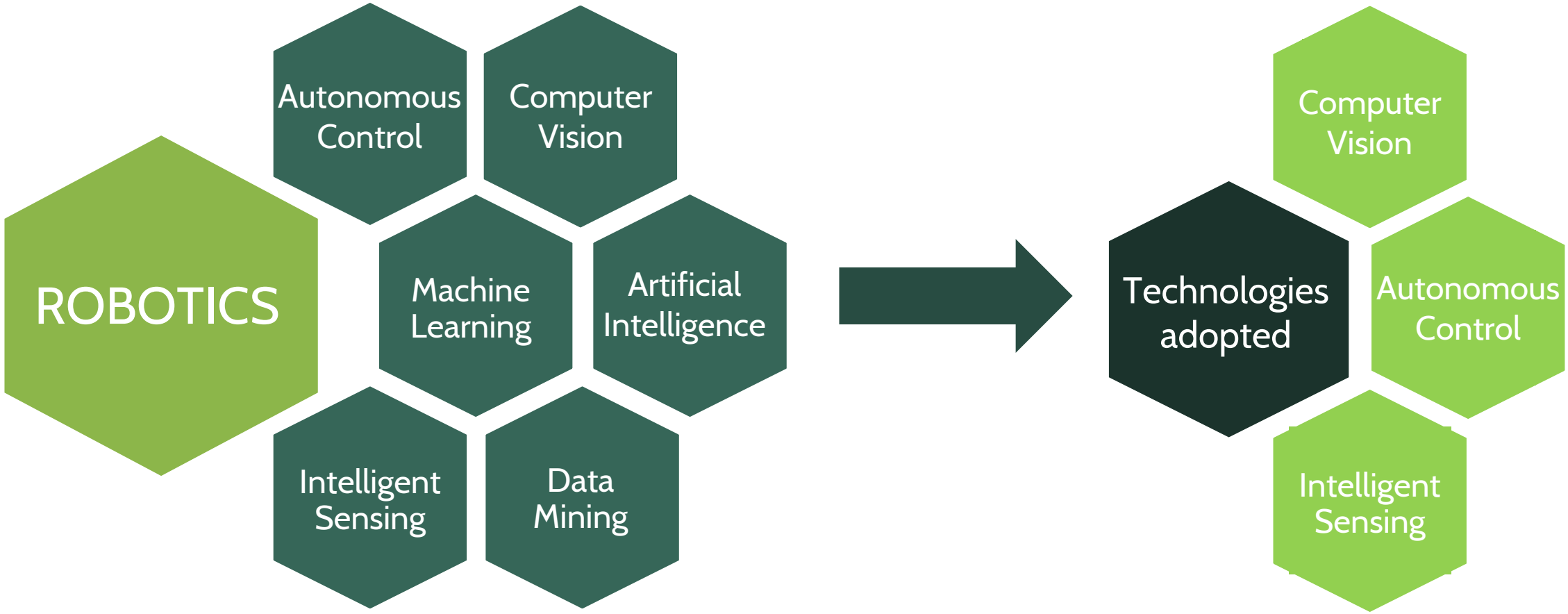
ROBOTIC ARM FOR SORTING RAW AND RIPE ORANGES USING COMPUTER VISION

MAJOR PROJECT - ELECTRONICS AND COMMUNICATION ENGINEERING DEPARTMENT, MANIT BHOPAL

INTRODUCTION

- Sorting of fruits is repetitive and time consuming when done manually.
- It also requires skilled personnel and in many regions there are shortages of skilled manpower for various reasons.
- In 2015, India was ranked second in terms of fruit production in the world, just after China.
- This has created huge demands for automating the classification and sorting of fruits effectively and efficiently, in super market and agro packaging industries in India.

SUGGESTED SOLUTION



OUR APPROACH TO THE PROBLEM

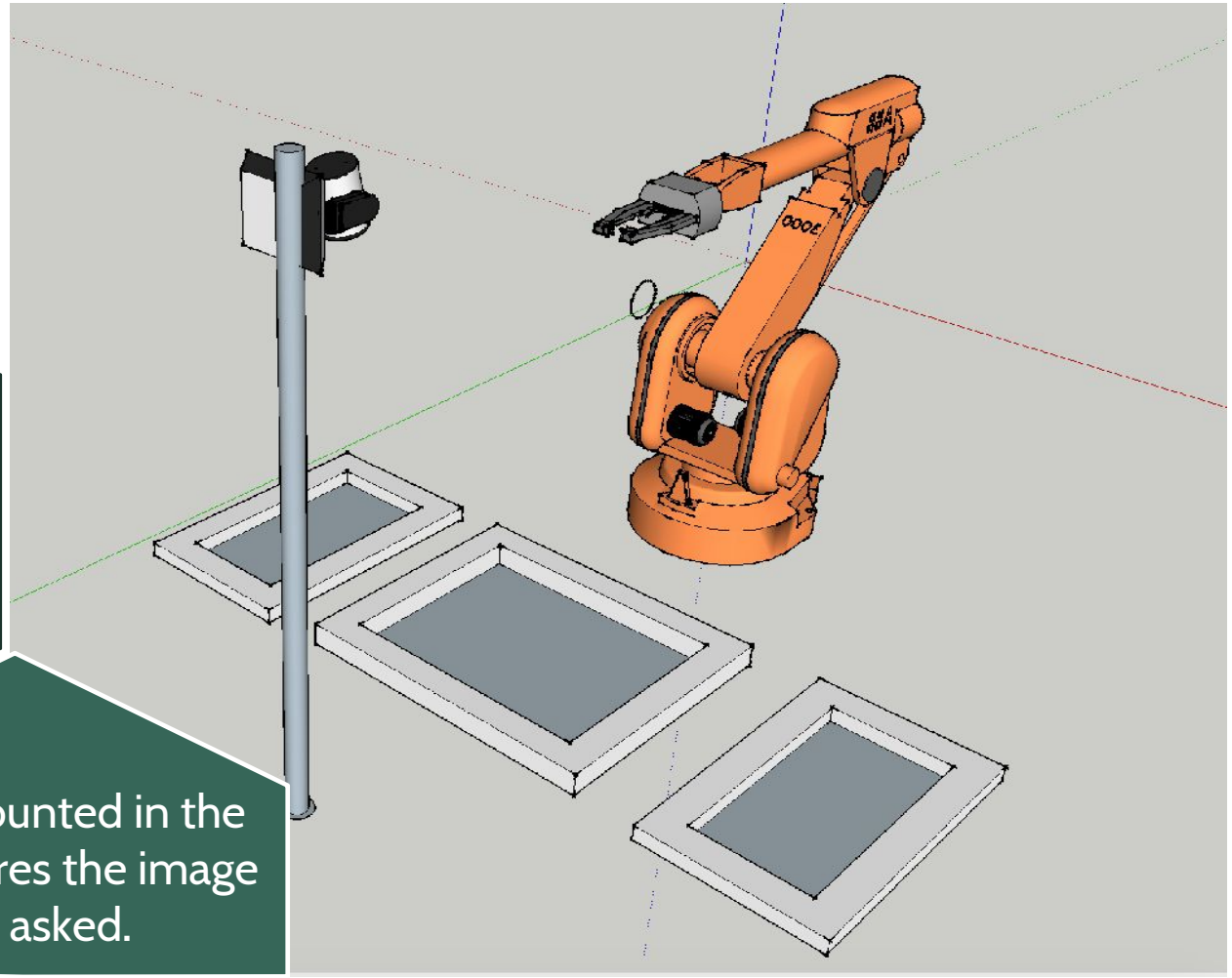
A ROBOTIC MODEL COMPRISING

- 4 DOF ROBOTIC ARM
- CAMERA MOUNT
- RASPBERRY PI INTERFACE

MODEL DESCRIPTION

- We use a RGB Camera (Raspberry Pi Camera) connected to a processor (Raspberry Pi 2)
- Processor drives 5 servo motors, which form the robotic arm and the gripper
- The camera is mounted on a separate stand with the processing board
- In between, there will be 3 boxes for keeping ripe, unsorted and raw fruits respectively
- Power supply will be 5V, 2A given by mobile adapter

GRAPHIC MODEL



CAMERA MOUNT

Camera mounted in the stand captures the image when asked.

Pick up and placement of oranges will be governed by the coordinates calculated by the image data captured by the camera

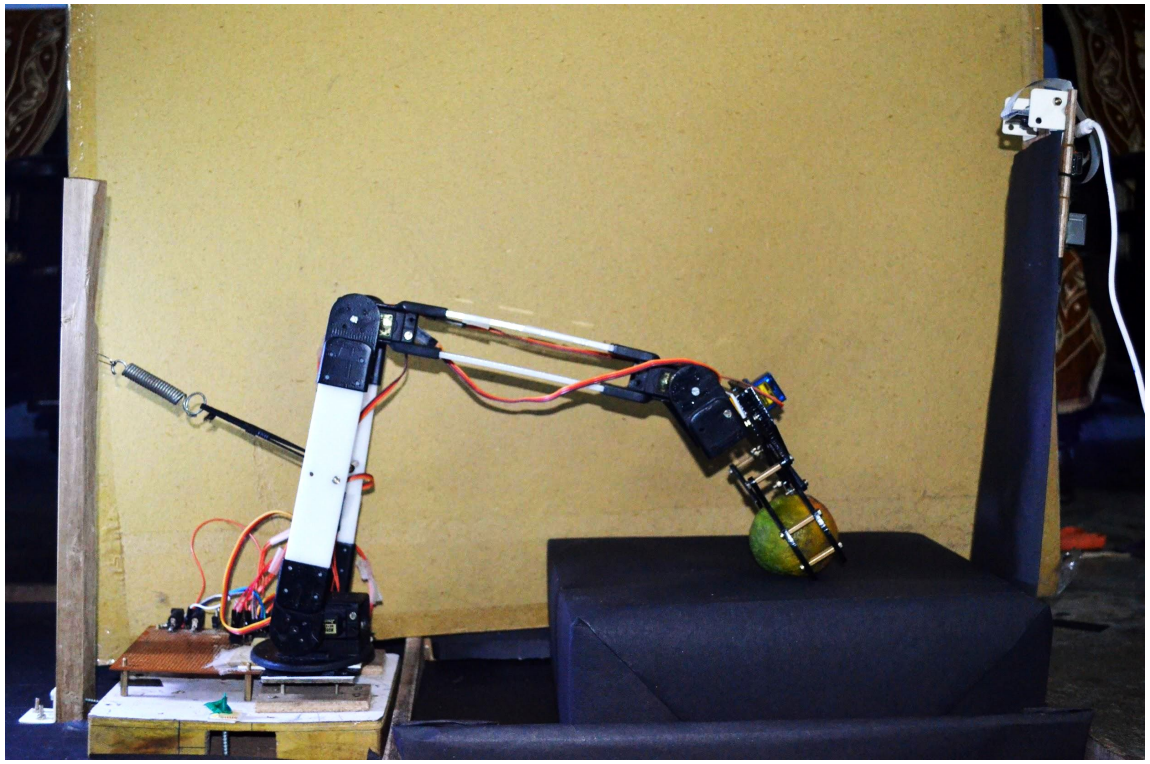
4 DOF ROBOTIC ARM

Arm will be used to pick up and move the oranges from one container to another

MODEL REALISATION

ROBOTIC ARM

It is the complete assembly of arm along with a circuit to safely power the motors.

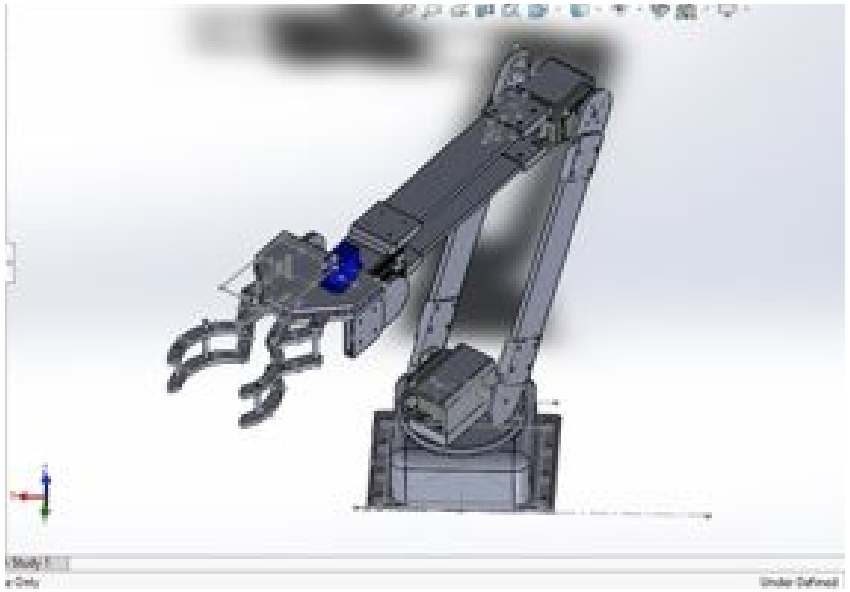


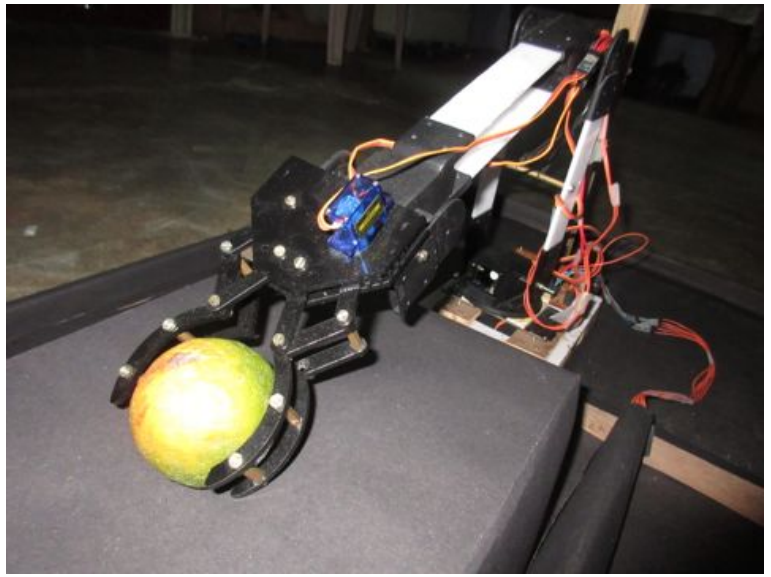
CAMERA MOUNT

It houses the Raspi Camera, and the Raspberry Pi 2 board used for processing.

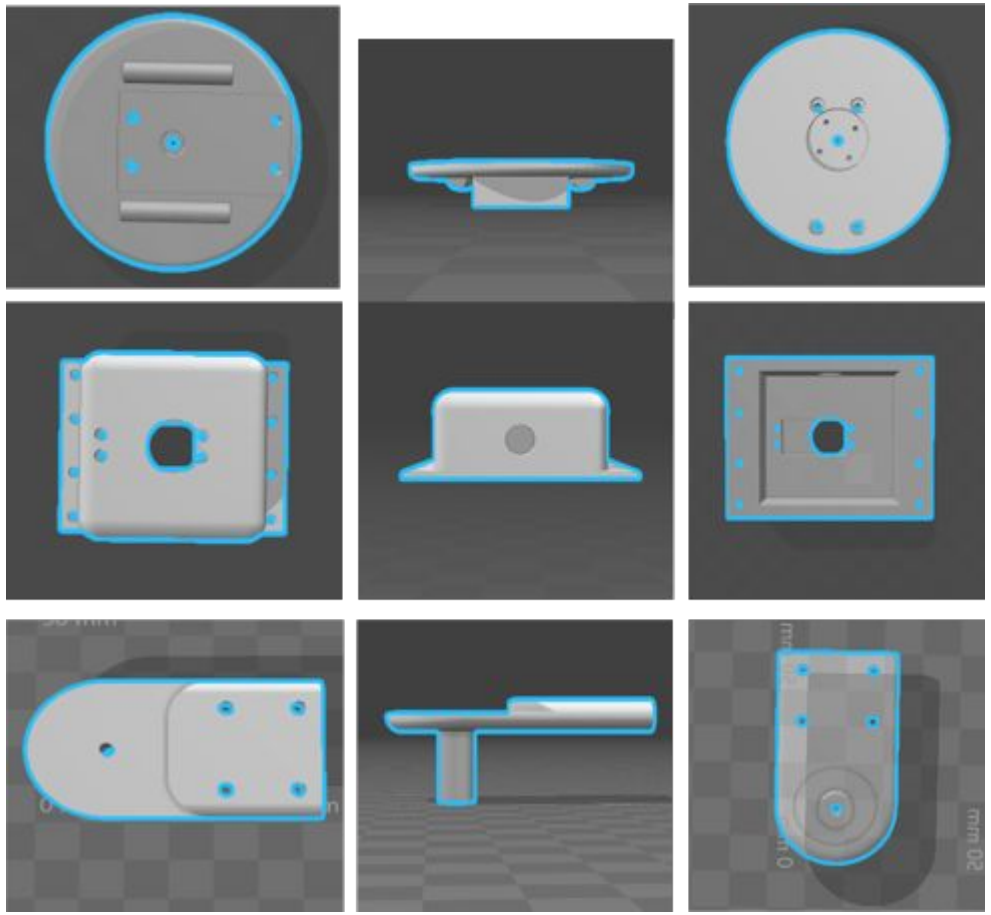
DESIGN AND ASSEMBLY OF THE ROBOT ARM

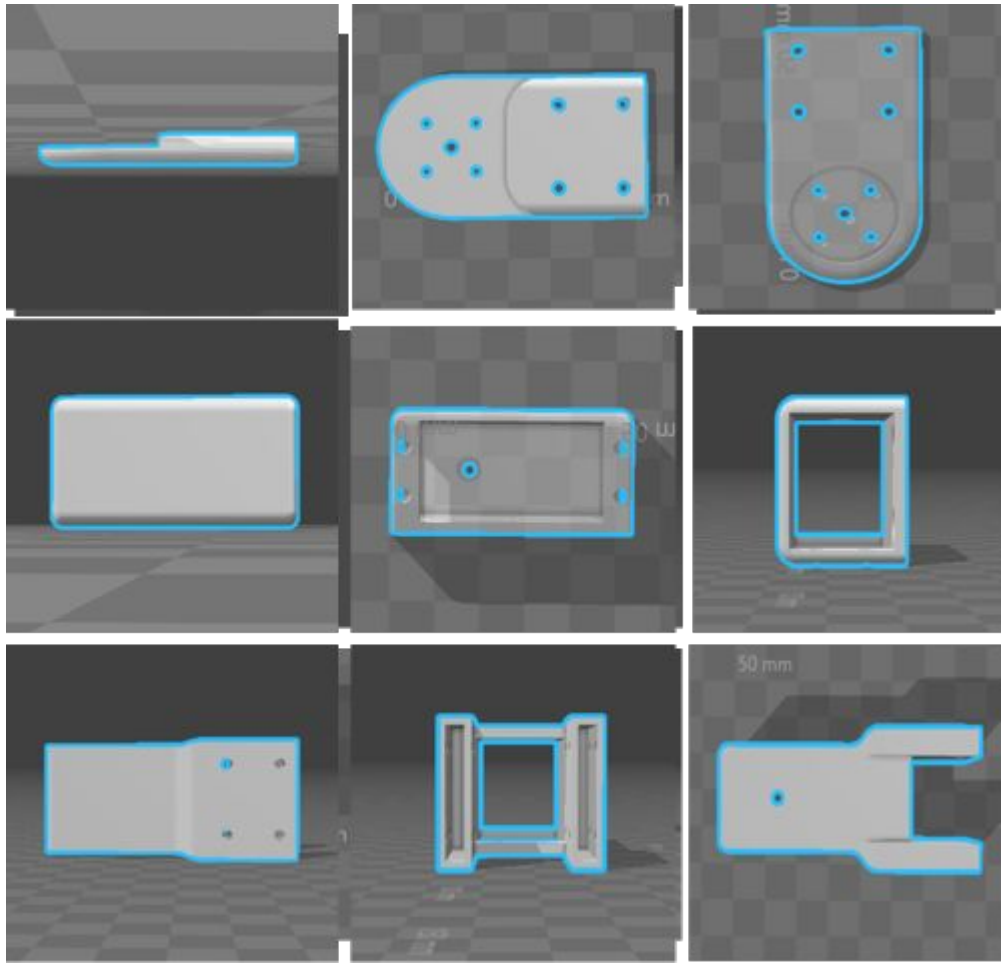
- All the parts used for the assembly of the arm were designed by us using SOLIDWORKS 3D modelling software.
- The designs are realised using 2 methods of printing: 3D parts for the joints of the arm, and CNC Laser cut parts for the gripper and arm links from 4mm Acrylic sheet.
- We have successfully realised a 4 DOF Robotic Arm and a gripper to hold circular objects such as fruits.



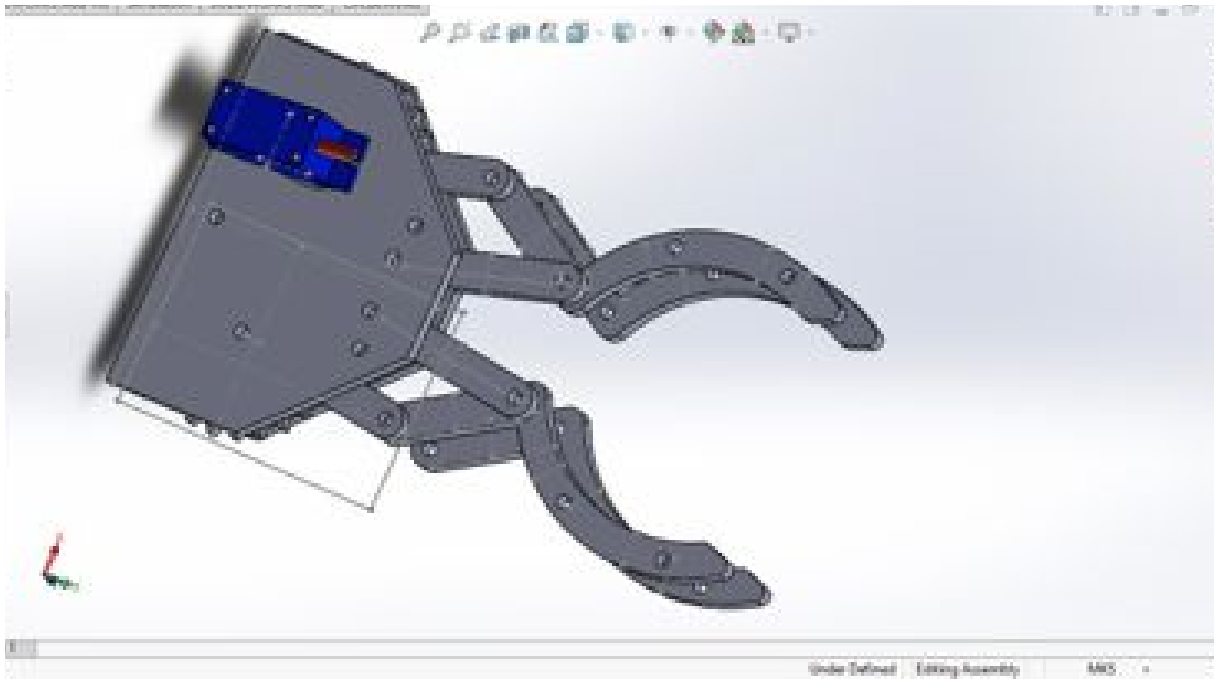


DESIGN AND ASSEMBLY OF THE ROBOT ARM

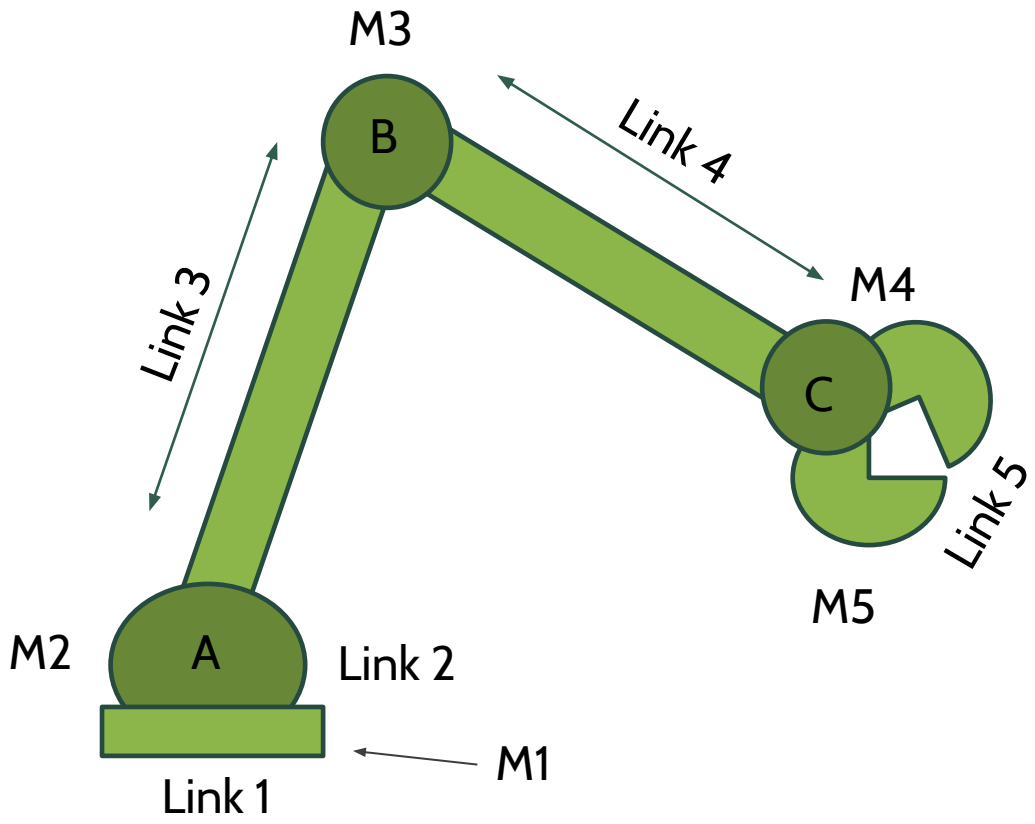




DESIGN AND ASSEMBLY OF THE GRIPPER CLAW



DEGREE OF FREEDOM OF THE MODEL



Degrees of freedom (DOF) are the set of independent displacements and/or rotations that specify completely the displaced or deformed position and orientation of the body or system. The degree of freedom (DOF) for planar configuration as shown in figure is computed using Gruebler's equation as,

Number of links, n = 5

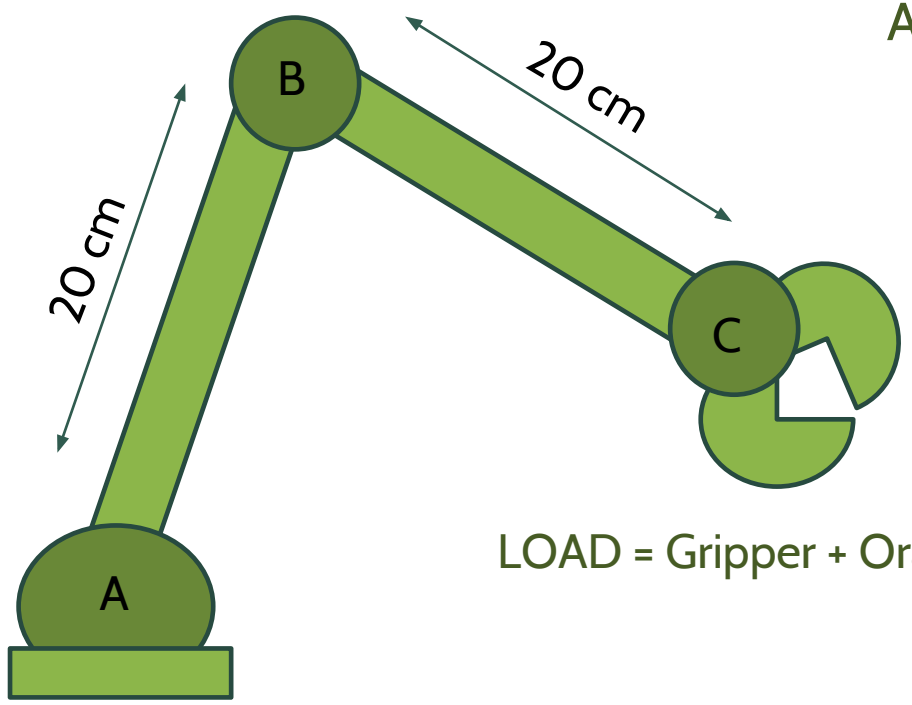
Number of lower pairs, M = 4

Number of higher pairs, h = 0

DOF = 3(n-1) – 2M– h = 3(5-1) – 2(4) – 0 = 4

As we have 4 actuators and 4 DOF, the robotic arm is non-redundant.

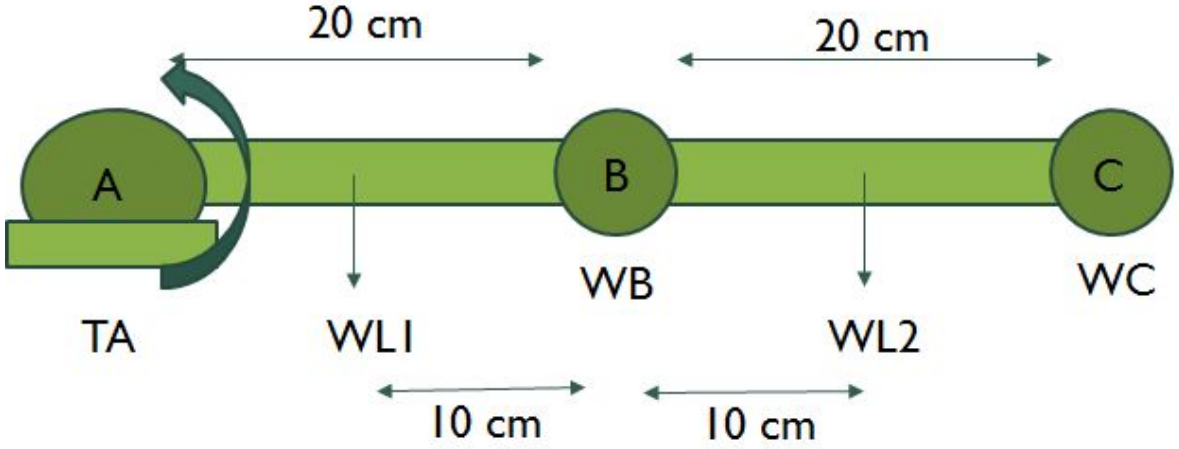
LOAD ANALYSIS



LOAD = Gripper + Orange

Average weight at C = Weight of gripper + Load
= Weight of (M4 + M5 + Material) + Load
= 10 + 55 + 100 + 130
= 295 gm
~ 300 gm

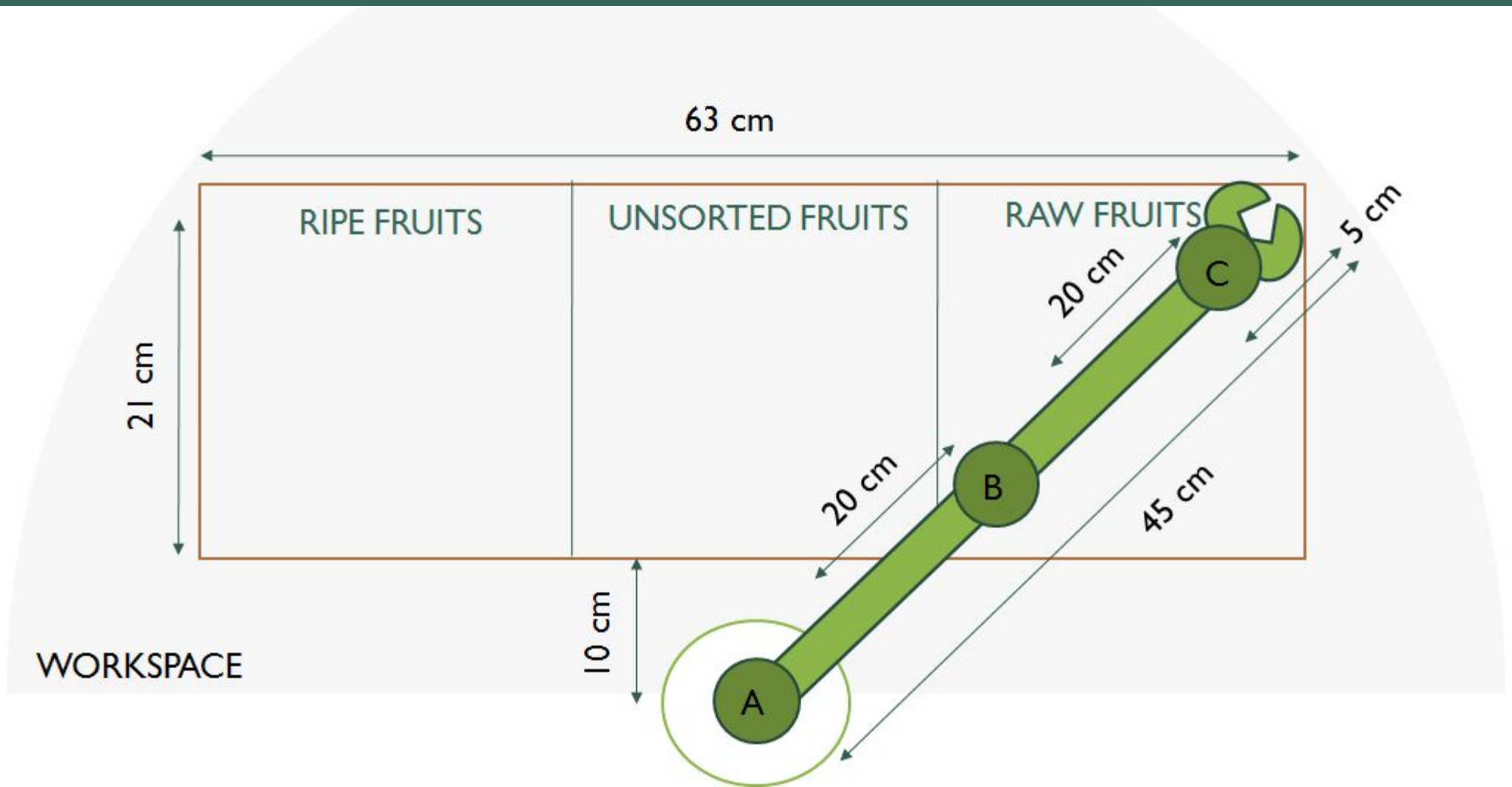
WORST CASE TORQUE CALCULATION



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

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

MODEL SCHEMATIC TOP VIEW



FORWARD AND INVERSE KINEMATICS

The transformation matrix for each joint, transforms the coordinates of its previous link to its next link. For each joint, this matrix is represented by,

$$^{j-1}A_j$$

The forward kinematics is then given by the multiplication of all the transformation matrices,

$$\xi_E \sim {}^{0}T_E = {}^{0}A_1.{}^{1}A_2.{}^{2}A_3$$
$$\xi_E = K(q)$$

After obtaining the forward kinematics equation, we can calculate the inverse kinematics as,

$$q = K^{-1}(\xi_E)$$

Using this inverse kinematics equation, we can calculate the joint angles required for any particular end effector pose (ξ_E) required.

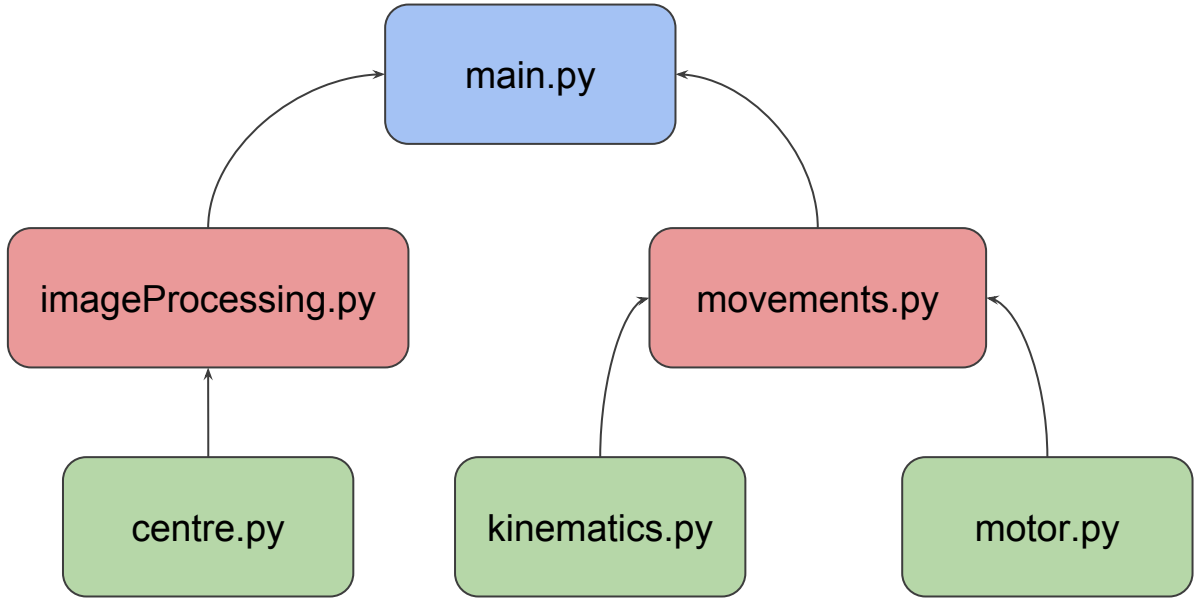
SOFTWARE DETAILS

1. Capture Image.
2. Apply perspective shift, smoothing
3. Use segmentation techniques to separate into two images:
   a. green : for raw fruits
   b. orange : for ripe fruits
4. Calculate centroids of each fruits in respective images and store them separately
   a. connected-component analysis (blob detection)
5. First pick up all the raw_fruits and place in Destination 1 (hard coded coordinate for raw fruits box.)
   a. Use inverse kinematics equation to correctly move to the required place
6. Then pick the ripe_fruits and place in Destination 2 (hard coded coordinate for ripe fruits box.)

Complete code is written in Python 2 using OpenCV libraries for Image Processing and is run on Raspberry Pi 2 board.

CODE ARCHITECTURE

- 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



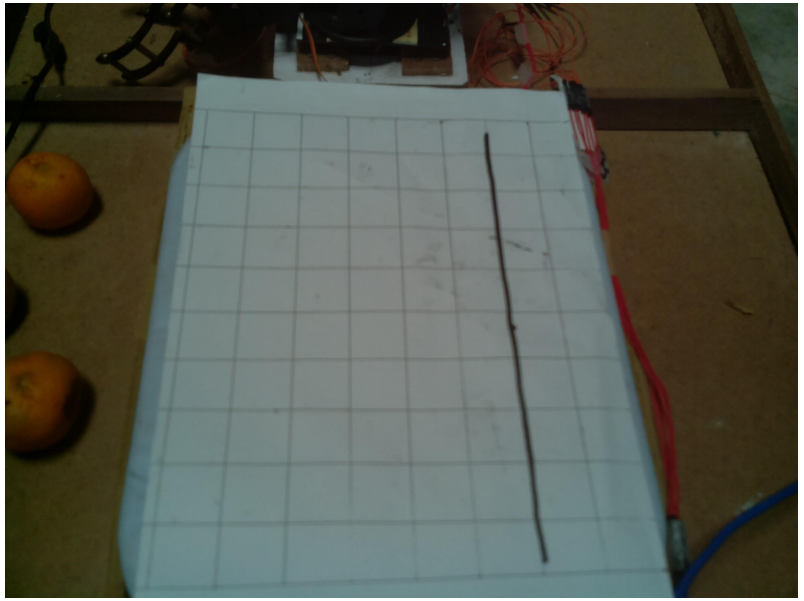
MODULE DESCRIPTION

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

IMAGE PROCESSING ALGORITHMS

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.

a. **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

IMAGE PROCESSING ALGORITHMS

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.

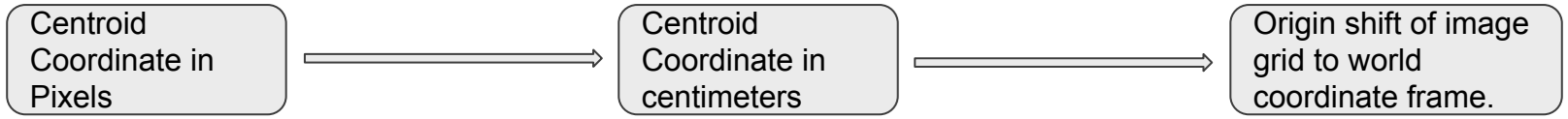
a. **Segmentation** is done by converting the raw image into HSV and using the correct Hue channels to create filters for green and orange color, indication different maturity level of the oranges.

Image after perspective correction is applied.

Marked centroids of ripe oranges.

CENTROID COORDINATE PROCESSING

TRANSFORMATION OF CENTROIDS: 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.




```
19  def transformToWorld(centroid):
20      # size of the image in centimeters
21      widthCM = 20
22      lengthCM = 24
23      # size of image in pixels
24      IMG_LEN = 1000
25      IMG_WIDTH = 800
26      # distance of base of robot from grid of orange
27      baseDistance = 10
28
29      # convert pixel coordinates to centimeteres
30      coordinateCM =[widthCM/IMG_WIDTH*centroid[0], lengthCM/IMG_LEN*centroid[1]]
31      # convert centimeters coordinates into world coordinate frame
32      worldCoordinate = [widthCM/2 - coordinateCM[0], baseDistance + coordinateCM[1]]
33
34      return worldCoordinate
35
```

KINEMATICS

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.

```
36  def calculateDestinationAngles(coordinate):
37      ## Calculate joint angles from world coordinates. Inverse kinematics
38  
39      # z coordinate of end effector
40      heightEndEffector = PLATFORM + CLAW_LENGTH + ORANGE_RANGE
41  
42      # 3D coordinate of the end effector
43      endEffectorCoordinate = np.array([coordinate[0], coordinate[1], heightEndEffector])
44      # base coordinates
45      baseCoordinate = np.array([0, 0, HEIGHT_BASE])
46  
47      # base of triangle between arm1, arm2 and gripper
48      triangleBase = np.linalg.norm(endEffectorCoordinate - baseCoordinate)
49  
50      # calculate joint angles
51      angleBase = math.atan2(coordinate[1], coordinate[0])
52      angleArm2 = calculateAngle(ARM1, ARM2, triangleBase)
53      angleArm1 = calculateAngle(,,ARM2)
54      angleGripper = math.pi/2 + math.pi - (angleArm1 + angleArm2)
55  
56      return [angleBase, angleArm1, angleArm2, angleGripper]
57
```

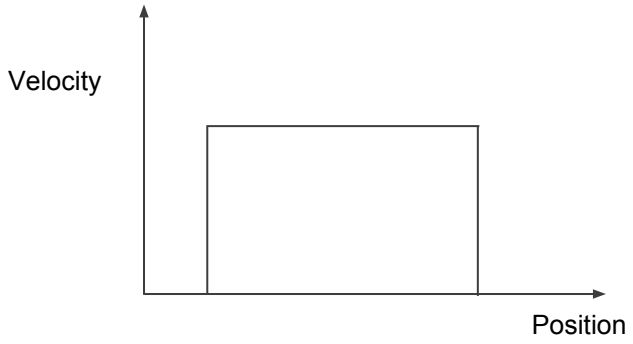
MOTOR CONTROL

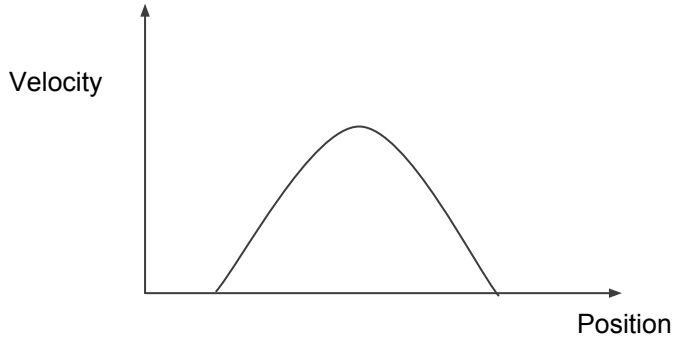
CONVERSION OF ANGLES TO PULSE WIDTHS: The motor control of the servo is done by PWM and thus, the angle calculated in the previous step is converted into the PWM pulse value required to reach the particular angle. We have created the conversion functions by practical experimentation of the motor's profile.

```
58  def calculatePulseWidth(angles):
59      # separate each joint's angle
60      baseTheta = angles[0]
61      arm1Theta = angles[1]
62      arm2Theta = angles [2]
63      gripperTheta = angles[3]
64
65      # convert angle to the pulse width required to reach that angle. Calculated by trail and error from practical observations.
66      # Completely dependent on how motors are connected to arm.
67      pulseWidthBase = 8.55 * math.degrees(baseTheta) + 500
68      pulseWidthArm1 = 7.77 * math.degrees(arm1Theta) + 1510
69      pulseWidthArm2 = -(35/3) * math.degrees(arm2Theta) + 2500
70      pulseWidthGripper = 0
71
72      pulseWidths = [math.floor(pulseWidthBase), math.floor(pulseWidthArm1), math.floor(pulseWidthArm2), math.floor(pulseWidthGripper)]
73
74      return pulseWidths
75
```

MOTOR CONTROL

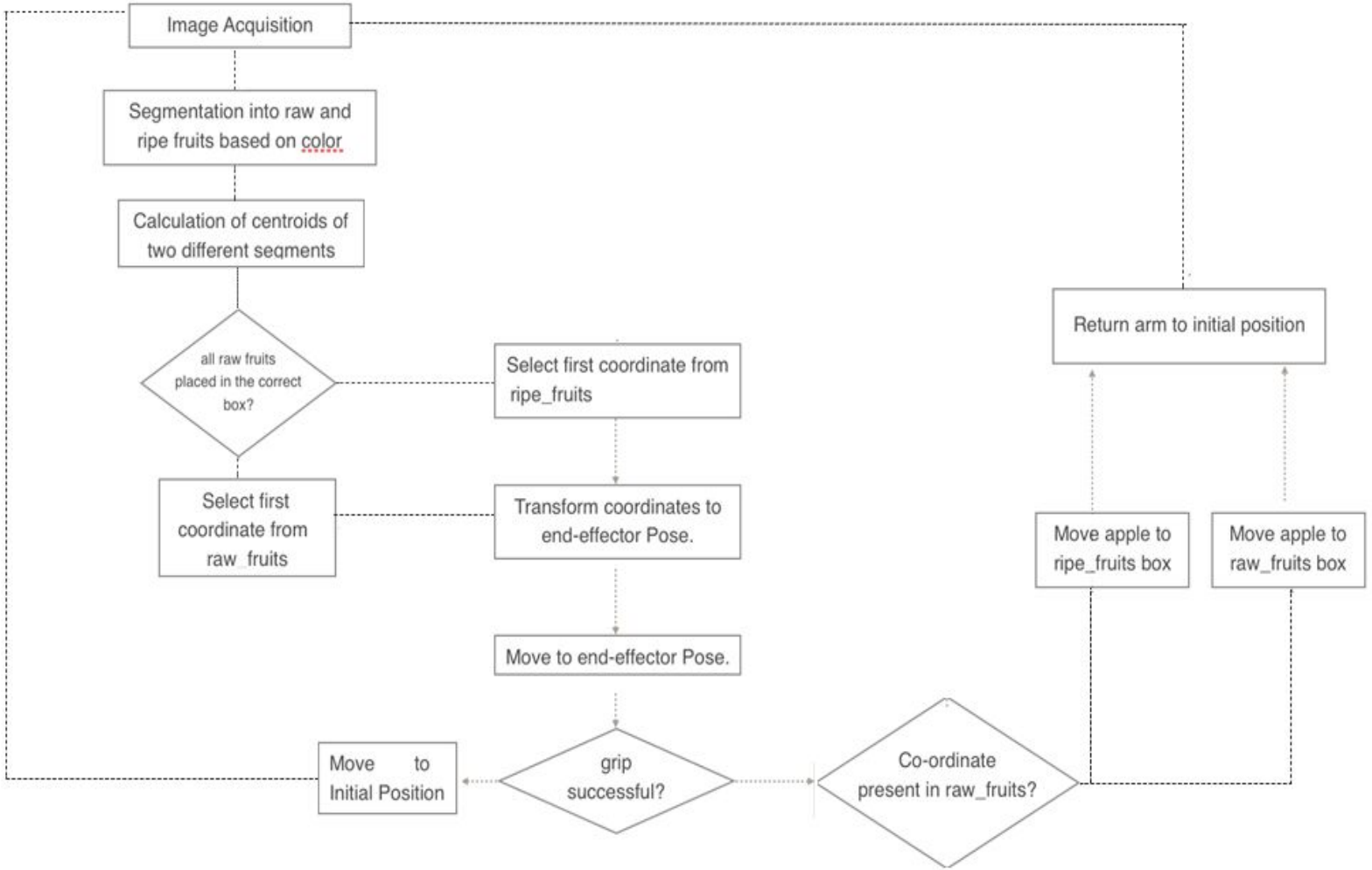
VELOCITY AND ACCELERATION PROFILING: The default property of servo motors is such that when we give it as input the destination pulse width, it tends to rotate at full speed. This leads to very abrupt motion of the whole robot arm. We have created functions to achieve velocity and acceleration control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





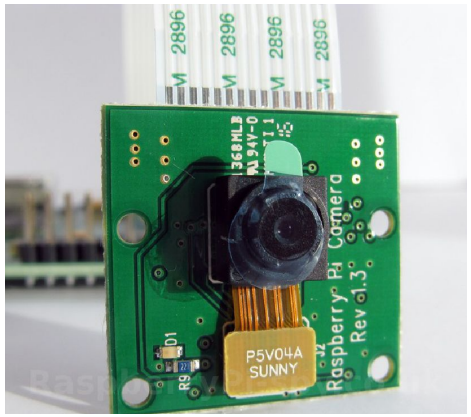

```
87          # while position of motor is between initial and destination position
88          while min(initial, destination)<= position <= max(initial, destination):
89              # Mid point calculation of motion for changing accelaration
90              middle = initial + (destination - initial)/2
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)
```

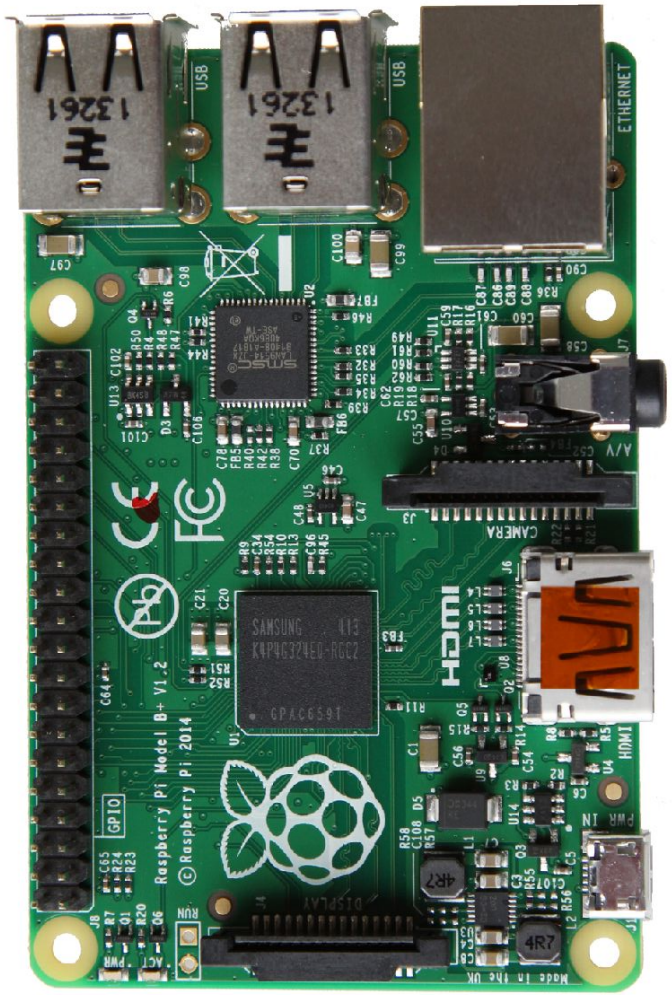
PROGRAM FLOW DEFINED IN MAIN FUNCTION



HARDWARE AND MATERIAL USED

- Camera - Raspberry Pi Camera 5MP resolution OmniVision 5647 Sensor
- Processor - Raspberry Pi 2 A 900MHz quad-core ARM Cortex-A7 CPU
- 1 GB RAM
- 16 GB MicroSD Card
- Servo Motors of required torque (4 x MG996, 1 x MG958, 1 x MicroServo)
- Power : Custom Built 4.8V 2.1A micro USB power supply
- Acrylic plastic sheet used for laser cutting the proposed arm structure





CONCLUSION

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. 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.

PROJECT TEAM

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