorithm where you specify which are all valley points are to be merged and which are not. It is an interactive image segmentation. What we do is to give different labels for our object we know. Label the region which we are sure of being the foreground or object with one color (or intensity), label the region which we are sure of being background or non-object with another color and finally the region which we are not sure

of anything, label it with 0. That is our marker. Then apply watershed algorithm. Then our marker will be updated with the labels we gave, and the boundaries of objects will have a value of -1.

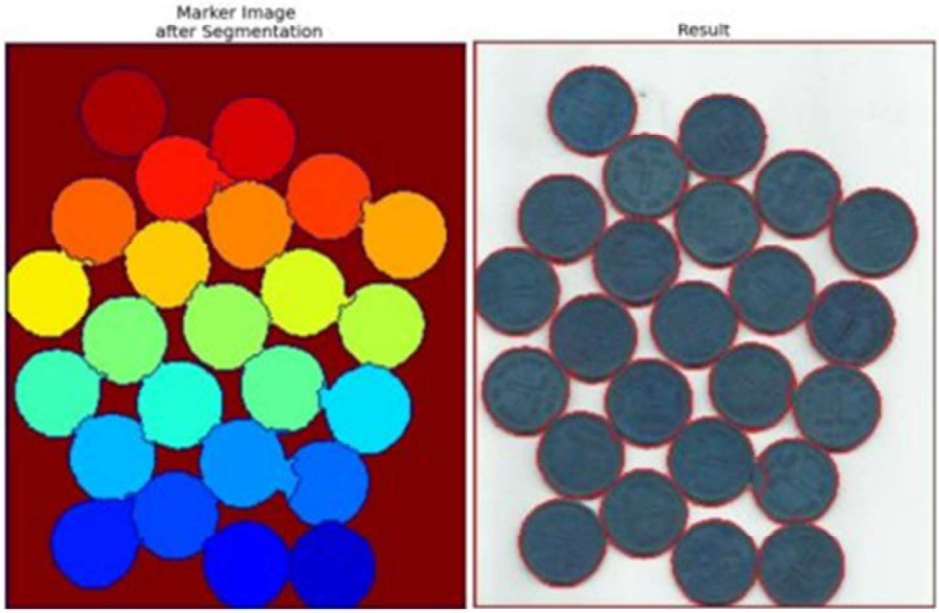


Fig. 11 Watershed Algorithm_2

2.1.11 Perspective Shift Algorithm

The algorithm under this section perform various geometrical transformations of 2D images. They do not change the image content but deform the pixel grid and map this deformed grid to the destination image. In fact, to avoid sampling artifacts, the mapping is done in the reverse order, from destination to the source. That is, for each pixel (x, y) of the destination image, the functions compute coordinates of the corresponding "donor" pixel in the source image and copy the pixel value:

$$\texttt{dst}(x,y) = \texttt{src}(f_x(x,y), f_y(x,y))$$

In case when you specify the forward mapping $\langle g_x, g_y \rangle : \texttt{src} \rightarrow \texttt{dst}$, the OpenCV functions first compute the corresponding inverse mapping $\langle f_x, f_y \rangle : \texttt{dst} \rightarrow \texttt{src}$ and then use the above formula.

The actual implementations of the geometrical transformations, from the most generic remap() and to the simplest and the fastest resize() , need to solve two main problems with the above formula:

- Extrapolation of non-existing pixels. Similarly to the filtering functions described in the previous section, for some (x, y), either one of $f_x(x, y)$, or $f_y(x, y)$, or both of them may fall outside of the image. In this case, an extrapolation method needs to be used. OpenCV provides the same selection of extrapolation methods as in the filtering functions. In addition, it provides the method BORDER TRANSPARENT. This means that the corresponding pixels in the destination image will not be modified at all.

- Interpolation of pixel values. Usually $f_x(x, y)$ and $f_y(x, y)$ are floating-point numbers. This means that $\langle f_x, f_y \rangle$ can be either an affine or perspective transformation, or radial lens distortion correction, and so on. So, a pixel value at fractional coordinates needs to be retrieved. In the simplest case, the coordinates can be just rounded to the nearest integer coordinates and the corresponding pixel can be used. This is called a nearest-neighbor interpolation. However, a better result can be achieved by using more sophisticated interpolation methods , where a polynomial function is fit into some neighborhood of the computed pixel $(f_x(x, y), f_y(x, y))$, and then the value of the polynomial at $(f_x(x, y), f_y(x, y))$ is taken as the interpolated pixel value. In OpenCV, you can choose between several interpolation methods.

Get Perspective function from OpenCV calculates a perspective transform from four pairs of the corresponding points.

The function calculates the 3×3 matrix of a perspective transform so that:

$$\begin{bmatrix} t_i x'_i \\ t_i y'_i \\ t_i \end{bmatrix} = \texttt{map\_matrix} \cdot \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

$$\texttt{dst}(i) = (x'_i, y'_i), \texttt{src}(i) = (x_i, y_i), i = 0, 1, 2, 3$$

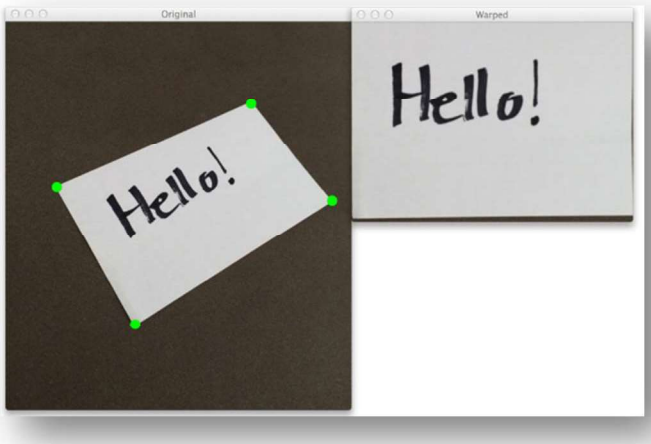


Fig. 12 Perspective shift

2.2 Robotic Arm

An industrial robot is a robot system used for manufacturing. Industrial robots are automated, programmable and capable of movement on two or more axes. Typical applications of robots include welding, painting, assembly, pick and place for printed circuit boards, packaging and labeling, palletizing, product inspection, and testing; all accomplished with high endurance, speed, and precision. They can help in material handling and provide interfaces.

2.2.1 Types and features

The most commonly used robot configurations are articulated robots, SCARA robots, delta robots and Cartesian coordinate robots, (gantry robots or x-y-z robots). In the context of general robotics, most types of robots would fall into the category of robotic arms (inherent in the use of the word manipulator in ISO standard 1738). Robots exhibit varying degrees of autonomy:

- Some robots are programmed to faithfully carry out specific actions over and over again (repetitive actions) without variation and with a high degree of accuracy. These actions are determined by programmed routines that specify the direction, acceleration, velocity, deceleration, and distance of a series of coordinated motions.

- Other robots are much more flexible as to the orientation of the object on which they are operating or even the task that has to be performed on the object itself, which the robot may even need to identify. For example, for more precise guidance, robots often contain machine vision sub-systems acting as their visual sensors, linked to powerful computers or controllers. Artificial intelligence, or what passes for it, is becoming an increasingly important factor in the modern industrial robot.

2.2.2 Defining Parameters

- **Number of axes –** two axes are required to reach any point in a plane; three axes are required to reach any point in space. To fully control the orientation of the end of the arm (i.e. the wrist) three more axes (yaw, pitch, and roll) are required. Some designs (e.g. the SCARA robot) trade limitations in motion possibilities for cost, speed, and accuracy
- **Degrees of freedom –** this is usually the same as the number of axes.
- **Working envelope –** the region of space a robot can reach.
- **Kinematics –** the actual arrangement of rigid members and joints in the robot, which determines the robot's possible motions. Classes of robot kinematics include articulated, Cartesian, parallel and SCARA.
- **Carrying capacity or payload –** how much weight a robot can lift.

- **Speed –** how fast the robot can position the end of its arm. This may be defined in terms of the angular or linear speed of each axis or as a compound speed i.e. the speed of the end of the arm when all axes are moving.
- **Acceleration –** how quickly an axis can accelerate. Since this is a limiting factor a robot may not be able to reach its specified maximum speed for movements over a short distance or a complex path requiring frequent changes of direction.
- **Accuracy –** how closely a robot can reach a commanded position. When the absolute position of the robot is measured and compared to the commanded position the error is a measure of accuracy. Accuracy can be improved with external sensing, for example, a vision system or Infra-Red. See robot calibration. Accuracy can vary with speed and position within the working envelope and with payload (see compliance).
- **Repeatability –** how well the robot will return to a programmed position. This is not the same as accuracy. It may be that when told to go to a certain X-Y-Z position that it gets only to within 1 mm of that position. This would be its accuracy which may be improved by calibration. But if that position is taught into controller memory and each time it is sent there it returns to within 0.1mm of the taught position then the repeatability will be within 0.1mm.

2.2.3 Robot Kinematics

Kinematics is the branch of science which deals with motion without considering the forces which cause the motion. Robot kinematics applies geometry to the study of the movement of multi-degree of freedom kinematic chains that form the structure of robotic systems.[1][2]The emphasis on geometry means that the links of the robot are modeled as rigid bodies and its joints are assumed to provide pure rotation or translation. For doing a kinematic analysis of any model we need to overview kinematic links, kinematic chains, mechanism and its inversion for getting real life application from it.

2.2.4 Kinematics link and pair

- **Link** It is defined as a machine element having relative motion with respect to other parts of the machine element.
- **Kinematic pair** is defined as a joint of two links which permit relative motion between elements or links. Kinematic pair is constituted when there is mechanical contact between them and having an arrangement of the mechanism.
- **Lower pair** is said to be constituted when two elements have surface contact between them.
- **Higher pair** is said to be constituted when two elements has point contact between them.

2.2.5 Degree of freedom of any mechanism

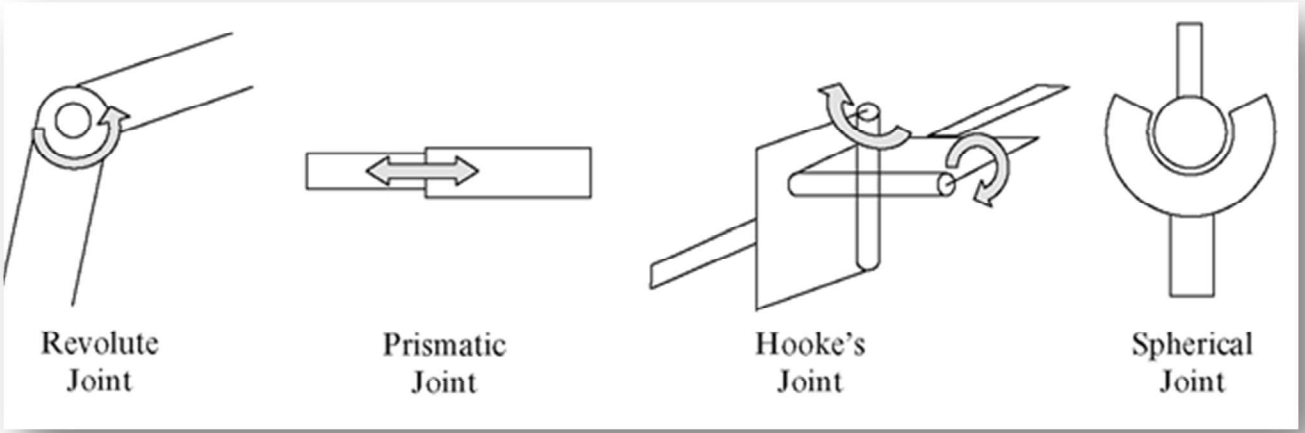
The definition of the degrees of freedom of a mechanism is the number of independent relative motions among the rigid bodies. The term kinematic pairs actually refer to kinematic constraints between rigid bodies. Kinematic constraints are constraints between rigid bodies that result in the decrease of the degrees of freedom of rigid body system. The kinematic pairs are divided into lower pairs and higher pairs, depending on how the two bodies are in contact. There are two kinds of lower pairs in planar mechanisms - revolute pairs and prismatic pairs. A revolute pair keeps the axes of two rigid bodies together. A prismatic pair keeps two axes of two rigid bodies align and allow no relative rotation. Two rigid bodies constrained by this kind of constraint will be able to have an independent translational motion along the axis. In general, a rigid body in a plane has three degrees of freedom. Kinematic pairs are constraints on rigid bodies that reduce the degrees of freedom of a mechanism. Figure 4-11 shows the three kinds of pairs in planar mechanisms. These pairs reduce the number of the degrees of freedom. If we create a lower pair (Figure 4-11a,b), the degrees of freedom are reduced to 2. Similarly, if we create a higher pair (Figure 4-11c), the degrees of freedom are reduced to 1.

Therefore, we can write the following equation:

$$F = 3(n-1) - 2l - h$$

Where,

F = total degrees of freedom in the mechanism
n = number of links (including the frame)
l = number of lower pairs (one degree of freedom)
h = number of higher pairs (two degrees of freedom)
This equation is also known as Gruebler's equation



2.2.6 Types of Kinematic Machines

2.2.6.1. PARALLEL KINEMATIC MACHINES

A parallel kinetic machine is a mechanical system that uses several controlled serial chains to support a single platform or end-effector. Also known as parallel robots, or generalized Stewart platforms (in the Stewart platform, the actuators are paired together on both the basis and the platform), these systems are articulated robots that use similar mechanisms for the movement of either the robot on its base or one or more manipulator arms.

Fig. 1 Parallel Kinematic Machine

Their 'parallel' distinction, as opposed to a serial manipulator, is that the end effector (or 'hand') of this linkage (or 'arm') is connected to its base by a number of (usually three or six) separate and independent linkages working in parallel.

2.2.6.2. SERIAL KINEMATIC MACHINE

Serial kinematics machine is the most common industrial machine. They are designed as a series of links connected by motor-actuated joints that extend from a base to an end-effector. Often they have an anthropomorphic arm structure described as having a "shoulder", an "elbow", and a "wrist". The main advantage of a serial manipulator is a large workspace with respect to the size of the robot and the floor space it occupies. The main disadvantages of these robots are:

- The low stiffness inherent to an open kinematic structure
- Errors are accumulated and amplified from link to link
- The fact that they have to carry and move the large weight of most of the actuators

Fig. 2 Serial Kinematic Machine

2.2.7 Comparison between PKMs and SKMs

Feature	***Serial robot***	***Parallel Robot***
Workspace	*Large*	*Small and complex*
Solving forward kinematics	*Easy*	*Very difficult*
Solving inverse kinematics	*Difficult*	*Easy*
Position error	*Accumulates*	*Averages*
Force error	*Averages*	*Accumulates*
Maximum force	*Limited by minimum actuator force*	*Summation of all actuator forces*
Stiffness	*Low*	*High*
Dynamics characteristics	*Poor, especially with increasing the size*	*Very high*
Modeling and solving dynamics	*Relatively simple*	*Very complex*
Inertia	*Large*	*Small*
Areas of application	*A great number*	*Currently limited*
Payload/weight ratio	*Low*	*High*
Speed and acceleration	*Low*	*High*
Accuracy	*Low*	*High*
Uniformity of components	*Low*	*High*
Calibration	*Relatively simple*	*Complicated*
Workspace/robot size ratio	*High*	*Low*

Table 1. Comparison between PKMs and SKMs

Rigidity is the important criteria for most of the applications, serial kinematic machines or serial robots achieve this by using high quality rotary joints permitting movement in one axis but rigid outside this. A movement requiring several axes then involves several actuated joints. The unwanted flexibility and sloppiness in one joint cause an extended sloppiness in the arm and the error at the initial point amplifies and prevent precise movement. The inevitable hysteresis and off axis flexibility accumulates along the arm's kinematic chain. A precision serial manipulator is a compromise between precision, complexity, mass (of the manipulator and of the manipulated objects) and cost. While talking about parallel kinematic machines, we have high rigidity with a small mass and the comparatively lesser number of actuated joints. This allows high precision and high speed of movements and motivates the use of parallel manipulators

However, these parallel manipulators suffer a drawback of limited workspace. The workspace is limited by the geometrical and mechanical limits of the design i.e. collisions between legs maximal and minimal lengths of the legs. The workspace is also limited by the existence of singularities, which are positions where, for some trajectories of the movement, the variation of the lengths of the legs is infinitely smaller than the variation of the position. Conversely, at a singular position, a force (like gravity) applied on the end-effector induce infinitely large constraints on the legs, which may damage the manipulator.

2.2.8 Forward Kinematics

Forward kinematics specifies the joint parameters and computes the configuration of the chain. For serial manipulators, this is achieved by direct substitution of the joint parameters into the forward kinematics equations for the serial chain. For parallel manipulators substitution of the joint parameters into the kinematics equations requires the solution of the set of polynomial constraints to determine the set of possible end-effector locations.

2.2.9 Inverse Kinematics

Inverse kinematics specifies the end-effector location and computes the associated joint angles. For serial manipulators, this requires the solution of a set of polynomials obtained from the kinematics equations and yields multiple configurations for the chain. The case of a general 6R serial manipulator (a serial chain with six revolute joints) yields sixteen different inverse kinematics solutions, which are solutions of a sixteenth-degree polynomial. For parallel manipulators, the specification of the end-effector location simplifies the kinematics equations, which yields formulas for the joint parameters.

2.2.10 Kinematic Analysis

The parallel part is same as that of the 3RRR parallel planar robot, we applied inverse kinematics for it. However, for the serial model, we have to use the forward kinematics and some geometrical analysis discussed below. Geometric solution approach is based on decomposing the spatial geometry of the manipulator into several plane geometry problems. It is applied to the simple robot structures, such as, 2-DOF planer manipulator whose joints are both revolute and link lengths are l1 and l2 shown in Figure 5a. Consider Figure 5b in order to derive the kinematics equations for the planar manipulator. The components of the point P (px and py) are determined as follows.

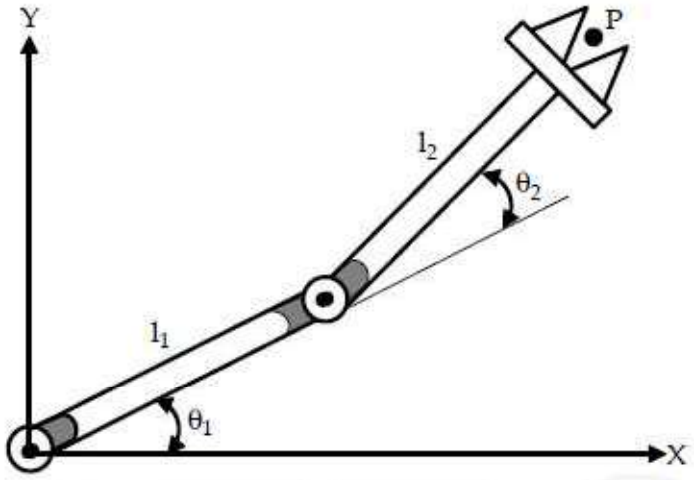


Fig. 3 Serial Arm Manipulator

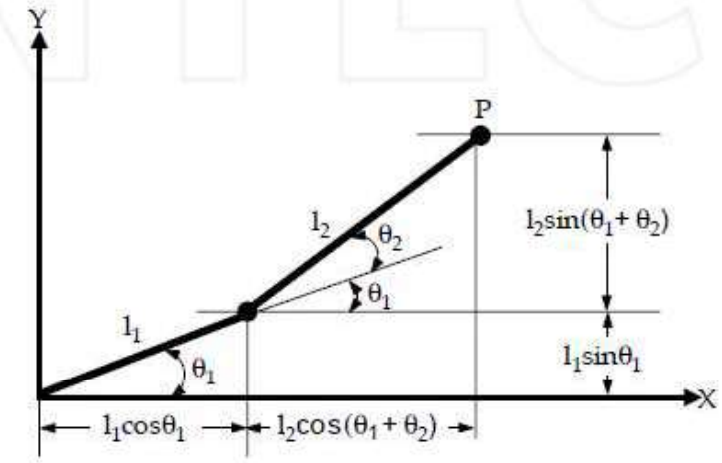


Fig. 4 Geometry based calculations

$$p_x = l_1 c\theta_1 + l_2 c\theta_{12}$$

$$p_y = l_1 s\theta_1 + l_2 s\theta_{12}$$

The solutions computed from summation of squaring both equations, where,

$$c\theta_{12} = c\theta_1 c\theta_2 - s\theta_1 s\theta_2$$

$$s\theta_{12} = s\theta_1 c\theta_2 + c\theta_1 s\theta_2$$

$$p_x^2 = l_1^2 c^2\theta_1 + l_2^2 c^2\theta_{12} + 2l_1 l_2 c\theta_1 c\theta_{12}$$

$$p_y^2 = l_1^2 s^2\theta_1 + l_2^2 s^2\theta_{12} + 2l_1 l_2 s\theta_1 s\theta_{12}$$

$$p_x^2 + p_y^2 = l_1^2 (c^2\theta_1 + s^2\theta_1) + l_2^2 (c^2\theta_{12} + s^2\theta_{12}) + 2l_1 l_2 (c\theta_1 c\theta_{12} + s\theta_1 s\theta_{12})$$

$$p_x^2 + p_y^2 = l_1^2 + l_2^2 + 2l_1 l_2 (c\theta_1 [c\theta_1 c\theta_2 - s\theta_1 s\theta_2] + s\theta_1 [s\theta_1 c\theta_2 + c\theta_1 s\theta_2])$$

$$p_x^2 + p_y^2 = l_1^2 + l_2^2 + 2l_1 l_2 (c^2\theta_1 c\theta_2 - c\theta_1 s\theta_1 s\theta_2 + s^2\theta_1 c\theta_2 + c\theta_1 s\theta_1 s\theta_2)$$

$$p_x^2 + p_y^2 = l_1^2 + l_2^2 + 2l_1 l_2 (c\theta_2 [c^2\theta_1 + s^2\theta_1])$$

$$p_x^2 + p_y^2 = l_1^2 + l_2^2 + 2l_1 l_2 c\theta_2$$

$$c\theta_2 = \frac{p_x^2 + p_y^2 - l_1^2 - l_2^2}{2l_1 l_2}$$

$$s\theta_2 = \pm\sqrt{1 - \left(\frac{p_x^2 + p_y^2 - l_1^2 - l_2^2}{2l_1 l_2}\right)^2}$$

$$\theta_2 = A\tan 2\left(\pm\sqrt{1 - \left(\frac{p_x^2 + p_y^2 - l_1^2 - l_2^2}{2l_1 l_2}\right)^2}, \frac{p_x^2 + p_y^2 - l_1^2 - l_2^2}{2l_1 l_2}\right)$$

$$c\theta_1 p_x = l_1 c^2\theta_1 + l_2 c^2\theta_1 c\theta_2 - l_2 c\theta_1 s\theta_1 s\theta_2$$

$$s\theta_1 p_y = l_1 s^2\theta_1 + l_2 s^2\theta_1 c\theta_2 + l_2 s\theta_1 c\theta_1 s\theta_2$$

$$c\theta_1 p_x + s\theta_1 p_y = l_1 (c^2\theta_1 + s^2\theta_1) + l_2 c\theta_2 (c^2\theta_1 + s^2\theta_1)$$

$$c\theta_1 p_x + s\theta_1 p_y = l_1 + l_2 c\theta_2$$

$$-s\theta_1 p_x = -l_1 s\theta_1 c\theta_1 - l_2 s\theta_1 c\theta_1 c\theta_2 + l_2 s^2\theta_1 s\theta_2$$
$$c\theta_1 p_y = l_1 s\theta_1 c\theta_1 + l_2 c\theta_1 s\theta_1 c\theta_2 + l_2 c^2\theta_1 s\theta_2$$
$$-s\theta_1 p_x + c\theta_1 p_y = l_2 s\theta_2 (c^2\theta_1 + s^2\theta_1)$$

$$-s\theta_1 p_x + c\theta_1 p_y = l_2 s\theta_2$$

$$c\theta_1 p_x^2 + s\theta_1 p_x p_y = p_x (l_1 + l_2 c\theta_2)$$
$$-s\theta_1 p_x p_y + c\theta_1 p_y^2 = p_y l_2 s\theta_2$$
$$c\theta_1 (p_x^2 + p_y^2) = p_x (l_1 + l_2 c\theta_2) + p_y l_2 s\theta_2$$

$$c\theta_1 = \frac{p_x (l_1 + l_2 c\theta_2) + p_y l_2 s\theta_2}{p_x^2 + p_y^2}$$

$$s\theta_1 = \pm\sqrt{1 - \left(\frac{p_x (l_1 + l_2 c\theta_2) + p_y l_2 s\theta_2}{p_x^2 + p_y^2} \right)^2}$$

$$\theta_1 = A\tan 2\left(\pm\sqrt{1 - \left(\frac{p_x (l_1 + l_2 c\theta_2) + p_y l_2 s\theta_2}{p_x^2 + p_y^2} \right)^2}, \frac{p_x (l_1 + l_2 c\theta_2) + p_y l_2 s\theta_2}{p_x^2 + p_y^2} \right)$$

Although the planar manipulator has a very simple structure, as can be seen, its inverse kinematics solution based on geometric approach is very cumbersome. However using software like Mathematica and MATLAB it can be done easily.

2.2.11 Load Analysis

Torque (**T**) is defined as a turning or twisting "force" and is calculated using the following relation:

$T = F \times L$

The force (**F**) acts at a length (**L**) from a pivot point. In a vertical plane, the force acting on an object (causing it to fall) is the acceleration due to gravity (**g** = 9.81m/s^2) multiplied by its mass:

$F = m \times g$

The force above is also considered the object's weight (**W**).

$W = m \times g$

The torque required to hold a mass at a given distance from a pivot is therefore - $T = (m \times g) \times L$
This can be found similarly by doing a torque balance about a point. Therefore, replacing F with m*g, we find the same equation above. This method is the more accurate way to find torque (using a torque balance).

In order to estimate the torque required at each joint, we must choose the worst case scenario.
In the above image, a link of length L is rotated clockwise. Only the perpendicular component of length between the pivot and the force is taken into account. We observe that this distance decreases from L3 to L1 (L1 being zero).

Since the equation for torque is length (or distance) multiplied by the force, the greatest value will be obtained using L3, since F does not change. You can similarly rotate the link counterclockwise and observe the same effect. It can be safe to assume that the actuators in the arm will be subjected to the highest torque when the arm is stretched horizontally. Although your robot may never be designed to encounter this scenario, it should not fail under its own weight if stretched horizontally without a load.

The weight of the object (the "load") being held (**A1** in the diagram), multiplied by the distance between its center of mass and the pivot gives the torque required at the pivot. The tool takes into consideration that the links may have a significant weight (**W1**, **W2**..) and assumes its center of mass is located at roughly the center of its length. The torques caused by these different masses must be added.

2.3 Design

2.3.1 SolidWorks

SolidWorks (stylized as **SOLIDWORKS**) is a solid modeling computer-aided design (CAD) and computer-aided engineering (CAE) computer program. SolidWorks is a solid modeler, and utilizes a par approach to create models and assemblies. The software is written on Parasolid-kernel ametric feature-based. *Parameters* refer to constraints whose values determine the shape or geometry of the model or assembly. Parameters can be either numeric parameters, such as line lengths or circle diameters, or geometric parameters, such as tangent, parallel, concentric, horizontal or vertical, etc. Numeric parameters can be associated with each other through the use of relations, which allows them to capture design intent. *Design intent* is how the creator of the part wants it to respond to changes and updates. *Features* refer to the building blocks of the part. They are the shapes and operations that construct the part.

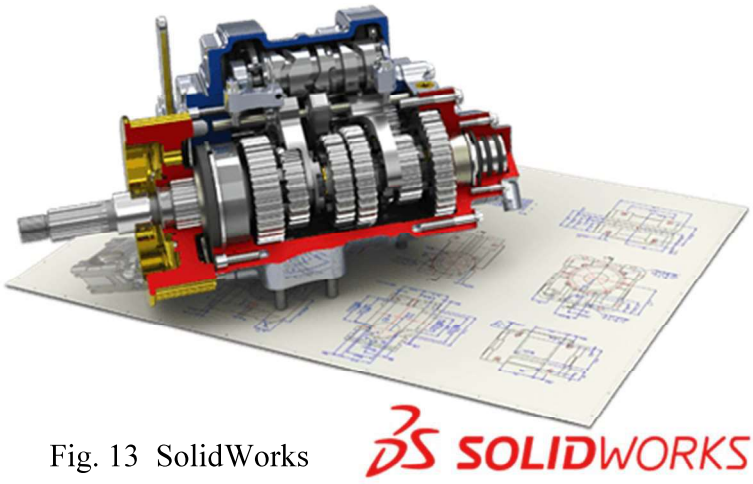


Fig. 13 SolidWorks

2.3.2 3D Printing

3D printing is also known as desktop fabrication or additive manufacturing. It is a prototyping process whereby a real object is created from a 3D design. The digital 3D-model is saved in STL format and then sent to a 3D printer. The 3D printer then prints the design layer by layer and forms a real object. There are several different 3D printing technologies. The main differences are how layers are built to create parts.

SLS (selective laser sintering), FDM (fused deposition modeling) & SLA (Stereo-lithography) are the most widely used technologies for 3D printing. Selective laser sintering (SLS) and fused deposition modeling (FDM) use melted or softened materials to produce layers.

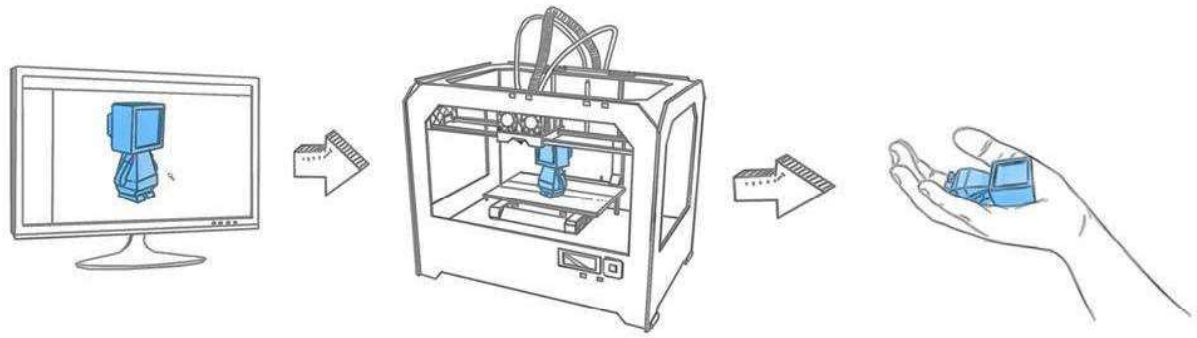


Fig. 14 3D printing in steps

Fig. 15 A commercial 3D printer

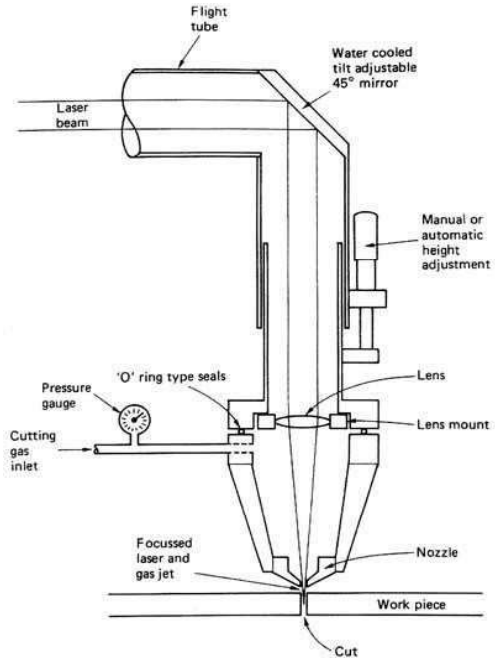


Fig. 16 Laser cutting jet

2.3.3 Laser Cutting

Laser cutting is a technology that uses a laser to cut materials, and is typically used for industrial manufacturing applications. Laser cutting works by directing the output of a high-power laser most commonly through optics. The laser optics and CNC (computer numerical control) are used to direct the material or the laser beam generated. A typical commercial laser for cutting materials would involve a motion control system to follow a CNC or G-code of the pattern to be cut onto the material. The focused laser beam is directed at the material, which then either melts, burns, vaporizes away, or is blown away by a jet of gas,[1] leaving an edge with a high-quality surface finish.

Chapter 3
IMPLEMENTATION

In this project the Software and hardware part is managed in a parallel manner, interfaced using Raspberry Pi processer. SSH techniques is used to get inside the Raspberry Pi through the remote desktop. The technique of detection and sorting is achieved by image processing through OpenCV in Python language. The whole code architecture is firmly designed and presented in Section 3.1.1. The flow chart of the used algorithm is presented in Section 3.1.2. The whole developed Python code is present in Section 3.1.3 with description. Hardware part is thoroughly discussed in Section 3.2. The load analysis and MATLAB analysis is described in full so as to get the reader through the model development phase (Section 3.2.1 and 3.2.2). The components used are described with their specification (Section 3.3) and further designs of the final assembly is given part by part in Section 3.4. The photo gallery of the developed prototype is in Section 3.4.4. and the detailed circuit diagram of the electronic components is in Section 3.4.5.

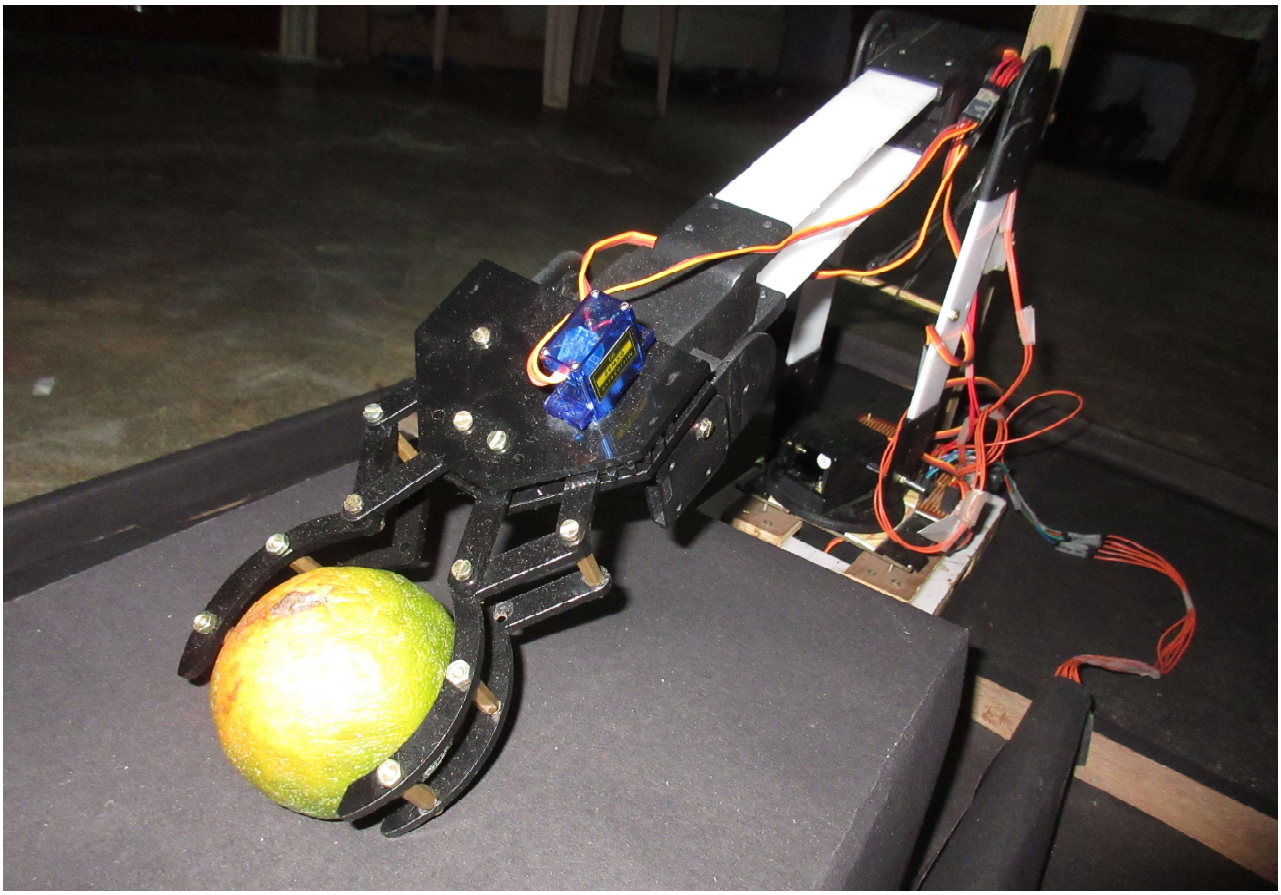
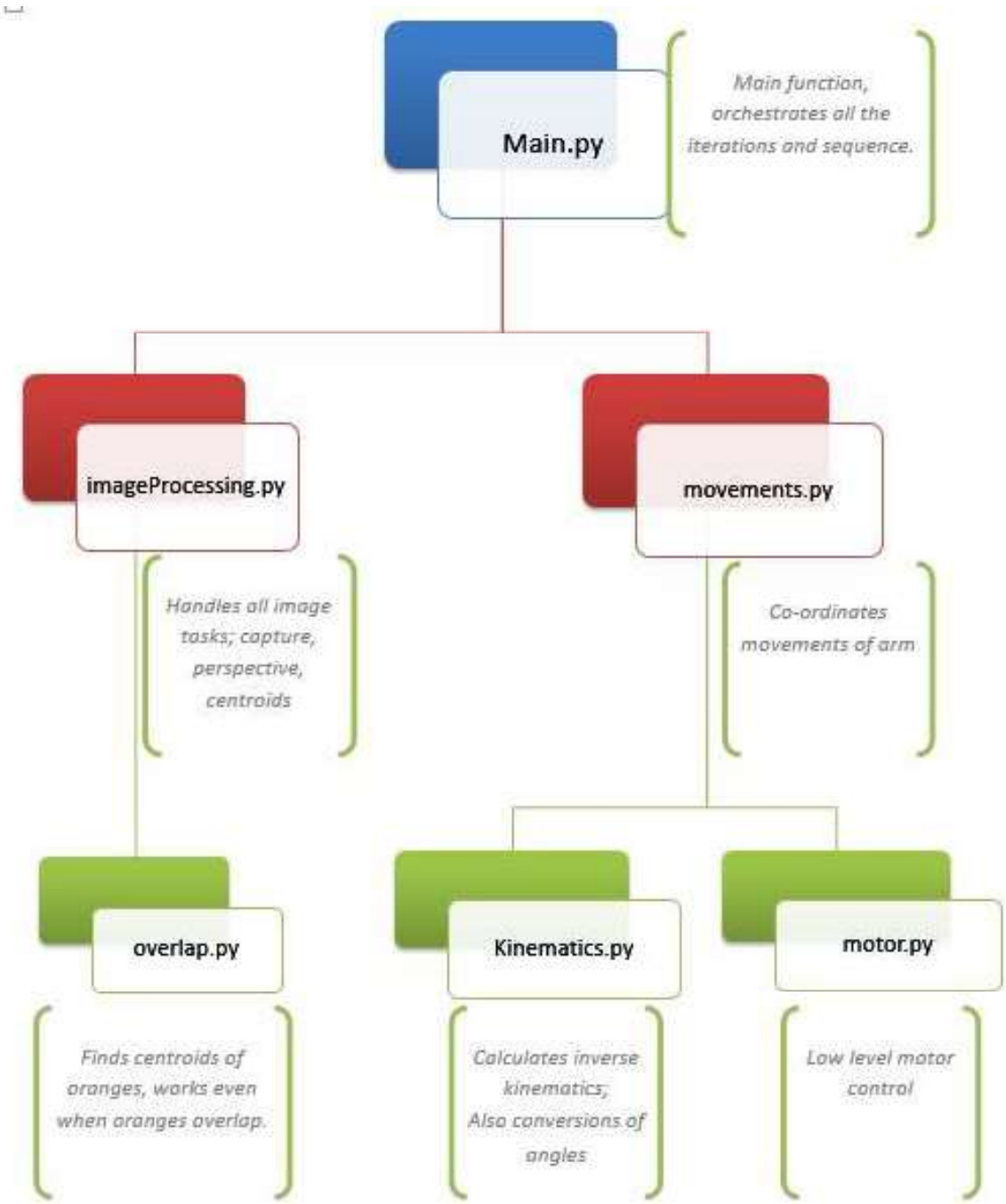


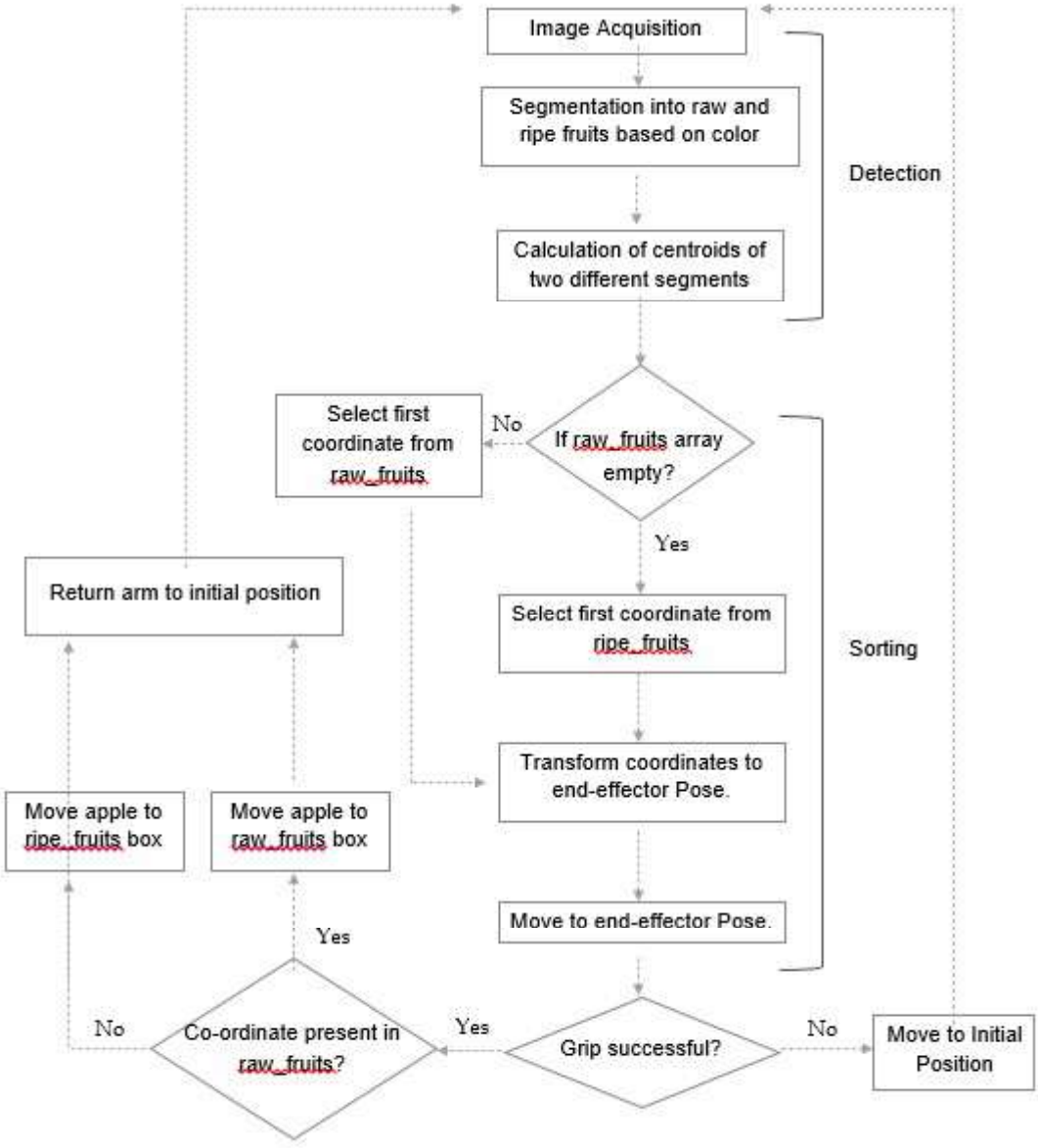
Fig. 17 Complete Robotic Arm Assembly

3.1 Software

3.1.1 Code Architecture



3.1.2 Algorithm Flow Chart

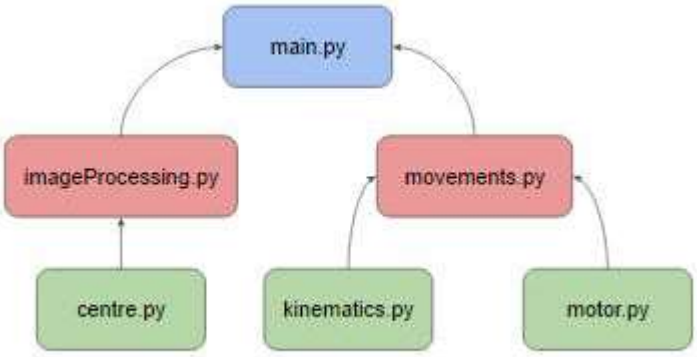


3.1.3 Code

The software part is handled fully by python by using OpenCv libraries. Contouring is done involving Watershed algorithm and for servo motor control PiGPIO libraries are used. The flow of code is as under.

- Capture Image.
- Apply perspective shift, smoothing
- Use segmentation techniques to separate into two images:
  - Green : for raw fruits
  - Orange : for ripe fruits
- Calculate centroids of each fruits in respective images and store them separately
  - connected-component analysis (blob detection)
- First pick up all the raw_fruits and place in Destination 1 (hard coded coordinate for raw fruits box.)
  - Use inverse kinematics equation to correctly move to the required place
- Then pick the ripe_fruits and place in Destination 2 (hard coded coordinate for ripe fruits box.)
- To achieve the desired performance, we have developed a modular code structure to handle two different parts of the code.
  - Image Processing
  - Motor Control + Kinematics

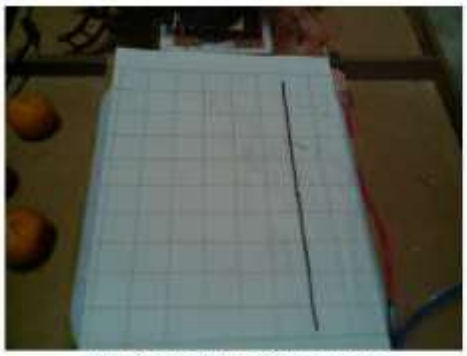
Final code comprises of



- **main.py** : This file integrates the working of different modules and define the iterations and sequence of working.
- **imageProcesssing.py** : This file handles the image operations : image capture, perspective shift, and segmentation.
- **centre.py** : This file calculates the centers of different segments from a binary image. Uses watershed algorithm.
- **movements.py** : This file is used to coordinate the movement of the arm.
- **kinematics.py** : This file calculated the inverse kinematics and handles the coordinate transformation into world coordinate frame.
- **motor.py** : This file is used for low level motor control.

Perspective Correction

Since the camera is mounted 50 cm above the ground plane, the image that it takes are warped due to perspective shift. To correctly find the position of oranges, we need to apply a perspective correction to the raw image taken.
getPerspectiveTransform() function of the OpenCV library takes in 4 points as input to give a transformation matrix to warp the image to give a correct perspective. Then, a non-affine transformation is applied to warp the image.



Raw image taken from camera.

Corrected perspective

Centroid Detection

To command the arm to pick up the oranges, we needed their coordinate position from the camera image. We achieve this by properly segmenting the image and using watershed algorithm to handle overlapping circular segments when oranges are placed closely.

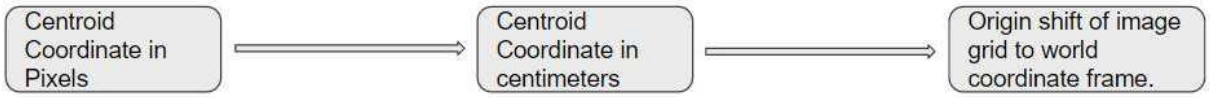
Segmentation is done by converting the raw image into HSV and using the correct Hue channels to create filters for green and orange color, indication different maturity level of the oranges.

Image after perspective correction is applied.

Marked centroids of ripe oranges.

Transformation of Centroid

The centroid values calculated till now are in the units of pixels with a coordinate frame different than of the world coordinate. Each centroid coordinate is converted into world coordinate frame using geometric transformations.



Inverse Kinematics

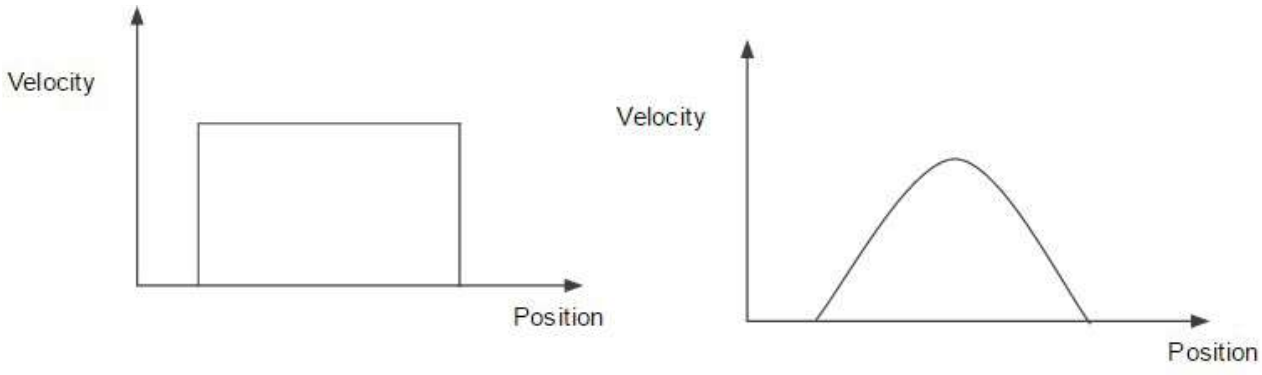
Once the destination coordinate is calculated, we use inverse kinematics to find the joint angles of the arm necessary to reach the orange. We do this by forming a triangle with the desired position of the arms as the sides, and calculate the angles of the triangle. We get the angles of the joints in Radians.

Conversion of angles to pulse width

The motor control of the servo is done by PWM and thus, the angle calculated in the previous step is converted into the PWM pulse value required to reach the particular angle. We have created the conversion functions by practical experimentation of the motor's profile.

Velocity and Acceleration Profile

The default property of servo motors is such that when we give it as input the destination pulse width, it tends to rotate at full speed. This leads to very abrupt motion of the whole robot arm. We have created functions to achieve velocity and acceleration control to achieve a smooth motion through software.



main.py | imageProcessing.py | centre.py | movements.py

```python
from imageProcessing import *
from movements import *

# Capture image from camera
image = captureImage()

# Calculate centroids from the image taken
centroids = findCentroids(image)

# transform all centroids into world coordinates
for maturity in centroids:
    coordinate_maturity = []
    for centroid in maturity:
        coordinate_maturity.append(transformToWorld(centroid))
    coordinates.append(coordinate_maturity)

RIPE_BOX = [20, 25]
RAW_BOX = [-20, 25]
destination = [RIPE_BOX, RAW_BOX]

# Go to coordinate, pick orange, drop in the correct box, again go to intial position
i = -1
for maturity in coordinates:
    i += 1
    for coordinate in maturity:
        goToCoordinate(coordinate)
        closeClaw()
        goToCoordinate(destination[i])
        openClaw()
        goToInitial()
```

main.py | imageProcessing.py | centre.py | movemen

```python
import io
import time
import picamera
import cv2
import numpy as np
from centre import find_orange

def captureImage():
    ## Capture image using picamera

    #Initialise bit stream
    stream = io.BytesIO()
    # initialise camera
    with picamera.PiCamera() as camera:
        # caputre image using video port to capture decrease processing time
        camera.capture(stream, format='jpeg', use_video_port=True)
        #convert bit stream into numpy array
        data = np.formstring(stream.getValue(), dtype=np.uint8)
        # create opencv image object from data
        img = cv2.imdecode(data, 1)
    return img

def correctPerspective(image):
    ## Correct the perspective of the image taken from the camera

    # pixel values of the corners of the grid image
    pts1 = np.float32([[661, 348], [1843, 385], [463, 1911], [2165, 1853]])
    # desired pixel values of those corners
    pts2 = np.float32([[0, 0], [1000, 0], [0, 1000], [1000, 1000]])

    # get transformation matrix for the non affine operation
    M = cv2.getPerspectiveTransform(pts1, pts2)
    # apply the transformation to correct the image
    dst = cv2.warpPerspective(image, M, (1000, 1000))

    return image

def findCentroids(image):
    image = correctPerspective(image)
    centroids = find_orange(image)
    return centroids

```

main.py | imageProcessing.py | centre.py | movements.py

```python
from __future__ import division
import cv2
from matplotlib import pyplot as plt
import numpy as np
from math import cos, sin
from skimage.feature import peak_local_max
from skimage.morphology import watershed
from scipy import ndimage
import timeit

def watershed_centroids(image):
    ## Find centroids of segments using watershed algorithm

    # Apply distance transform on the binary image
    D = ndimage.distance_transform_edt(image)
    # find local maximas
    localMax = peak_local_max(D, indices=False, min_distance=20, labels=image)
    # define markers for watershed
    markers = ndimage.label(localMax, structure=np.ones((3, 3)))[0]
    # create lables to find unique segments
    labels = watershed(-D, markers, mask=image)
    print("[INFO] {} unique segments found".format(len(np.unique(labels)) - 1))

    # initialise empty list to store centroids
    ripe_oranges = []
    # for each label/segment
    for label in np.unique(labels):
        if label ==0:
            continue

        # initialise zero matrix
        mask = np.zeros(image.shape, dtype="uint8")
        # copy each label into buffer variable
        mask[labels == label] = 255
        # find countour of the label to extract centroid
        cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPL
        # take the largest countour
        c = max(cnts, key=cv2.contourArea)
        # calculate moment of the contour
        M = cv2.moments(c)
        # calculate centroid
        cX = int(M["m10"] / M["m00"])
        cY = int(M["m01"] / M["m00"])
        # crete list of centroids
        ripe_oranges.append((cX, cY))

    return ripe_oranges
```

```
48
49  def find_orange(image):
50      # RGB is red green blue
51      # BGR is blue green red
52      # convert to the correct color scheme
53      image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
54
55      # clean image
56      image_blur = cv2.GaussianBlur(image, (7,7), 0)
57
58      # separates the color from the brightness of it. Luma from chroma.
59      image_blur_hsv = cv2.cvtColor(image_blur, cv2.COLOR_RGB2HSV)
60
61      # defining our filters
62      # filter by the color
63      # orange filter
64      min_red = np.array([13,100,80])
65      max_red = np.array([23,256,256])
66
67      mask1 = cv2.inRange(image_blur_hsv, min_red, max_red)
68      # green filter
69      min_green = np.array([40, 100, 80])
70      max_green = np.array([68, 256, 256])
71
72      mask2 = cv2.inRange(image_blur_hsv, min_green, max_green)
73
74      #combine our mas
75
76      masks = [mask1, mask2]
77      centroids = []
78
79      # for both the filters
80      for mask in masks:
81          # perform segmentation
82          kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (15,15))
83          # dilation
84          mask_closed = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)
85          # erosion
86          mask_clean = cv2.morphologyEx(mask_closed, cv2.MORPH_OPEN, kernel)
87          # pass thresholded image to find centroids
88          centers = watershed_centroids(mask_clean)
89          centroids.append(centers)
90          print centers
91
```

main.py | imageProcessing.py | centre.py | movements.py

```
1   from kinematics import *
2   from motor import *
3
4   def openClaw():
5       open_claw()
6
7   def closeClaw():
8       close_claw()
9
10  def goToCoordinate(coordinate):
11      ## Function to move motors
12
13      # Calculate arm joint angles from coordinate values
14      destination_angles = calculateDestinationAngles(coordinate)
15
16      # Calculate pulse width need to reach a particular joint angles
17      pulse_widths = calculatePulseWidth(destination_angles)
18
19      # move motors
20      moveMotors(pulse_widths)
21      print "Motors moved."
22
23  def goToInitial():
24      moveToInital()
25
26
64
65      # convert angle to the pulse width required to reach that angle.
66      # Calculated by trail and error from practical observations.
67      # Completely dependent on how motors are connected to arm.
68      pulseWidthBase = 8.55 * math.degrees(baseTheta) + 500
69      pulseWidthArm1 = 7.77 * math.degrees(arm1Theta) + 1510
70      pulseWidthArm2 = -(35/3) * math.degrees(arm2Theta) + 2500
71      pulseWidthGripper = 0
72
73      pulseWidths = [math.floor(pulseWidthBase), math.floor(pulseWidthArm1),
74      math.floor(pulseWidthArm2), math.floor(pulseWidthGripper)]
75
76      return pulseWidths
77
78
79
80
81
```

centre.py | imageProcessing.py | kinematics.py | main.py

```python
import numpy as np
import math
from main_coordinate import *

PATIYA = 11
ORANGE_RANGE = 3
CLAW_LENGTH = 12
HEIGHT_BASE = 8.5

ARM1 = 20.5
ARM2 = 22.5

def calculateAngle(a, b, c):
    # cosine rule to find angles of a triangle to find joint angles
    cosTheta = (c**2 - a**2 - b**2) / 2*a*b
    theta = math.acos(cosTheta)
    return theta

def transformToWorld(centroid):
    # size of the image in centimeters
    widthCM = 20
    lengthCM = 24
    # size of image in pixels
    IMG_LEN = 1000
    IMG_WIDTH = 800
    # distance of base of robot from grid of orange
    baseDistance = 10

    # convert pixel coordinates to centimeteres
    coordinateCM =[widthCM/IMG_WIDTH*centroid[0], lengthCM/IMG_LEN*centroid[1]]
    # convert centimeters coordinates into world coordinate frame
    worldCoordinate = [widthCM/2 - coordinateCM[0], baseDistance + coordinateCM[1]]

    return worldCoordinate

def calculateDestinationAngles(coordinate):
    ## Calculate joint angles from world coordinates. Inverse kinematics

    # z coordinate of end effector
    heightEndEffector = PATIYA + CLAW_LENGTH + ORANGE_RANGE
```

```
    # 3D coordinate of the end effector
    endEffectorCoordinate = np.array([coordinate[0], coordinate[1], heightEndEffector])
    # base coordinates
    baseCoordinate = np.array([0, 0, HEIGHT_BASE])

    # base of triangle between arm1, arm2 and gripper
    triangleBase = np.linalg.norm(endEffectorCoordinate - baseCoordinate)

    # calculate joint angles
    angleBase = math.atan2(coordinate[1], coordinate[0])
    angleArm2 = calculateAngle(ARM1, ARM2, triangleBase)
    angleArm1 = calculateAngle(,,ARM2)
    angleGripper = math.pi/2 + math.pi - (angleArm1 + angleArm2)

    return [angleBase, angleArm1, angleArm2, angleGripper]

def calculatePulseWidth(angles):
    # separate each joint's angle
    baseTheta = angles[0]
    arm1Theta = angles[1]
    arm2Theta = angles [2]
    gripperTheta = angles[3]

    # convert angle to the pulse width required to reach that angle.
    # Calculated by trail and error from practical observations.
    # Completely dependent on how motors are connected to arm.
    pulseWidthBase = 8.55 * math.degrees(baseTheta) + 500
    pulseWidthArm1 = 7.77 * math.degrees(arm1Theta) + 1510
    pulseWidthArm2 = -(35/3) * math.degrees(arm2Theta) + 2500
    pulseWidthGripper = 0

    pulseWidths = [math.floor(pulseWidthBase), math.floor(pulseWidthArm1),
    math.floor(pulseWidthArm2), math.floor(pulseWidthGripper)]

    return pulseWidths
```

centre.py | imageProcessing.py | kinematics.py

```python
#!/usr/bin/env python
import time
import pigpio
import math

PULSE_WIDTH = 1500
MIN_PULSE_WIDTH = 500
MAX_PULSE_WIDTH = 2500

# Width change variable
WIDTH_CHANGE = 70

# Sleep time
DELAY = 0.4

# Initial values of servo motors
BASE_PULSE_INIT = 1290
ARM1_PULSE_INIT = 2000 # calculated
ARM2_PULSE_INIT = 2000
GRIPPER_PULSE_INIT = 1500
CLAW_PULSE_INIT = 1200
CLAW_OPEN = 1800
CLAW_CLOSE = 1200

# GPIO connection pins
baseGPIO = 2
arm1GPIO = 3
arm2GPIO = 4
gripperGPIO = 17
clawGPIO = 27

# Initial pulses of motors
pulseBase = BASE_PULSE_INIT
pulseArm1 = ARM1_PULSE_INIT
pulseArm2 = ARM2_PULSE_INIT
pulseGripper = GRIPPER_PULSE_INIT
pulseClaw = CLAW_PULSE_INIT

# Claw Positions
clawOpenPosition = CLAW_OPEN
clawClosePosition = CLAW_CLOSE
```

```
clawOpenPosition = CLAW_OPEN
clawClosePosition = CLAW_CLOSE

# Motor class initialisation
base = motor(baseGPIO, BASE_PULSE_INIT)
arm1 = motor(arm1GPIO, ARM1_PULSE_INIT)
arm2 = motor(arm2GPIO, ARM2_PULSE_INIT)
gripper = motor(gripperGPIO, GRIPPER_PULSE_INIT)
claw = motor(clawGPIO, CLAW_PULSE_INIT)


class motor(object):
    # Class self parameters declaration
    def __init__(self, gpioPin, initPulse):
        print 'Initialised'
        self.motorPi = pigpio.pi()
        self.initialPos = initPulse
        self.pulse = initPulse
        self.gpioPin = gpioPin

    # Move motor to destination pulse
    def move(self, nextPulse):
        # Calcute change required to reach destination
        pwmWidthChange = nextPulse - self.pulse
        # Initialise 2 variables for later use
        initial = self.pulse
        position = initial

        # Calculate new pulse of the motor
        self.pulse += pwmWidthChange

        # Define velocity and accelaration values
        velocity = 5
        acceleration = 2

        # Limit destination pulse between 500 -2500
        if self.pulse < 500:
            self.pulse = MIN_PULSE_WIDTH
            print "Minimum position reached."
```

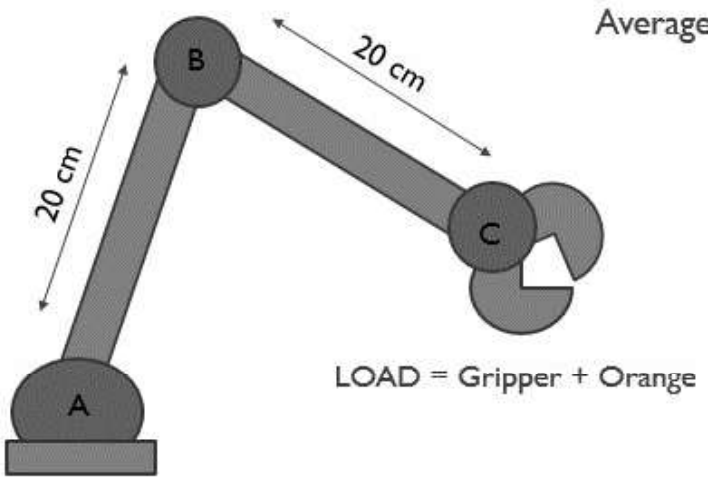
```
79
80          if self.pulse > 2500:
81              self.pulse = MAX_PULSE_WIDTH
82              print "Max position reached."
83
84          # Use this variable for loop
85          destination = self.pulse
86
87          # while position of motor is between initial and destination position
88          while min(initial, destination)<= position <= max(initial, destination):
89              # Mid point calculation of motion for changing accelaration
90              middle = initial + (destination - initial)/2
91              # Set accelaration direction, accelaration is positive halfway and negative in the other half
92              acc_direction = 1 if min(initial, middle) <= position <= max(initial, middle) else -0.6
93              vel_direction = 1 if destination >= initial else -1
94
95
96              self.motorPi.set_servo_pulsewidth(self.gpioPin, position)
97
98              # Update position and velocity according to velocity and accelaration respectively
99              position +=  vel_direction * velocity
100             velocity += acc_direction * acceleration
101
102             # Set maximum and minimum velocity limits. Calculated by practical trail and error
103             velocity = 35 if velocity > 35 else velocity
104             velocity = 5 if velocity < 5 else velocity
105
106             # Delay to be safe
107             time.sleep(0.005)
108
109         # This set_servo_pulsewidth is probably redundant, but I don't want to remove it,
110         # because I trust by earlier self.
111         self.motorPi.set_servo_pulsewidth(self.gpioPin, self.pulse)
112         print("Servo moved to {} micro pulses".format(self.pulse))
113
114
115     def stop(self):
116         # End class connection
117         self.motorPi.stop()
```

```
118
119 def moveToInitial():
120     # move to initial positions
121     base.move(base.initialPos)
122     arm1.move(arm1.initialPos)
123     arm2.move(arm2.initialPos)
124     gripper.move(gripper.initialPos)
125     claw.move(claw.initialPos)
126
127 def moveMotors(pulse_widths):
128
129     try:
130         # move motor only whe button is pressed. Safety step.
131         button = raw_input("Press button: ")
132         print "Base :"
133         print pulse_widths[0]
134         base.move(pulse_widths[0])
135
136         print "Arm1 :"
137         print pulse_widths[1]
138         arm1.move(pulse_widths[1])
139
140         print "Arm2 :"
141         print pulse_widths[2]
142         arm2.move(pulse_widths[2])
143
144         print "Gripper :"
145         print pulse_widths[3]
146         gripper.move(pulse_widths[3])
147
148     except KeyboardInterrupt:
149         # End the code by keyboard interrupt
150         moveToInitial()
151         base.stop()
152         arm1.stop()
153         arm2.stop()
154         gripper.stop()
155         print "Program exited"
156
157 def open_claw():
158     claw.move(CLAW_OPEN)
159     print "Claw is open."
160
161 def close_claw():
162     claw.move(CLAW_CLOSE)
163     print "Claw is closed."
164
165
```

3.2 Hardware

The model comprises of a serial 4 DOF arm. The lengths of the arm is meticulously calculated through MATLAB analysis of serial robot giving the range as the input. The load analysis is used to calculate the required torque at each joint after which the motor type is decided. The design is done on the basis of proposed calculations and the system is tested for required load conditions.

3.2.1 Load Analysis

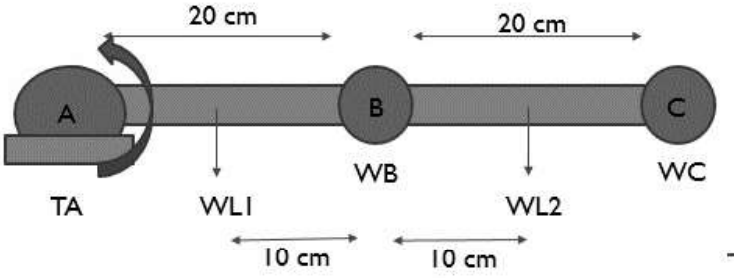


Average weight at C

= Weight of gripper + Load
= Weight of (M4 + M5 + Material) + Load
= 10 + 55 + 100 + 130
= 295 gm

WC ~ 300 gm

Fig. 18 Load Analysis_1



TA = 10 x WL1 + 20 x WB + 30 x WL2 + 40 x WC
= 10 x .03 + 20 x 0.055 + 30 x 0.03 + 40 x 0.300
= 0.3 + 1.3 + 0.9 + 1.2
= 14.3 kg cm

Fig. 19 Load Analysis_2

TB = 10 x WL1 + 20 x WC
= 10 x .03 + 20 x 0.300
= 0.3 + 6
= 6.3 kg cm

3.2.2 MATLAB Analysis

CODE for Serial Kinematics in MATLAB

```
% --- Executes on button press in draw_plot.
function draw_plot_Callback(hObject, eventdata, handles)
%--------------------Basic mechanism bot movement------------------------%

px = get(handles.x_coord,'Value');  %X_coordinate
py = get(handles.y_coord,'Value');  %Y_coordinate
l1 = get(handles.link_1,'Value');   %Link 1 length
l2 = get(handles.link_2,'Value');   %Link 2 length

set(handles.x_disp,'String',px);
set(handles.y_disp,'String',py);
set(handles.link1_disp,'String',l1);
set(handles.link2_disp,'String',l2);

gp =  5;  %Length of the gripper arm
bs =  5;  %Base length

%Calculation

r = sqrt(px^2+py^2);
l = l2+gp;

thr  = acos((l1^2+l^2-r^2)/(2*l1*l));
thl1 = acos((l1^2+r^2-l^2)/(2*l1*r));

th1 = pi/2-thl1;
th2 = pi-thr ;
th3 = atan(abs(py/px));
   if px<0
        if py<0
            th3 = th3-pi;
        else
            th3 = pi-th3;
        end
    else
        if py<0
            th3 = -th3;
        end
   end

set(handles.act3,'String',th3*180/pi);
th3 = th3-pi/2;
a1 = thl1-pi/2;
a2 = -(th1+th2);
lt1 = [0 bs];
lt2 = [l1*cos(thl1) l1*sin(thl1)+bs];
```

Resultant GUI

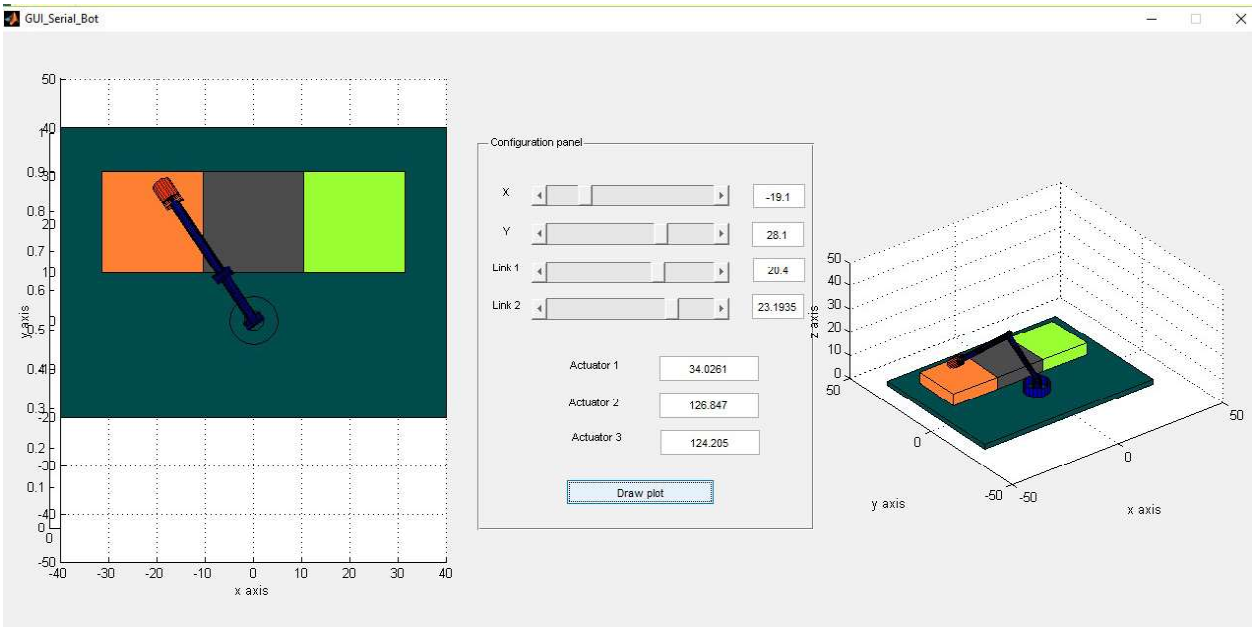


Fig. 20 MATLAB GUI based testing

3.2.3 Schematic Model

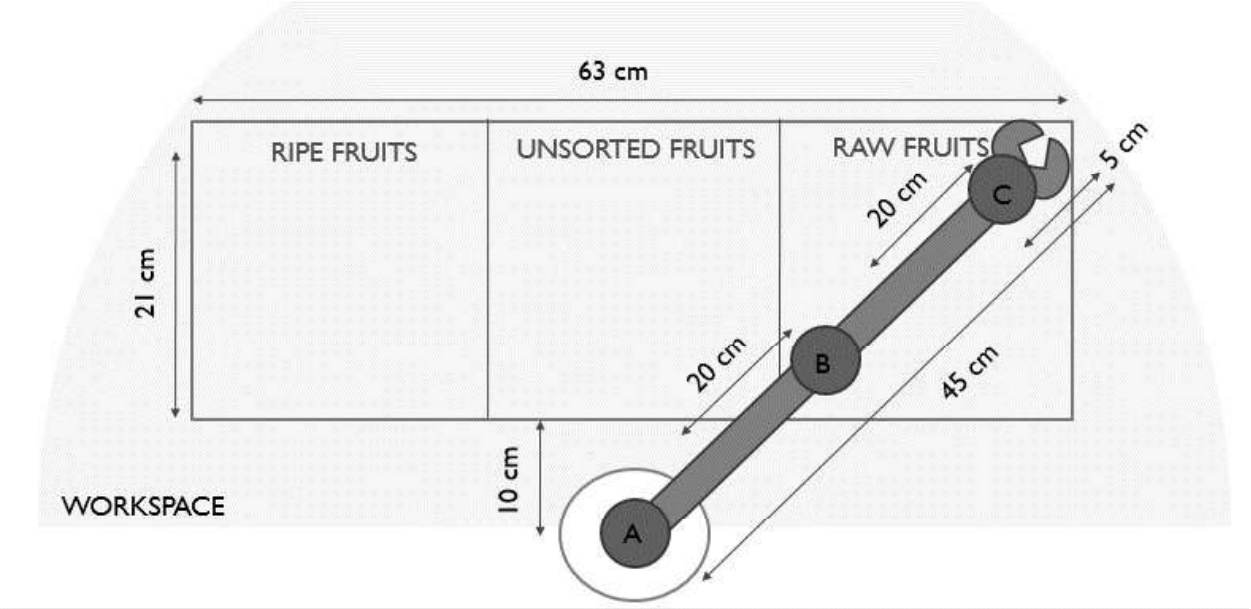


Fig. 21 Schmatic diagram of the serial arm assembly

3.3 Components Description

3.3.1. RASPBERRY PI PROCESSOR

Model used - Raspberry Pi 2

The Raspberry Pi 2 Model B is the second generation Raspberry Pi. The Raspberry Pi is open hardware, with the exception of the primary chip on the Raspberry Pi, the Broadcomm SoC (System on a Chip), which runs many of the main components of the board–CPU, graphics, memory, the USB controller, etc.

Features/ Specifications:

- SoC: Broadcom BCM2836 (CPU, GPU, DSP, SDRAM)
- CPU: 900 MHz quad-core ARM Cortex A7 (ARMv7 instruction set)
- GPU: Broadcom VideoCore IV @ 250 MHz
- More GPU info: OpenGL ES 2.0 (24 GFLOPS); 1080p30 MPEG-2 and VC-1 decoder (with license); 1080p30 h.264/MPEG-4 AVC high-profile decoder and encoder
- Memory: 1 GB (shared with GPU)
- USB ports: 4
- Video input: 15-pin MIPI camera interface (CSI) connector
- Video outputs: HDMI, composite video (PAL and NTSC) via 3.5 mm jack
- Audio input: I²S
- Audio outputs: Analog via 3.5 mm jack; digital via HDMI and I²S
- Storage: MicroSD
- Network: 10/100Mbps Ethernet
- Peripherals: 17 GPIO plus specific functions, and HAT ID bus
- Power rating: 800 mA (4.0 W)
- Power source: 5 V via Micro USB or GPIO header
- Size: 85.60mm × 56.5mm
- Weight: 45g (1.6 oz.)

GPIO Pins

- Of the 40 pins, 26 are GPIO pins and the others are power or ground pins
- General Purpose Input/ Output (GPIO)
- Pins can be configured to be input/output

Fig. 22 Raspberry PI 2 Board

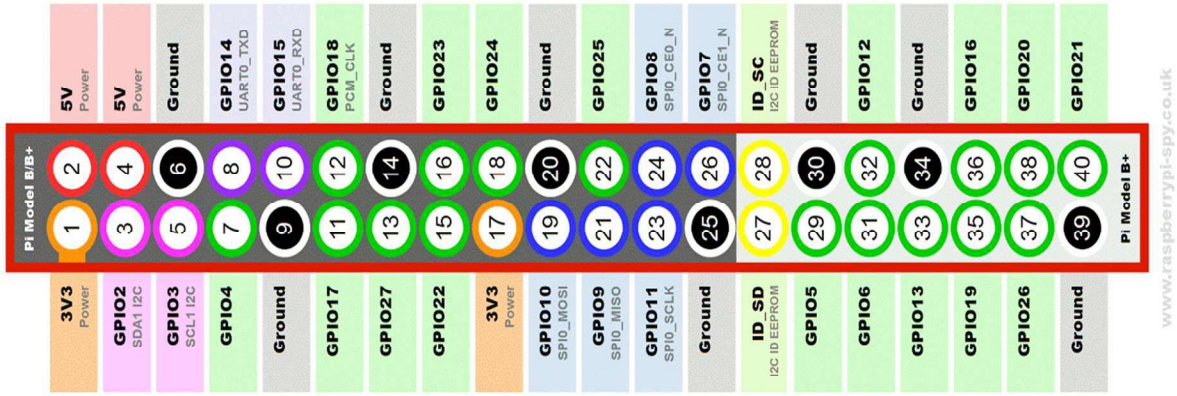


Fig. 23 Raspberry PI 2 GPIO pins

3.3.2. RASPBERRY PI CAMERA

The module attaches to Raspberry Pi, by way of a 15 Pin Ribbon Cable, to the dedicated 15-pin MIPI Camera Serial Interface (CSI), which was designed especially for interfacing to cameras. The CSI bus is capable of extremely high data rates.

High definition camera module compatible with the Raspberry Pi model A and model B. Provides high sensitivity, low crosstalk and low noise image capture in an ultra-small and lightweight design. The camera module connects to the Raspberry Pi board via the CSI connector designed specifically for interfacing to cameras. The CSI bus is capable of extremely high data rates, and it exclusively carries pixel data to the BCM2835 processor.

Fig. 24 Raspberry PI camera

Module used - Camera v2

	Camera Module v1	Camera Module v2
Net price	$25	$25
Size	Around 25 × 24 × 9 mm	
Weight	3g	3g
Still resolution	5 Megapixels	8 Megapixels
Video modes	1080p30, 720p60 and 640 × 480p60/90	1080p30, 720p60 and 640 × 480p60/90
Linux integration	V4L2 driver available	V4L2 driver available
C programming API	OpenMAX IL and others available	OpenMAX IL and others available
Sensor	OmniVision OV5647	Sony IMX219
Sensor resolution	2592 × 1944 pixels	3280 × 2464 pixels
Sensor image area	3.76 × 2.74 mm	3.68 x 2.76 mm (4.6 mm diagonal)
Pixel size	1.4 µm × 1.4 µm	1.12 µm x 1.12 µm
Optical size	1/4"	1/4"
Full-frame SLR lens equivalent	35 mm	
S/N ratio	36 dB	
Dynamic range	67 dB @ 8x gain	
Sensitivity	680 mV/lux-sec	
Dark current	16 mV/sec @ 60 C	
Well capacity	4.3 Ke-	
Fixed focus	1 m to infinity	
Focal length	3.60 mm +/- 0.01	3.04 mm
Horizontal field of view	53.50 +/- 0.13 degrees	62.2 degrees
Vertical field of view	41.41 +/- 0.11 degrees	48.8 degrees
Focal ratio (F-Stop)	2.9	

Table 2. Raspberry PI Camera Specifications

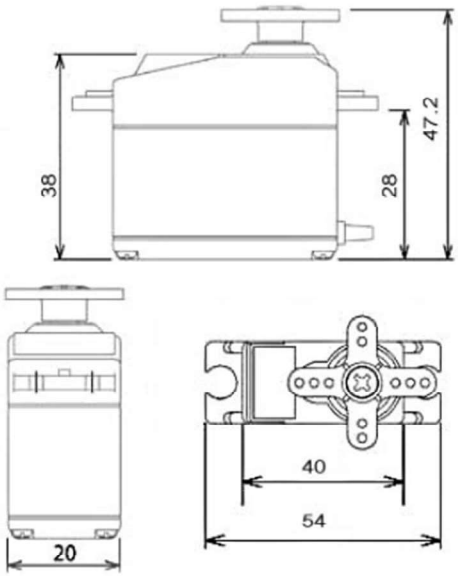
3.3.3. SERVO MOTORS

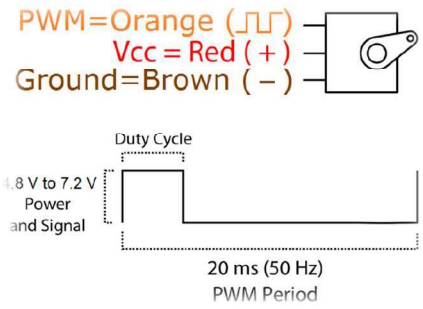
The following motors are used in the final assembly for the actuation of the robotics arm.

1. **MG995 High Speed Metal Gear Dual Ball Bearing Servo**
   The unit comes complete with 30cm wire and 3 pin 'S' type female header connector that fits most receivers, including Futaba, JR, GWS, Cirrus, Blue Bird, Blue Arrow, Corona, Berg, Spektrum and Hitec.This high-speed standard servo can rotate approximately 120 degrees (60 in each direction).
   You can use any servo code, hardware or library to control these servos, so it's great for beginners who want to make stuff move without building a motor controller with feedback & gear box, especially since it will fit in small places. The MG995 Metal Gear Servo also comes with a selection of arms and hardware to get you set up nice and fast.

   Number of motors used - 3
   Specifications:

   - Weight: 55 g
   - Dimension: 40.7 x 19.7 x 42.9 mm approx.
   - Stall torque: 8.5 kgf·cm (4.8 V), 10 kgf·cm (6 V)
   - Operating speed: 0.2 s/60º (4.8 V), 0.16 s/60º (6 V)
   - Operating voltage: 4.8 V a 7.2 V
   - Dead band width: 5 µs
   - Stable and shock proof double ball bearing design
   - Temperature range: 0 ºC – 55 ºC



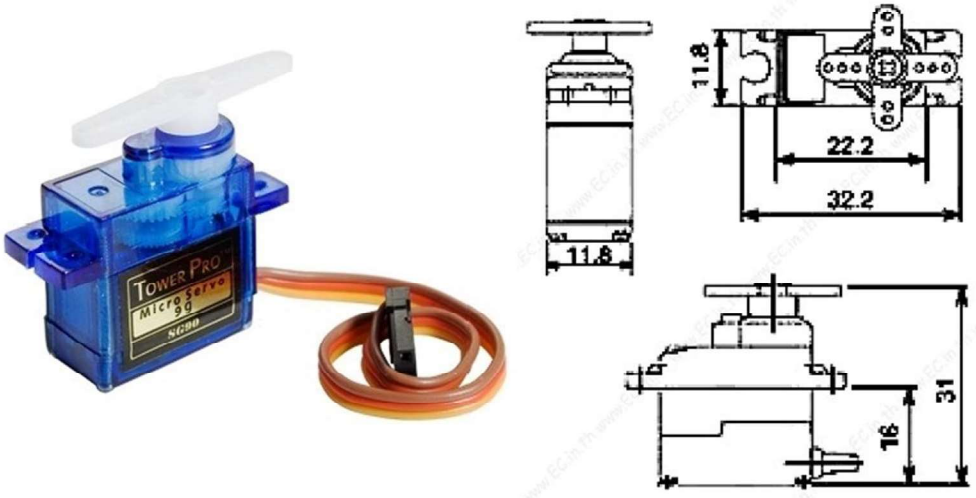


2. SG90 9 g Micro Servo

Tiny and lightweight with high output power. Servo can rotate approximately 180 degrees (90 in each direction), and works just like the standard kinds but smaller. You can use any servo code, hardware or library to control these servos. Good for beginners who want to make stuff move without building a motor controller with feedback & gear box, especially since it will fit in small places. It comes with a 3 horns (arms) and hardware.

Number of motors used – 1
Specifications:

- Weight: 9 g
- Dimension: 22.2 x 11.8 x 31 mm approx.
- Stall torque: 1.8 kgf·cm
- Operating speed: 0.1 s/60 degree
- Operating voltage: 4.8 V (~5V)
- Dead band width: 10 µs
- Temperature range: 0 ºC – 55 ºC



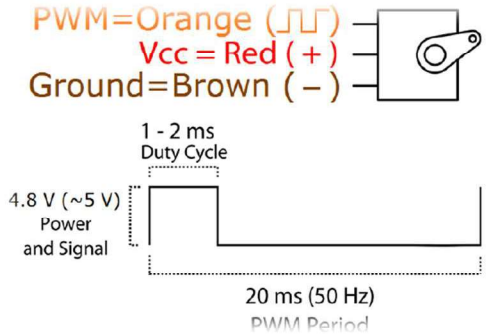


Fig. 26 Tower Pro S90 9g Servo Motor
Schematic and PWM pulse

3. MG958 High Speed Metal Gear Dual Ball Bearing Servo

Number of motors used - 1
Specifications:

- Weight: 65g
- Dimensions: 40.2 x 20.1 x 36.8mm
- Operating Voltage: 4.8v-6.6v
- Stall Torque (4.8v): 18.0kg.cm / 250oz-in
- Stall Torque (6.6v): 20.0kg.cm / 277.7oz-in
- Servo case: Alloy case in the middle
- Gear Type: Metal gear
- Bearing: Double ball bearing
- Temperature range: 0- 55deg
- Operating Temperature Range: -20 ~ +60 Degree C
- Operating Speed (4.8v): 0.18sec/60 degree
- Operating Speed (6.0v): 0.15sec/60 degree
- Servo wire length: 32cm
- Servo Plug: JR (Fits JR and Futaba)
- Servo arms & screws included and fit with Futaba servo arm
- CE & RoHS approved

Fig. 27 Tower Pro MG 958 Servo
Motor and Assembly

3.3.4. POWER SUPPLY

There are two power supplies that are used, as mentioned-

- Round pin power supply - 5V, 2A for powering servo motors
- Micro USB Power Supply - 5V, 1.5A for powering the raspberry pi board

3.3.5 MATERIALS USED

- Wooden Base for the whole assembly
- Wooden Base Robot's bottom
- ABS for 3D printing
- 4mm Acrylic Sheet for laser cutting
- Nuts, bolts and clamps
- Screws and Springs
- Perforated sheet for Circuit Making
- Round Adapter Pins
- Male-Male, Male-Female, Female-Female wire connecters
- Soldering Iron, Flux and Wire

3.4. Assembly of robotic arm

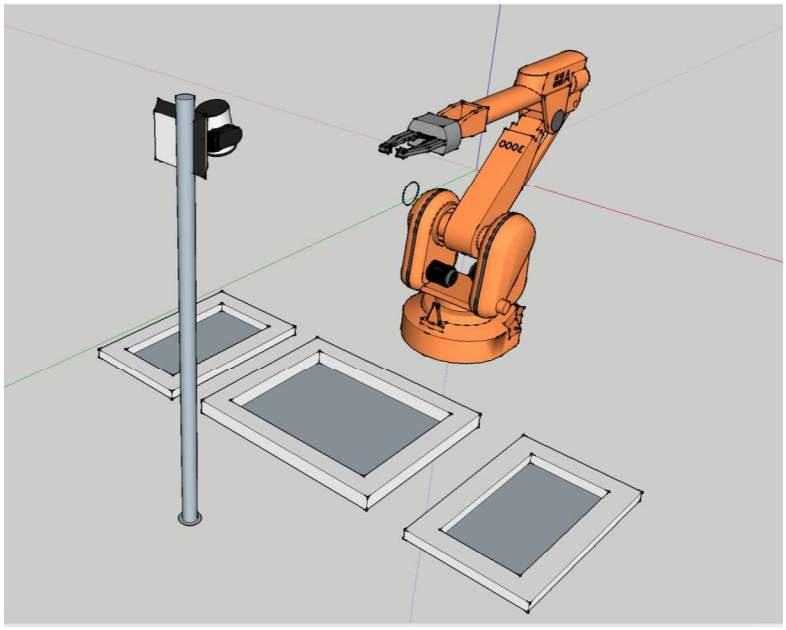
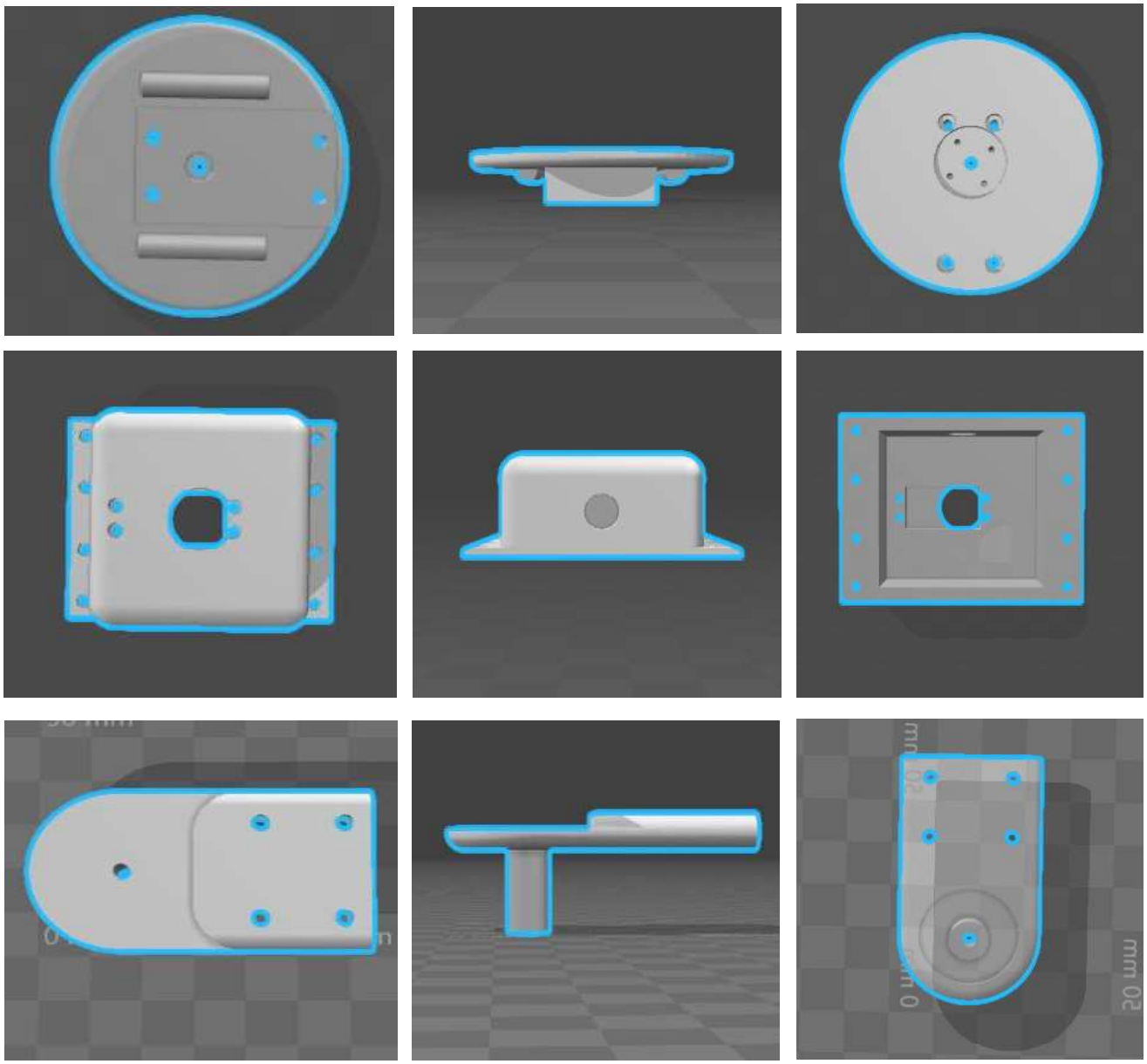


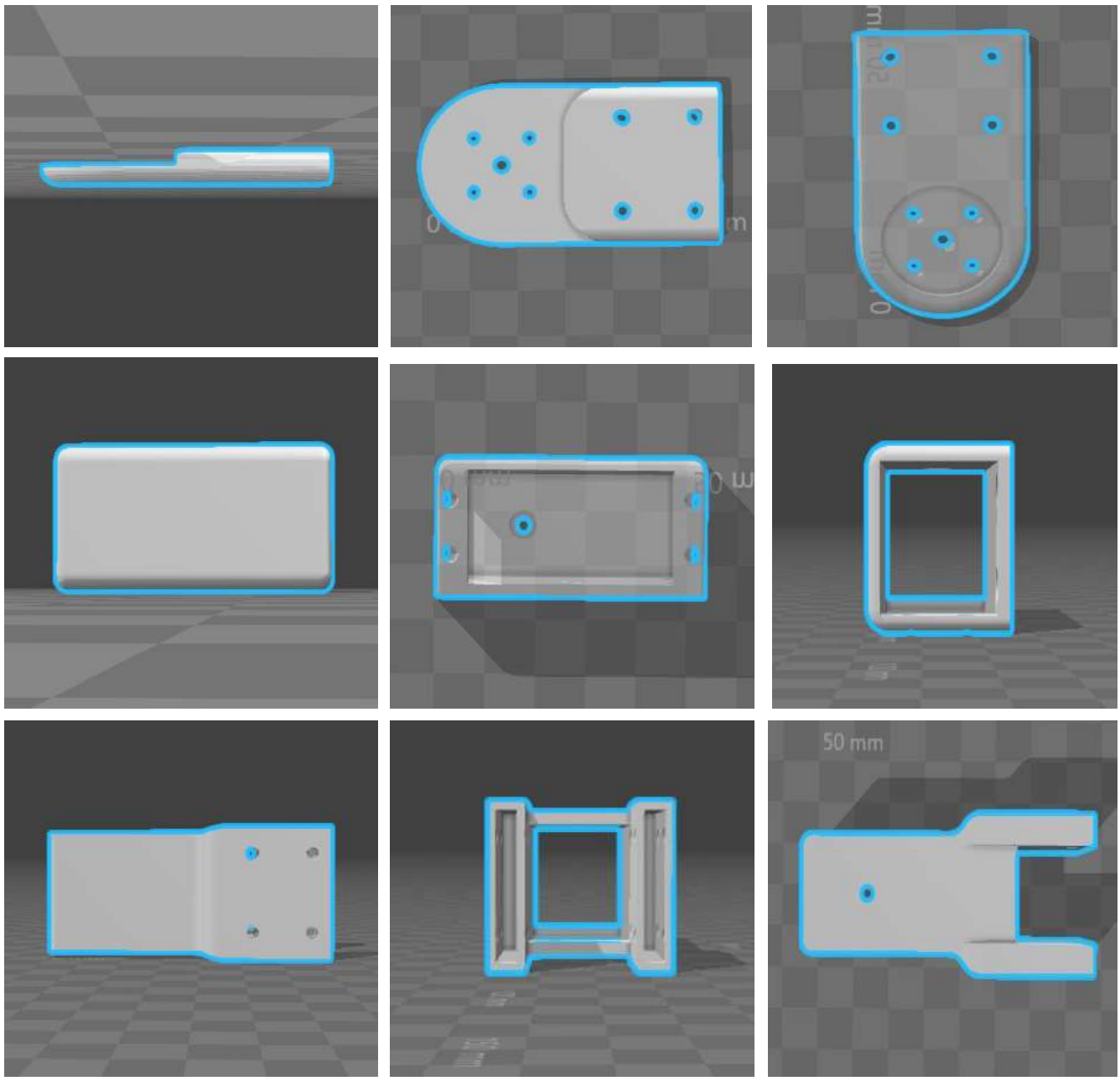
Fig. 28 Fully Assembled System Schematic

3.4.1 3D printed parts design



(a) (b) (c)

Fig. 29 3D printed parts (a) Top View (b) Side View (c) Bottom View



(a) (b) (c)

Fig. 30 3D printed parts (a) Top View (b) Side View (c) Bottom View

3.4.2 Laser cut parts

Four links are cut from 4 mm Acrylic sheet of dimensions 20 X 3

3.4.3 Final Assembly in SolidWorks

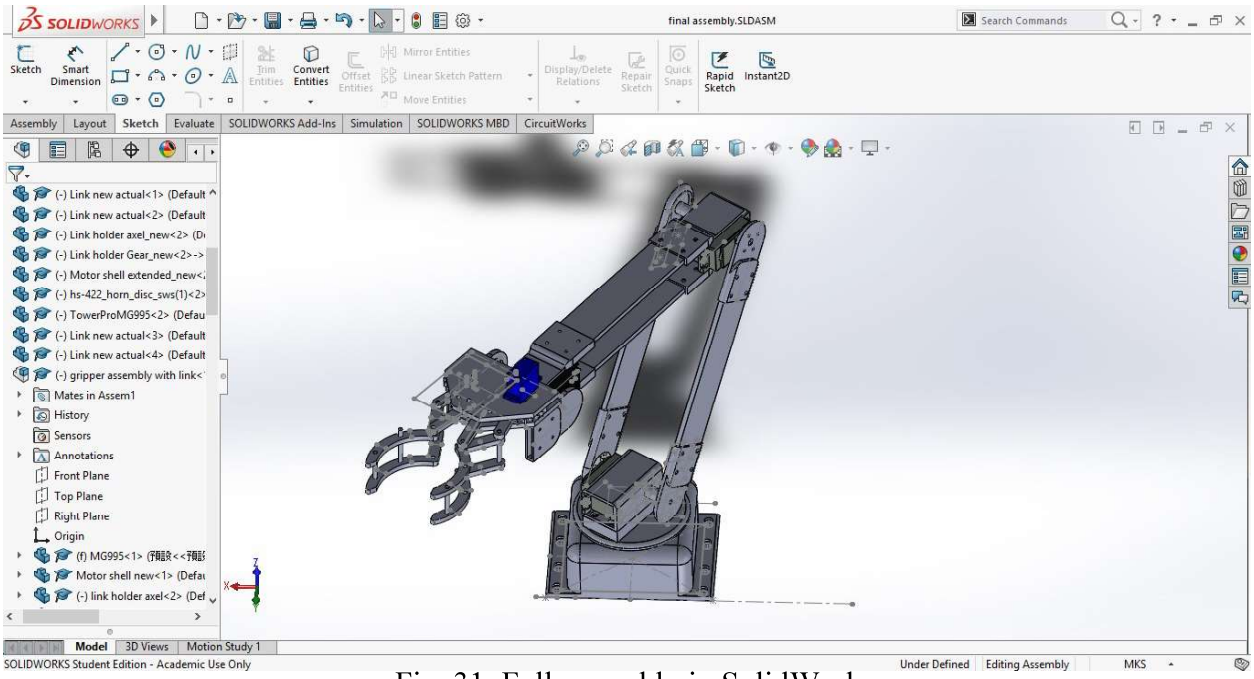


Fig. 31 Full assembly in SolidWorks

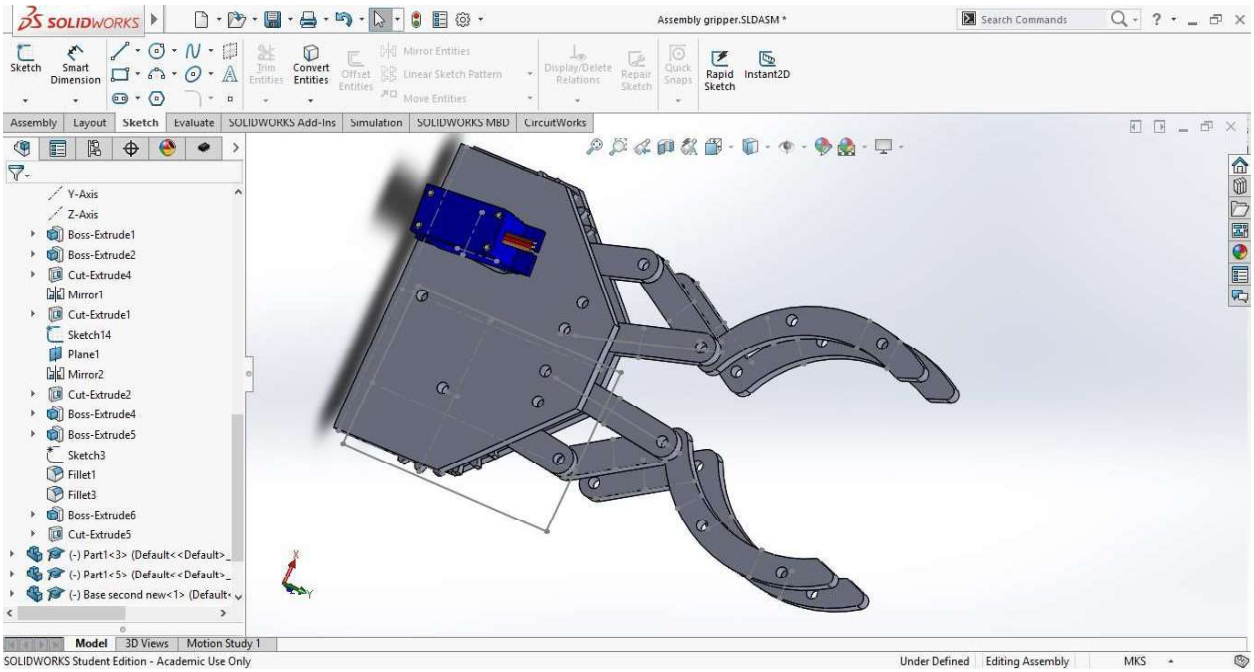


Fig. 32 Gripper Assembly in SolidWorks

3.4.4 Final Prototype

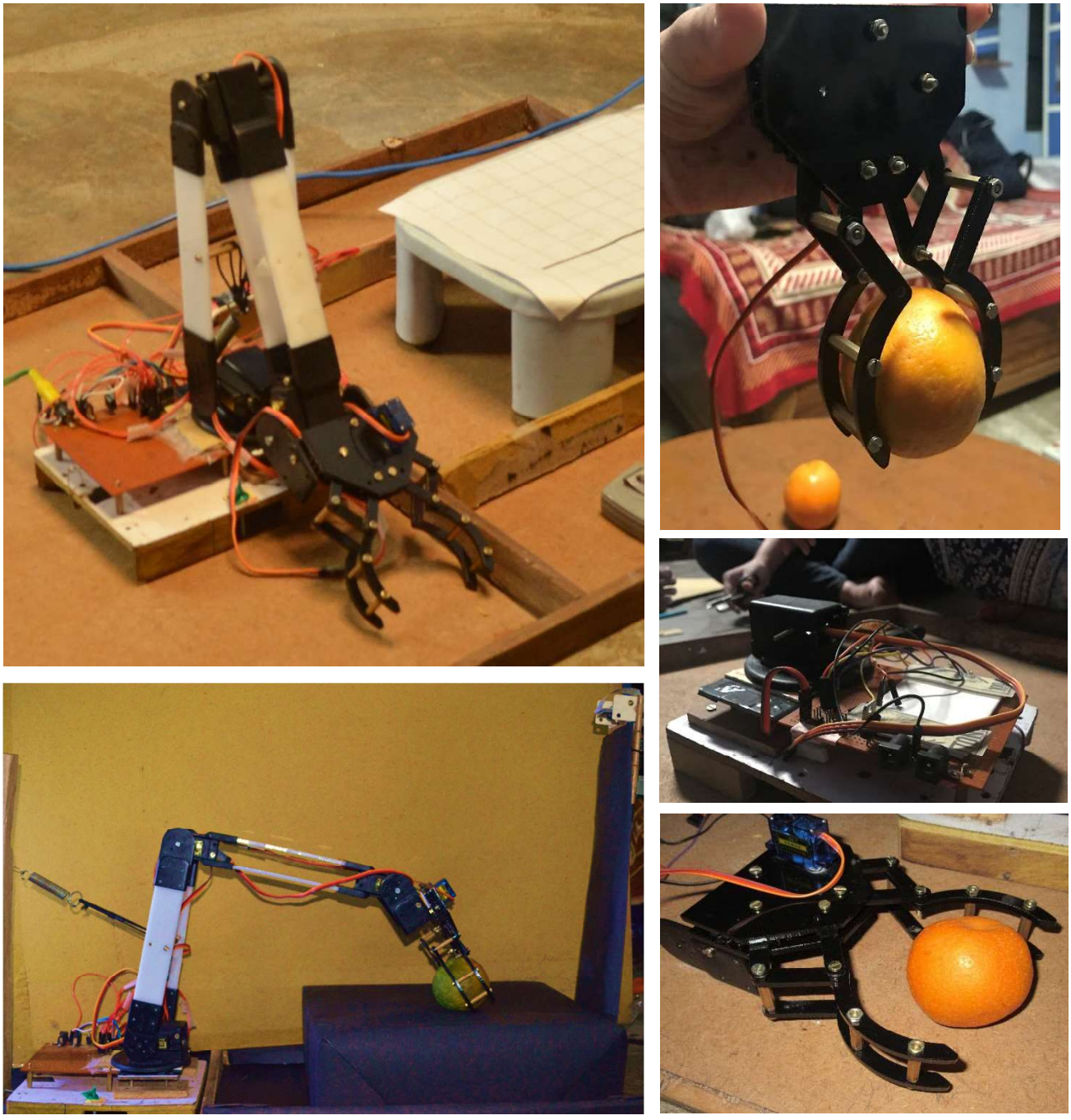


Fig. 33 4 DOF object sorting arm - Assembly and Setup

3.4.5 Circuit Diagram

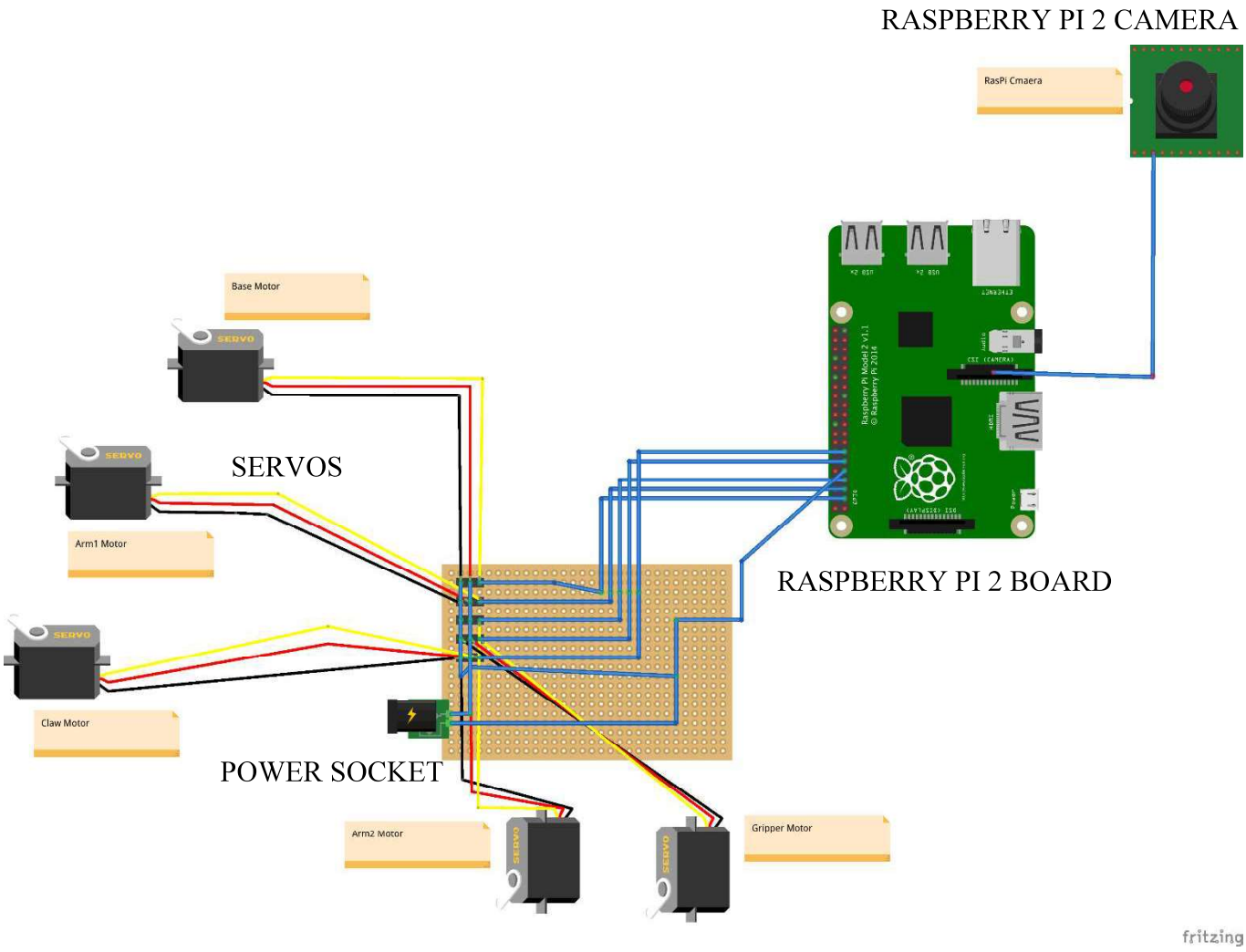


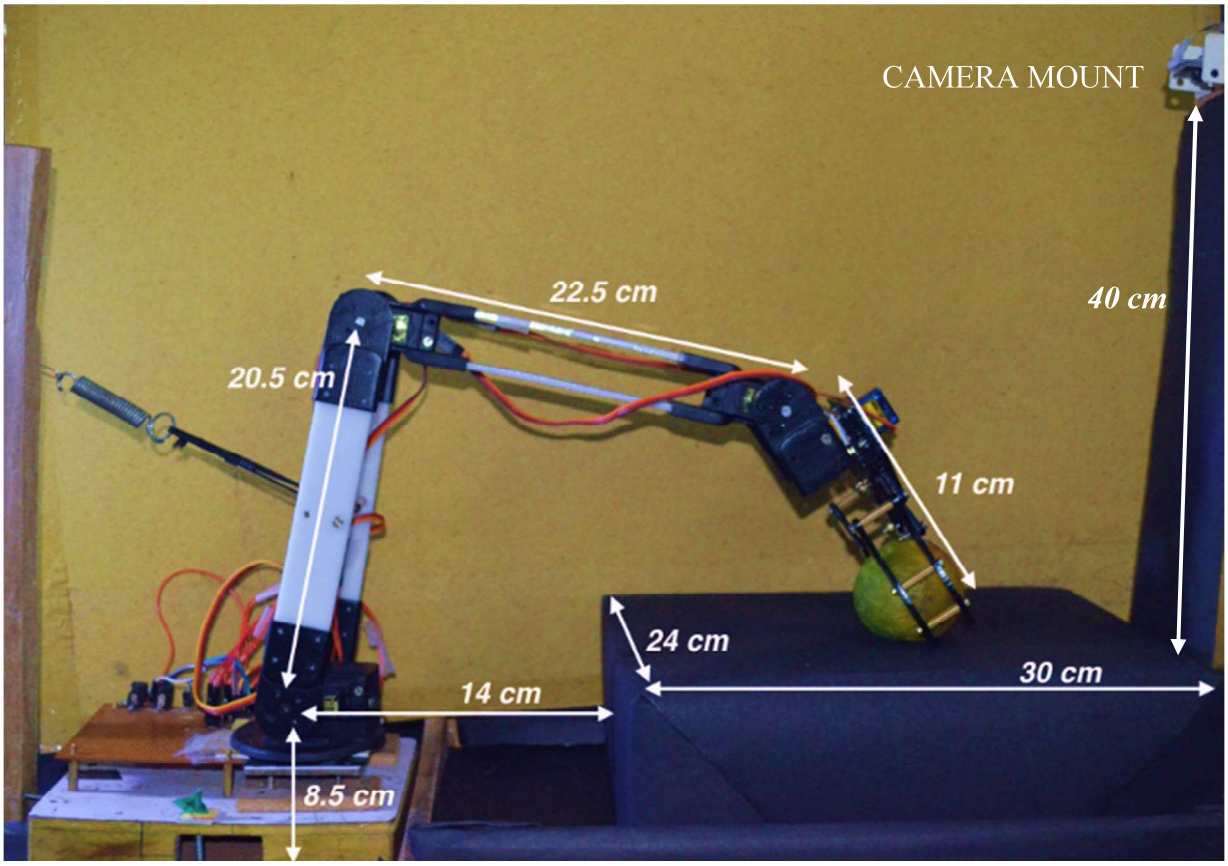
Fig. 34 Circuit assembly of all thecomponents

Chapter 4
RESULTS

On the basis of the theoretical calculations, we have taken a number of observations of the detection and sorting technique using varied test cases. The results of the detection part has been presented in Section 4.1.3 using number of test image dataset. The limitations have been discussed in Section 4.1.4. and Sorting mechanism and its performance is mentioned in Section 4.1.5.

4.1 Implementation Outcomes

4.1.1. COMPLETE ASSEMBLY WITH MEASUREMENTS



4.1.2. IMAGE DETECTION DATABASE DETAILS

Detection part is completed in four steps. A total of 30 images were tested out of which 15 are displayed with outcomes. 2 erroneous results are also displayed. A frame consists of maximum 9 oranges. Efforts are made to take as much variations as possible. The data set is given accordingly in Section 4.1.1.3.

- Applying Perspective shift (Image 1)
- Segmentation based on maturity for Ripe and Raw separately (Image 2 and 3 respectively)
- Smoothing using dilation operation
- Centroid detection through watershed algorithm (Image 4)

(Images are places in a vertical queue Image 1-top, Image 4- bottom)

1. PERSPECTIVE SHIFT (TEST IMAGE)

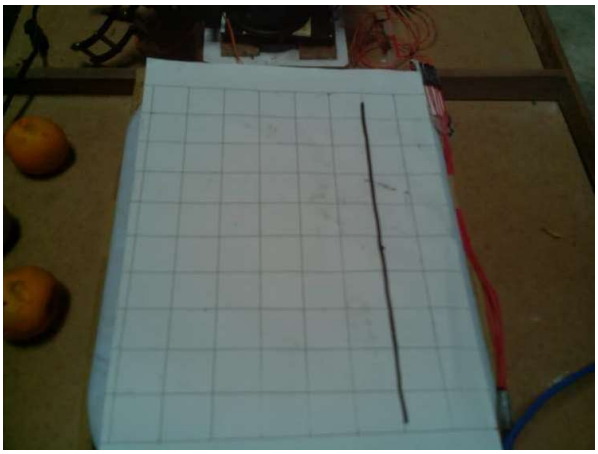


Fig. 35 Normal Image taken by Raspi Camera

Fig. 36 Perspective Shift

2. CENTROID DETECTION THROUGH WATERSHED ALGORITHM (TEST IMAGE)

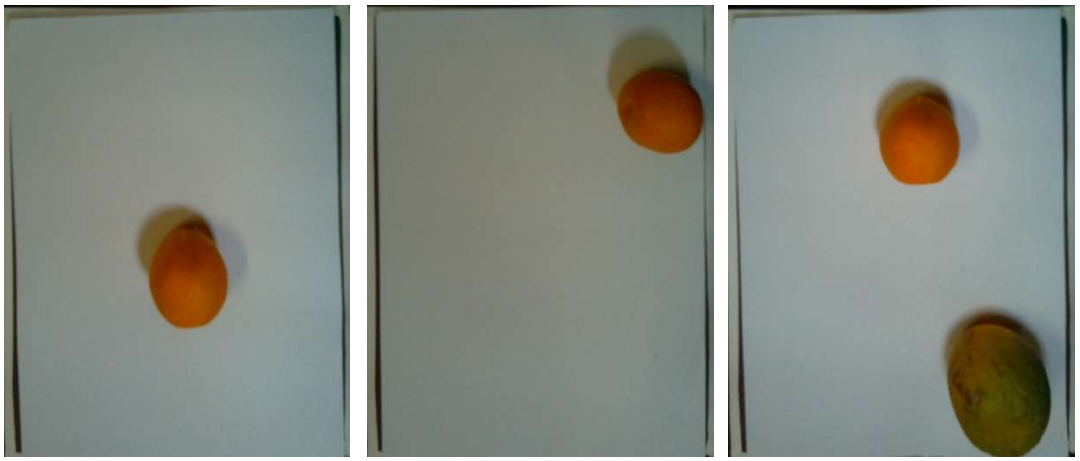
Fig. 37 Normal Image taken by Raspi Camera

Fig. 38 Image after centroid detection

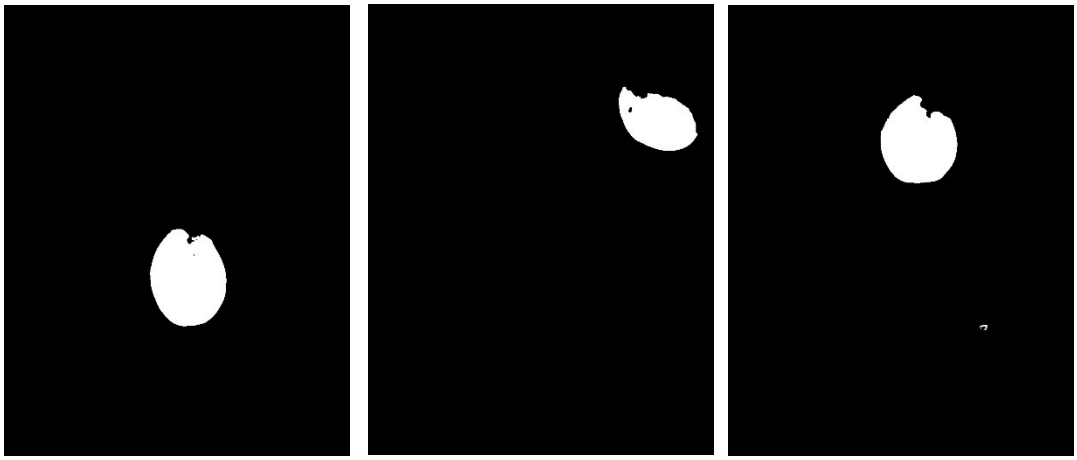
4.1.3. TEST IMAGE DATASET

Test Img. 1-15 (From left to right)

Test Img. 1-3
(From left to right)
Perspective Shift



Segmentation using
HSV values
RIPE ORANGES



Segmentation using
HSV values
RAW ORANGES

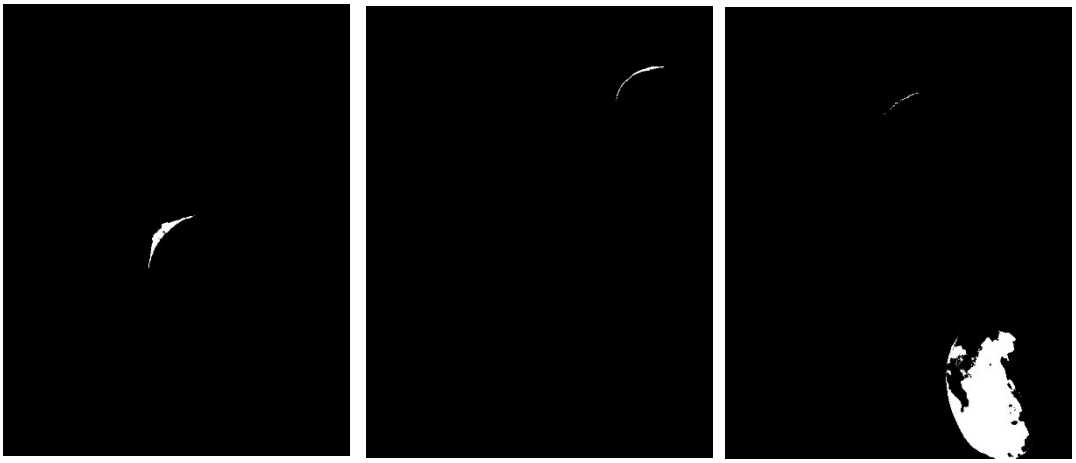
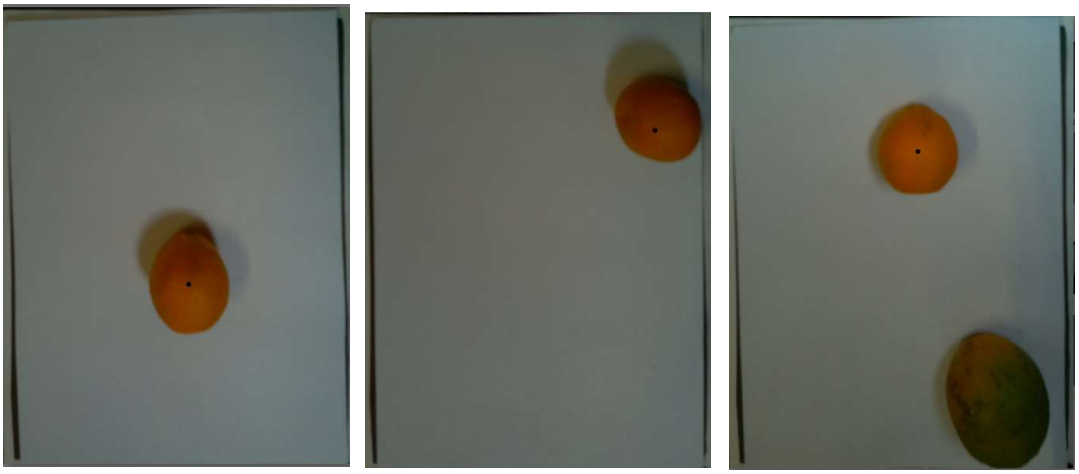
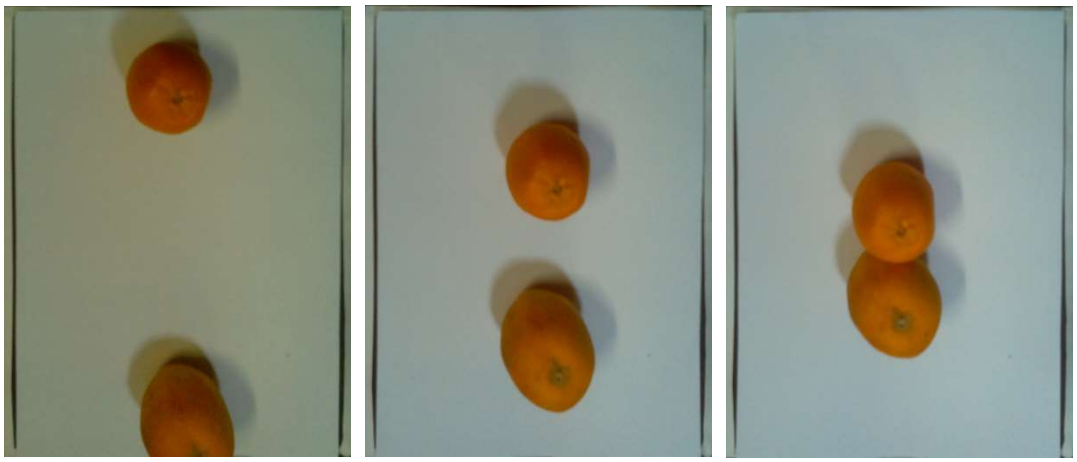


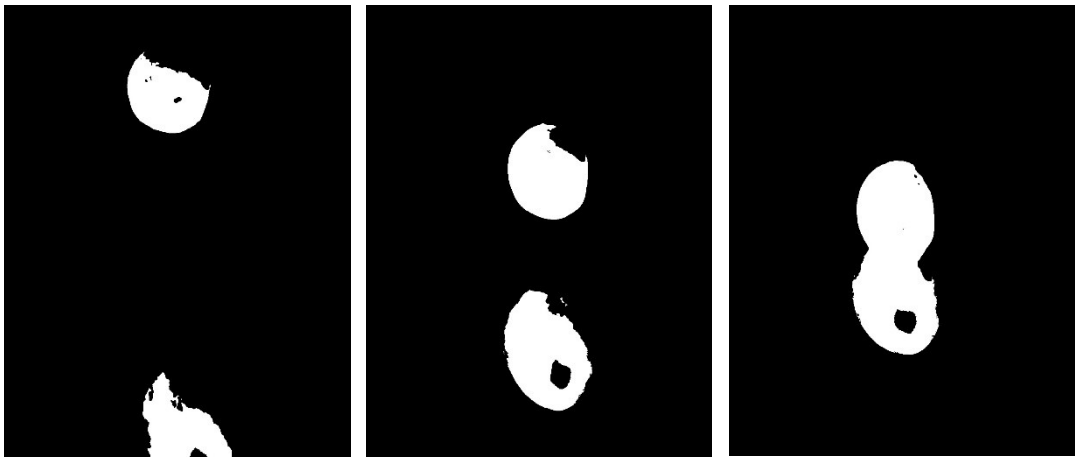
Image after
Centroid Detection



Test Img. 4-6
(From left to right)
Perspective Shift



Segmentation using
HSV values
RIPE ORANGES



Segmentation using
HSV values
RAW ORANGES

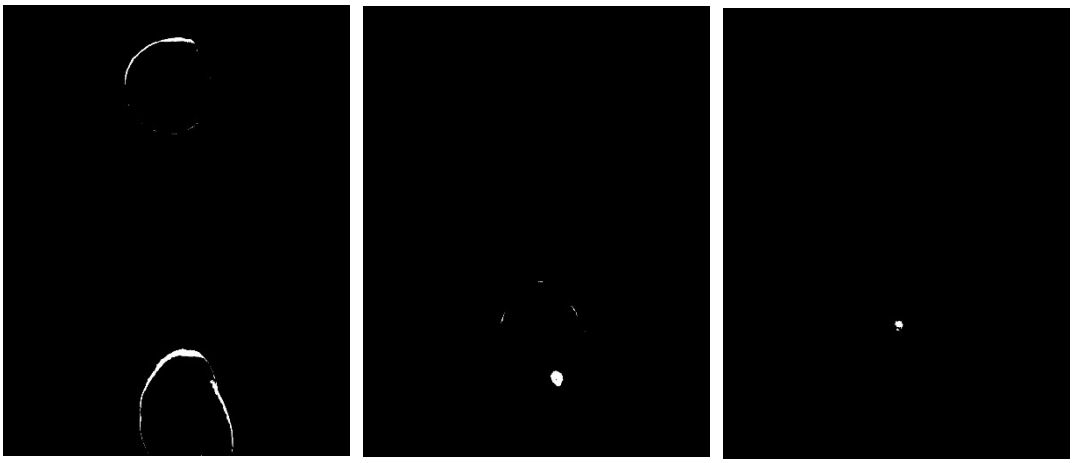


Image after
Centroid Detection

Test Img. 7-9
(From left to right)
Perspective Shift

Segmentation using
HSV values
RIPE ORANGES

Segmentation using
HSV values
RAW ORANGES

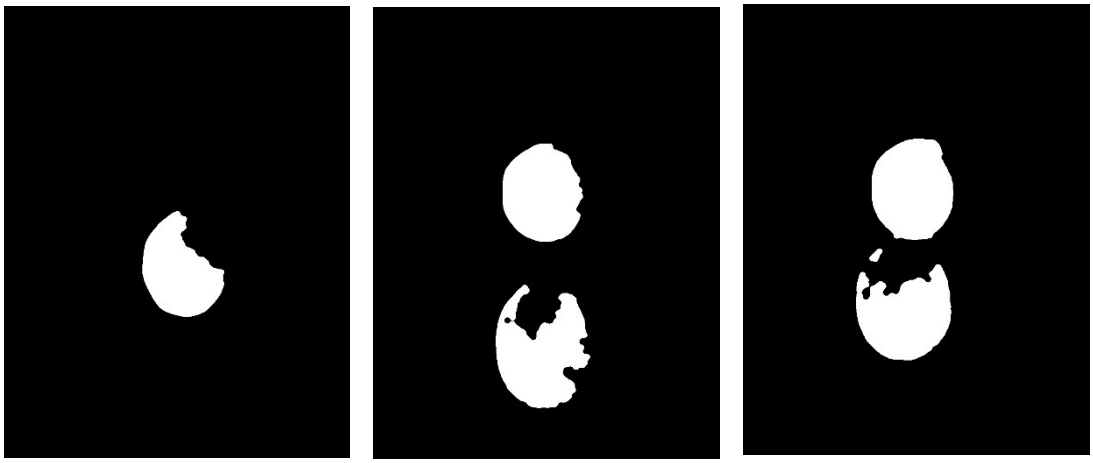
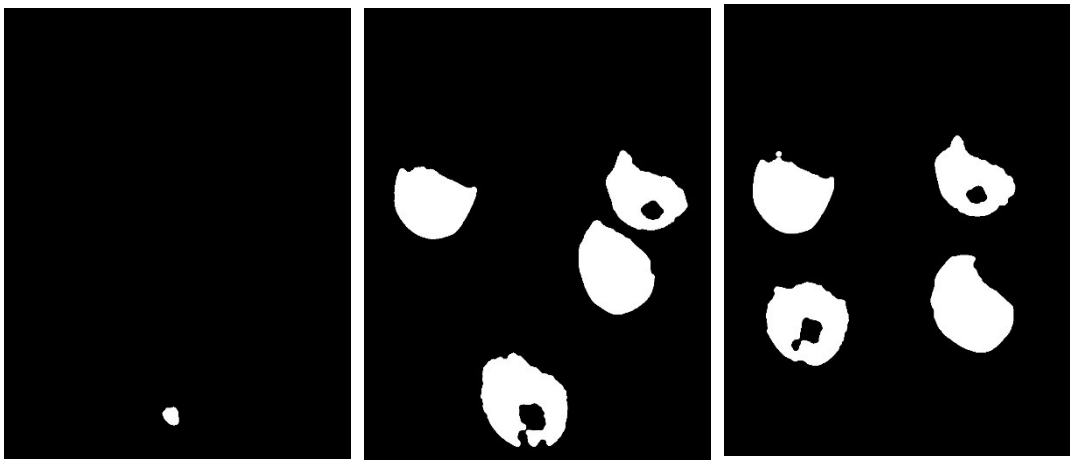


Image after
Centroid Detection

Test Img. 10-12
(From left to right)
Perspective Shift

Segmentation using
HSV values
RIPE ORANGES



Segmentation using
HSV values
RAW ORANGES

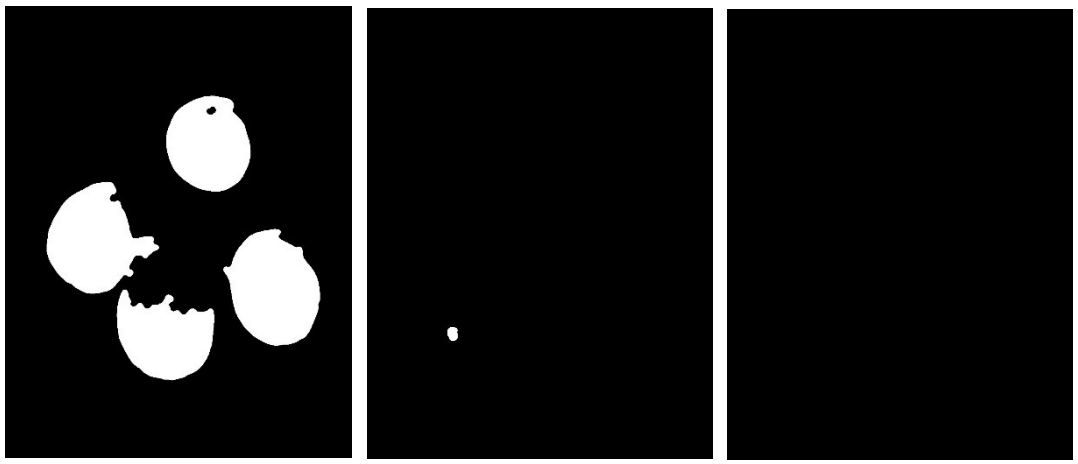
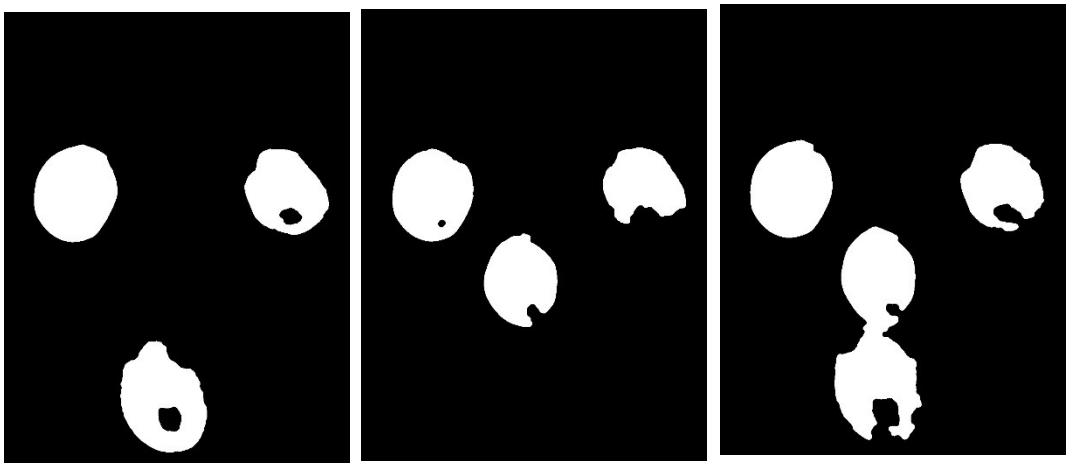


Image after
Centroid Detection

Test Img. 13-15
(From left to right)
Perspective Shift

Segmentation using
HSV values
RIPE ORANGES



Segmentation using
HSV values
RAW ORANGES

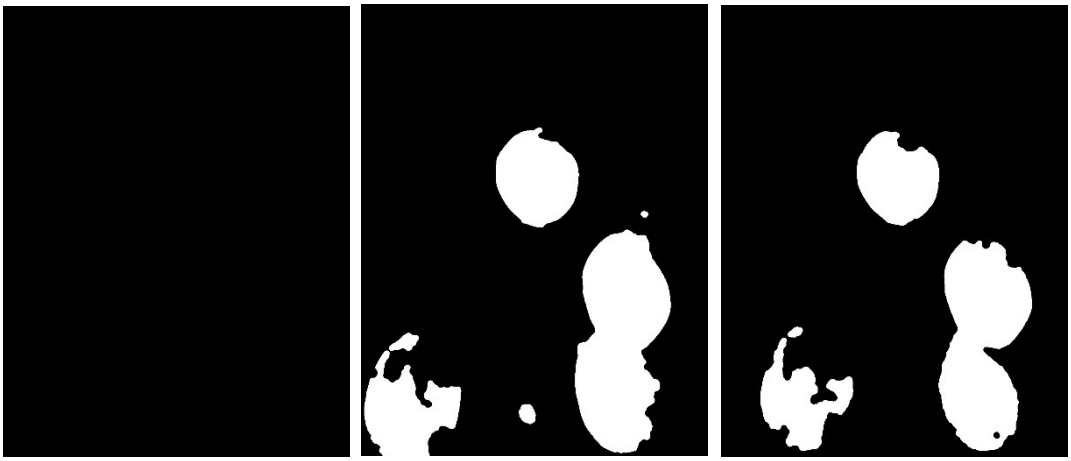


Image after
Centroid Detection

4.1.4. LIMITATIONS OBSERVED IN TEST IMAGES DATASET

Erroneous Test Image 1.

(a) Perspetive Shift
(b) Segmented Image
(c) Image after Centroid Detection

Centroid detection fails in this case because of the orange being to close to the boundary. The perspective shift applied removed the orange's area in the picture, which makes its segment lesser in area than the threshold kept for detection.

Erroneous Test Image 2.

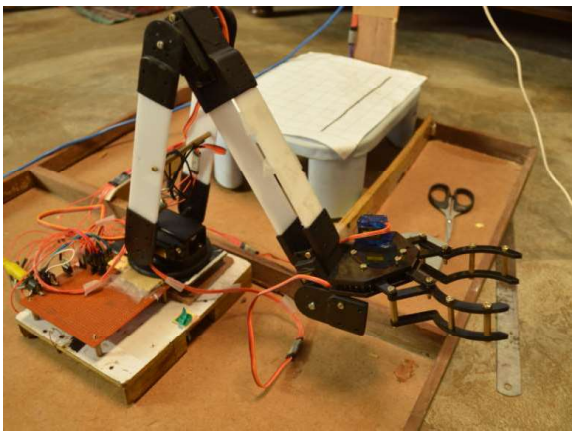
(a) Perspetive Shift
(b) Segmented Image
(c) Image after Centroid Detection

The HSV values of the raw orange in this case falls outside the typical values, leading to incorrect thresholding. Better lighting conditions can help overcome this problem.

4.1.5. SORTING - MOVEMENT OF THE ROBOTIC ARM

To measure the performance of the designed arm, its movement and task success rate of placing an orange correctly was observed by iterating through various possible variations of the orange arrangements. The complete movements of the arm were divided into two categories for better diagnosis of the results –

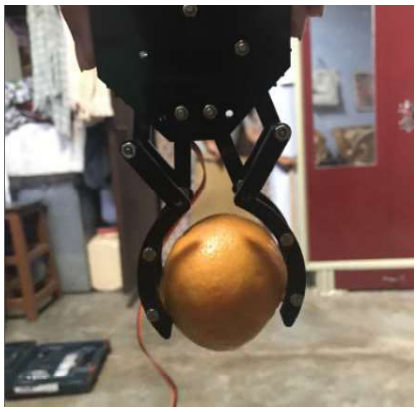
1. Inverse Kinematics



The movements which position the end effector of the arm in the desired position after a centroid coordinate is returned from the image processing part of the software come under this category.

A success rate of around ~90% was observed at this stage. The arm was able to reach almost all the points of the workspace area with success, and was only limited by motor performance for the extreme corners of area near to the camera stand.

2. Picking Oranges



After reaching a desired coordinate in the workspace area, the success rate of picking up oranges was observed. The claw could grip the oranges with ~75% success rate. The claw is easily able to pick an orange in situations where there are no other oranges in its vicinity. In closed packed situations, the claw movement often got hindered by the nearby oranges, causing either a failure of the complete claw motion, or an imperfect grip due to collisions. There were times when the grip was successful even in closely packed situations but most of our failures occurred in these situations.

Although, a software approach was used to improve the servo's abrupt movement when reaching a particular pulse width, this caused the motors to give lesser torque than it rated value. This leads to the movement being slow when the motors are working against gravity, and are fully extended. We can overcome most of the limitations and increase the success rate by using digital servo motors for precision with joint encoders to give way for incorporating a feedback loop on the motor angle. This would greatly improve both the stability and precision of the arm's movement, thus achieving industry grade performance in terms of task success rate.

Chapter 5
CONCLUSION AND FUTURE WORK

We have successfully implemented the hardware design and software architecture for a Robotic Arm intended to sort oranges to automate processes in an industrial environment. Oranges kept in the base to be sorted are successfully sorted by our robotic arm. The image has been taken by the Raspberry PI Camera and further the processing has been done by the Raspberry PI processor. Watershed algorithm has been efficiently implemented and it gives really good results for separating oranges into ripe and raw. The motion of the arm is controlled by the inverse kinematics. Number of iterative testing is done and the results are satisfactory. The software is implemented in a modular fashion for easier debugging and better control. We have also learned the nuances of hardware design and manufacturing through this project.

Due to the limitations of the electronic components used and the manufacturing processes, there are several limitations faced in the real world performance of our project, which are discussed below:

1. Good servo motors essential for precise control were out of our budget, thus, we are limited in our reach and weight of payload by the torques of our motors. This also leads to jittery motions as our motors are always close to their limits.
2. We decided to skip using a complex gear assembly to couple our motor with the mechanical arm because it would have taken the focus of our project away from our scope. Using a better coupling method would have made the movements of our arm more professional and easier to control.
3. Although the Raspberry Pi board is great for image processing operation, it has its limitation in simultaneously generating different PWM pulses for simultaneous motor control. Due to this, we can only move one motor at a time. This significantly increases our time to reach a destination arm position.
4. Due to all these limitations, coupled with a lack of parallel processing, the overall time is more than would be required in a practical and professional setting.

All these limitations stem from being out of scope for a student project. Our design and software works satisfactorily in terms of the actual objective of the project. This can be considered as a working prototype of what could be an actual product to help automate menial jobs in concerned industry. Major concern in this project is quick automated sorting of fruits on the basis of color maturity. The proposed project has come out to be successful in sorting raw and ripe oranges. Further future aim is set for stability analysis of the robot using the Lagrange's equations. Also more rigid contouring techniques can be employed so as to segregate fruits placed really in congestion. In future we require to work further on image processing algorithms and kinematics of the robot. Even if we are doing the best now in the given time slot, we can always speed up the process further and further by employing newer algorithms and techniques.

Chapter 6
REFERENCES

COMPUTER VISION

- http://docs.opencv.org/3.2.0/d3/db4/tutorial_py_watershed.html
- Basic object detection : Siraj Raval Object Detection https://www.youtube.com/watch?v=OnWIYI6-4Ss&feature=youtu.be
- http://stackoverflow.com/questions/26932891/detect-touching-overlapping-circles-ellipses-with-opencv-and-python
- http://stackoverflow.com/questions/25667053/find-overlapping-complex-circles-with-opencv
- http://docs.opencv.org/trunk/da/d6e/tutorial_py_geometric_transformations.html
- http://www.pyimagesearch.com/2015/11/02/watershed-opencv/

SERIAL ARM MANIPULATION

- Robot Kinematics, Forward and Inverse Kinematics by Serdar Kucuk and Zafer Bingul
- CIRP Annals - Manufacturing Technology, Volume 48, Issue 1, 1999, Pages 345-350 Putting Parallel Kinematics Machines (PKM) to Productive Work, F. Rehsteiner [a], R. Neugebauer [b], S. Spiewak [c], F. Wieland [b]
- Zoran PANDILOV, 2. Vladimir DUKOVSKI, COMPARISON OF THE CHARACTERISTICS BETWEEN SERIAL AND PARALLEL ROBOTS , University “Sv. Kiriil I Metodij”, Faculty of Mechanical Engineering-Skopje, Karpos Ii B.B., P.O.Box 464, Mk-1000, Skopje, Republic of MACEDONIA
- ACTA TEHNICA CORVINIENSIS – Bulletin of Engineering, Tome VII [2014] Fascicule 1 [January – March], ISSN: 2067 – 3809