Automation of Unloading Graincars using "Grain-o-bot"

by

Aravind Mohan Lokhamoorthi

A Thesis Submitted to the Faculty of Graduate Studies of
The University of Manitoba
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Biosystems Engineering
University of Manitoba
Winnipeg, Canada
R3T 5V6

THE UNIVERSITY OF MANITOBA
FACULTY OF GRADUATE STUDIES
COPYRIGHT PERMISSION PAGE

Automation of Unloading Graincars using "Grain-o-bot"

By

ARAVIND MOHAN LOKHAMOORTHI

A Thesis/Practicum submitted to the Faculty of Graduate Studies of The University of Manitoba in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

ABSTRACT

Large quantities of bulk grain are moved using graincars in Canada and other parts of the world. Automation has not progressed significantly in the grain industry probably because the market is limited for automated systems. A prototype of a robot ("Grain-o-bot") using machine vision to automatically open and close graincar hopper gates and detect the contents of the graincar was built and studied. The "Grain-o-bot" was a Cartesian robot equipped with two cameras and an opening tool as the end-effector. One camera acted as the eye to determine the sprocket location, and guided the end-effector to the sprocket opening.

For most applications, machine vision solutions based on pattern recognition were developed using images acquired in a laboratory setting. Major constraints with these solutions occurred when implementing them in real world applications. So the first step for this automation was to correctly identify the hopper gate sprocket on the grain car. Algorithms were developed to detect and identify the sprocket under proper lighting conditions with 100% accuracy. The performance of the algorithms was also evaluated for the identification of the sprocket on a grain car exposed to different lighting conditions, which are expected to occur in typical grain unloading facilities. Monochrome images of the sprocket from a model system were acquired using different light. Correlation and pattern recognition techniques using a template image combined with shape detection were used for sprocket identification. The images were pre-processed using image processing techniques, prior to template matching. The template image developed from the light source that was similar to the light source used to acquire

images was more successful in identifying the sprocket than the template image developed using different light sources.

A sample of the graincar content was taken by slightly opening and immediately closing the hopper gates. The sample was identified by taking an image using the second camera and performing feature matching. An accuracy of 99% was achieved in identifying Canada Western Red Spring (CWRS) wheat and 100% for identifying barley and canola.

ACKNOWLEDGEMENTS

With profound indebtedness, I wish to acknowledge my deep sense of gratitude to Dr. Digvir Singh Jayas for giving me an opportunity to pursue my doctoral research under his supervision, his excellent guidance, and support.

I am very grateful to Late Dr. Lakhdar Lamari for being in my research committee, and for his valuable suggestions in image processing techniques. I am also thankful to my Advisory Committee members Dr. N.D.G. White, Dr. M.G. (Ron) Britton for the encouragement and support throughout my program.

I thank the Biosystems Engineering Department staff members and technicians for their support and help.

I wish to acknowledge the support and cooperation I received from my fellow researchers Dr. Chitra Karunakaran, Dr. Fuji Jian, Dr. Suresh Neethirajan, Dr Manickavasagan Annamalai, Dr. Vadivambal Rajagopal, Dr. Mahesh Sivakumar, Ms. Sathya Gunasekaran, Mr. Chelladurai Vellaichamy, and Mr. Senthil Thiruppathi.

I also wish to express my gratitude to my parents, for their support and wish to thank my daughter Smrti and my beloved wife Vandhana for their moral support and being my strength in all situations.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

Canada is known worldwide as a consistent supplier of quality grains (cereals, oilseeds, and legumes) (CIGI 1993; Anonymous 2011a). In Canada about 52.7 Mt (million tonnes) of grains are produced annually and about 60-70% of the product is exported around the globe (FAOSTAT 2009). Grain produced on farms is usually stored in bins. There are approximately 120,000 grain-producing farms in Canada. Grain not used for consumption on the farms is transported from the bins to primary or country elevators to process and transfer or to terminal elevators (Anonymous 2011b). Primary elevators are those that receive grain directly from producers; process elevators process grain and oilseeds for domestic human consumption; and transfer and terminal elevators handle grain destined for export. Usually, trucks are used to transport the grain from the farms to the primary elevators. Some large producers load grain themselves into graincars (graincars) for direct shipment to terminal elevators. There are around 360 primary elevators in Western Canada (Anonymous 2011c). The primary elevators are situated adjacent to railway lines for ease of loading graincars for transportation. From the primary elevators, grain is transported to terminal or transfer elevators. Terminal elevators hold the grain until the grain is moved for domestic or export use. Grain from the primary elevators is transported to the terminal elevators in graincars. Each graincar holds 98-113 t of grain. The graincars are also called hopper cars as they are equipped with three or four hoppers at the bottom. These hoppers are used to unload grain contained in the graincars (Anonymous 2011d). The process of transporting grain from the primary elevator to a terminal elevator begins by moving one or more graincars to the trackside of the primary elevator. Individual graincars are opened and inspected for

soundness and cleanliness. Following this the loading spout is placed on an opening at the top to fill the grain. An elevation leg lifts the grain to the top of the elevator and drops it into a distributor. The distributor directs the grain to the loading spout from where the grain is filled into the graincar. Once the graincar is loaded with grain, the hatches of the opening are closed and sealed with a metal strip. Once the graincar is filled and secured, an I-90 tag is affixed to the car. Canadian Grain Commission requires the I-90 tag indicating the name and the specifications of the content of the graincar (Anonymous 2011e, 2011f). The identification number of the graincar and the volume of grain are recorded with the grain company and the railway. A passing train usually picks up the graincar and transports it to the elevator or leaves it at a transfer point for another train to pick the graincar and transport it to the terminal elevator. Trains transport about 90 graincars at a time. At the terminal elevators graincars are left in the docks until unloading. Usually 30-40 graincars are combined and unloaded. Opening the hopper gates at the bottom of the graincar and allowing the grain to flow down by gravity into a pit accomplishes unloading of the graincar at the terminal elevator. Prior to unloading, there is no physical verification as to the contents of the graincar. The person opening the hopper gate visually determines the contents and then matches with the information in the I-90 tag.

The pit contains a conveyor belt that transports the grain into the desired bin or processing unit. The unloading process of one graincar is completed in about 6 min. There are a lot of movements in the graincars between the primary elevator and the terminal elevator. As a result, there is a possibility of a mix-up of graincars. This problem

in addition to manual inspection, physical verification at the unloading point at the terminal elevator, creates a possibility of unloading a “wrong” car. The Canadian grain movement system is mainly designed for export and is thus a one-way system. Hence, if a “wrong” car is unloaded, it is very expensive to clean the system and potential exists for contamination of the grain in the elevator.

Another problem in unloading of graincars is the opening of the hopper gates. At present, a special tool is used to open the hopper gates. The tool is operated either manually or semi automatically. Unloading by the manual or the semi-automatic method is accomplished by fitting the tool into a sprocket on the hopper gate opener and turning it clockwise or counter clock wise, to open or close the hopper gate. A few elevators use an operator to control the opener with a joystick. To open the hopper gate, the operator positions the tool to a sprocket using images captured by a camera. Though the process of unloading is semi-automated, the tool and the sprocket are seldom fine-tuned to align perfectly. (The face of the tool is never aligned to the face of the sprocket hole, causing a hammering effect). This causes tool breakage that slows the unloading process. Also, if the sprocket is rounded by the hammering effect, the grain in the graincar remains there for a prolonged time as fixing the sprocket cannot be done at the elevators.

Introducing the “Grain-o-bot” could eliminate the above-mentioned problems. The “Grain-o-bot” is a robot, capable of identifying the sprocket location and aligning the tool to the sprocket and unloading the graincar. Also, by partially opening the hopper gates and immediately closing it will allow collection of a small sample for identifying the

contents of every graincar, prior to unloading. Hence the objectives for this Ph. D. dissertation were to:

(i) review the existing automation systems in the agricultural industry;

(ii) design a “Grain-o-bot” for automation of grain unloading from graincars;

(iii) fabricate the “Grain-o-bot” and interface it with user friendly operating system;

(iv) test the performance of the “Grain-o-bot” (with different variables such as varied lighting conditions, multiple backgrounds and human interventions); and

(v) to calibrate the “Grain-o-bot” to identify different grains.

2. REVIEW OF LITERATURE

2.1. Automation in the Agricultural Industry

Automation is replacing human involvement in many industries. This replacement is needed to reduce the production cost, to raise product quality and to fill a shortage of skilled labor. Agriculture has always been a very demanding industry that uses a very large labour force to fulfill many repetitive and tiresome tasks. However, even the most sophisticated machines, which perform multiple automation tasks in the manufacturing, and mining industries cannot perform the same tasks in the agriculture industry because of the diversity in the agriculture industry (Anonymous 2011b, 2011g). There is diversity both in the product, like non-uniform shape, size and colour and in the environment like varied terrain, ambient conditions, temperature, humidity and dust (Sistler 1987). Imaging devices, image processing techniques and robotics are used to solve many tasks in the agricultural industry. Informed decisions regarding the variation in the environment along with characteristics of the product (size and other physical properties, vibration characteristics, optical properties, and electrical properties), imparted to a robot to accomplish a task, becomes the most viable and optimal automation solution for the agricultural industry to produce more consistent, low cost and high quality products (Bye and Chanaron 1986). Research and implementation have contributed to the advances in the harvesting, post harvesting, and grading sectors of agriculture, and food industries. A review of the various robotic systems and classification techniques is necessary to facilitate the design of an optimal system for the graincar unloading process.

2.2. Robotic Systems in Agriculture

2.2.1. Orange harvesting robot Harvesting of oranges using robotics and machine vision is crucial because the economical system looks to technological progress as a means of reducing production costs and improving the product quality. However, the softness of the fruit, significant variations in shape, size, and colour pose many challenges (Grasso and Recce 1997). A spherical co-ordinate robot consisting of two revolute joints and a prismatic joint was seen to be advantageous for the process of orange harvesting, because of the simplicity of using the visual feedback, from a machine vision system (MVS) located at the end of the manipulator to control the robot's motion. The geometry of the robot was such that, a fruit centred in the image, lay along the axis of its prismatic joint. By having an arm with a cross sectional diameter slightly larger than the fruit, the prismatic joint could be extended in the canopy of the tree, to harvest the fruit (Slaughter and Harrel 1987). Chrominance information from outdoor conditions was used to enhance the contrast between the orange and the other variables like the leaves, branches and the sky. Slaughter and Harrel (1989) used both chrominance and intensity information for the decision making by the harvesting robot. This was implemented using a multivariate statistical pattern classification technique to find an orange and eliminate the risk of the robot picking a limb instead of an orange. Considerable progress with respect to fruit detection and speed was achieved with the exceptions of occluded fruits and fruits with very bright backgrounds (Lu et al. 2011). Allotta et al. (1990) investigated the act of detaching the fruit from the stem during orange harvesting. It was important to cut the peduncle at a certain length from the fruit and not by just grasping the fruit and pulling it to detach it. This act might result in the

peduncle being cut longer and thereby damaging the neighbouring fruit. The method used six degrees of freedom (DOF) robot attached to a three-fingered hand. The efficiency of orange harvesting was improved by using neural networks, given the same complex images, 86% of the fruits were correctly identified, but false positives were a major setback (Recce et al. 1996). Using neural network and shape-recognition algorithm, Grasso and Recce (1997) further improved the overall detection efficiency to 90%, which was comparable to a human operators' recognition rate similar to Hannan et al. (2009) who used segmentation, region labelling, size filtering, perimeter extraction, perimeter-based detection and enhancing the segmentation using the red chromaticity coefficient which in turn enabled adaptive segmentation under variable outdoor illumination correctly and identified 90% of the fruits. The algorithm also included detection of fruits which are in clusters by using shape analysis techniques A laser-based machine vision system, for automated orange detection was studied and tested in real time using AGRIBOT, an orange harvesting robot (Jimenez et al. 1999). Interferences in object recognition due to varied illumination were eliminated when the images with range and reflectance were recorded using an infrared laser-range sensor. The above suggested system was robust than any colour or monochrome-camera-based image system and provided good correct detection rates, and unlikely false alarms, as the variation in illumination or the maturity stages of the fruit did not influence the decision making process.

2.2.2. Tomato harvesting robot Automation of tomato production has been a challenge owing to the non-uniform shape and size factors coupled with the delicate

nature of tomatoes. The diversity of the environment and product demanded a robot with high flexibility and dexterity. When harvesting tomatoes, collision on obstacles like leaves, stems, stem supporting poles and unripe fruits have to be considered. A five DOF robot capable of turning left or right, and up and down about a fixed axis and moving up and down in a traverse motion, moving in and out and bending its wrist, coupled with a 3-D vision sensor consisting of a red and infrared diode attached to the end-effector mounted on a frame was used to study the efficiency of harvesting tomatoes (Gotou et al. 2003). The 3D vision sensor was chosen to make informed judgements on the fruit maturity and the fruit position by measuring the reflected light of the laser beams through a position sensitive device. The robot was situated at the turning back point of a mobile cultivation bed. This position was chosen because open spaces existed on either side of the cultivation bed along with the front portion of the plant for easy approach and reach. Tomatoes, which overlapped each other, were recognised because the peak produced by the reflection of the red and infrared signal was at a maximum at the fruit centre.

A tomato plucking robotic system with a human interface was developed and tested for seniors who were bed ridden but had not lost their passion for gardening (Takahashi et al. 2001). The system consisted of a three DOF robotic arm equipped with a hand system. A scissor tool and a colour camera were attached to the hand. The senior on one end monitored the images, captured by the camera and transmitted to a remote computer near his or her bed side. The senior observed the images on the computer and harvested the tomato by positioning the tomato to the screen centre and clicking on the tomato displayed on the screen. This process sent a signal to the hand system, which

automatically cut the stalk of the tomato by recognising the red colour of the tomato and by automatically centering the recognised object.

2.2.3. Melon harvesting robot Melons are delicate, fleshy and grow scattered in beds either singly or in groups and are usually hidden by leaves. The variation in size, shape and growing location demands a complex robotic harvester. Adding to the complexity is the difference in time for individual melons to ripe, which necessitates evaluating every melon for ripeness before harvesting. A melon harvester was designed and a prototype was built and tested (Edan et al. 1994). The prototype consisted of a robotic arm mounted on a tractor to move in the field and two vision sensors, namely a near-infrared vision sensor on the robot manipulator to provide motion updates and target orientation, and far-vision sensor at the front of the tractor to detect melon locations. The robotic arm was a Cartesian manipulator to position the end-effector at the appropriate position to pick the melon and place it on a trough which was to be replaced by a conveyor in real time. The end-effector was a gripping device capable of initially gripping and holding from under, retaining the melon, rather than gripping continuously, and also had a cutting attachment to detach the melon from the stem. Tests were conducted with grey level intensities, colour images, infrared and laser images, in sunlight and artificial lighting conditions. The fruit detections ranged from 82-88% for colour images, 82-87% for monochrome images and 74% for laser systems. Nearly 80-85% of the fruits were detected and picked under laboratory conditions with artificial lighting. A similar system equipped with two monochrome cameras for the near and far sensing, for visual feedback to pick melons was studied (Edan et al. 2000). In this study

the results of melon picking improved to 85% as opposed to 80% melon picking in the previous experiment.

2.2.4. Grape harvesting robot Flat canopies of grapevines develop on trellis, situated at a height of 1700-1900 mm from the ground. For a labourer working in a grape field it is extremely difficult to perform operations like harvesting, thinning, bagging and spraying. These processes require the labourers to continuously hold their hands raised for extended periods of time. The tasks are more suited for a vision equipped robotic operation, as the fruit identification is much simpler as the fruit hangs downwards; separating itself from other obstacles like leaves and stem. A multi-purpose robot consisting of a manipulator, visual sensor, a travelling device and four end-effectors was tested for the multiple operations in grape production (Monta et al. 1995). The four end-effectors were made based on the physical properties of the plant. The harvesting end-effector had fingers to grasp the fruit bunch and the pushing attachment to incline the branch and a cutter to detach bunches one after another. The thinning end-effector had three parts. The top part separated huge bunches as individual bunches connected by branches, the middle part penetrated the bunch and shed berries to make an even bunch and the lower part cut a bunch to unify the length. The spraying end-effector was designed to spray a target evenly at a constant speed. The bagging end-effector had fingers and a bag feeder to hold on to the top of each bunch and press the bag shut, when the bag feeder pulled up a bag on the bunch, thereby bagging each bunch. The spraying end-effector consisted of a plunger pump to supply the chemical to the end-effector and a manipulator. The results of the study were promising. However studies need to be done

on the safety aspects of the robots, if robots have to work alongside humans in the same field or green houses.

2.2.5. Hand held robot system for floral harvesting Harvesting of flowers requires trained pickers to harvest quality cutting at the required production rate. The pickers execute operations like localizing the flowers, judging the quality of the flowers, determining the cutting spot, followed by cutting and storing the flowers. In order to automate few of the processes, a manipulator was developed to act as a harvesting aid to facilitate the harvesting of Chrysanthemum plants (Rosier at al. 1996). The harvesting aid was capable of storing cuttings in a well-defined position and orientation and also assisted in improving the quality of the cuttings. The harvesting aid was a hand held device with two sets of belts and a cutting device. When the aide was positioned near the plant by the picker, the first set of belts grabbed the stem and brought it under the belt and cut the plant using knives. Once the cutting was accomplished a sensor recognized the presence of the cutting and activated the second belt which in turn moved it away from the cutting point. Path planning routines by varying the belt speeds were used, so as to position the cuttings at a distance of 10 mm from one another. After 50 such cuttings were collected they were transferred to a storage location.

2.2.6. Robot for horticulture Ornamental horticulture is a growing industry. The container plants growing in the nurseries have to be lifted and conveyed to and from trailers. This is a laborious process and requires a large amount of costly unskilled manpower (Schempf and Graham 2002). Hence a system capable of handling 35,000

containers per 8-h day was designed and tested. The system consisted of a self-mobile platform powered by an internal combustion engine and a laser range-finder to guide the system. The system was controlled by a programmable logic controller board fitted to a computer which actuated a set of electro-hydraulic and electro-mechanical systems to grab, lift, lower and transport the containers. The field trials indicated the reliability of handling 29,000 containers in an 8-h period with a failure rate of less than 3%.

2.2.7. Robot for micro propagation Micro propagation is a tissue culture technique to produce a large number of genetically identical plants. Plants when they are 30-40 mm in height are dissected into pieces, which are then grown under appropriate environmental conditions until they are further dissected for multiplication. The task of harvesting, cutting and replanting needs skilled labour and is a very time-consuming process. The use of a Cartesian robot coupled with a colour camera, image processing algorithms and a human operator acting as an interface has assisted in semi-automating the process (Tillet et al. 1990). The operator used a tracker ball to guide the graphics cursor to the cutting point. This information was used to process the appropriate coordinate transform and positional update to facilitate the robot movement. The robot was equipped with a pentagonal turret tool holder as the end-effector to perform variety of operations. These tools assisted in harvesting, dissecting and planting of the cuttings. The harvesting tool consisted of fingers and a cutter to hold on to the shoot after cutting. The dissection tool consisted of a cylindrical blade to excise y-shaped cuts on the plant by a rapid punching and slicing motion. The planting tool consisted of a T shaped wire attached to a motor.

Rotation of the motor caused the stem to be planted into the medium in an upright position.

Manually transplanting of the orchid *Phalaenopsis* plantlets consisted of usually grasping the root or the stem since the leaves were fragile and were easily damaged. Huang and Lee (2009) built a robot to ease this transplantation. The robot was equipped with a gripper suitable for grasping the plantlets and using a binocular stereovision algorithm to compute the 3D coordinates of the grasping point was able to identify 78.2% correct positions.

2.2.8. Autonomous agricultural vehicle Tractors have changed the farming practice of using horses and cattle (Li et al. 2009). The advancement from a tractor is the robotic tractor, where the tractor is automated to do many operations without the involvement of the human work force (Richey and Richey 1986). Many operations like pesticide spraying, fertilization and selective harvesting could be accomplished using robots, rather than using human labour. Usually, autonomous vehicles are tractors, combine harvesters and other farm vehicles equipped with sensors for feed back to the control system to do particular operations which might require the involvement of a person. Automatic turning during harvesting was one of the more complicated tasks in a list of tasks for automating an agricultural vehicle (Ito 1990). The control of the tractor used sensors for various tasks like row detection, measuring feed rate, calculating cutting height and straw feeding position for optimum threshing. But the turning of the harvester was complicated because the machine drifted away partly or fully from the field by the time the automatic turning

sensor acted with feedback information from the row detector sensor. This problem was initially addressed using a training process. An operator made the first run in the training process, so that information of the plant row length was recorded, and using this information the subsequent runs were carried out. A prototype of an autonomous tractor powered by 4 direct current motors and steered by two pulse motors was designed and tested (Toda et al. 1993). It was equipped with two sensors, an ultrasonic sensor and a flux gate digital compass. A fuzzy control model controlled the whole setup. The computer system on the vehicle had two modules, a lower level module and a higher-level module. The lower level module using sensors controlled the driving and the steering using the data received from the high-end module. The high-end module handled complicated decision making and inferences like path planning, based on the data received from the low level module. The vehicle had a -0.007 to 0.01 m error from the goal. The need to reduce a waste in time and money owing to the overlaps caused by the lack in concentration of the agricultural vehicle operators was studied (Klassen et al. 1993). An average of 10 % of the field was being covered twice and could be avoided by the use of an automatic guided system on agricultural vehicles. A reduction in mental fatigue caused by operating the equipment for extended periods of time was seen to be one of the primary reasons for the problem. So a guidance sensor was developed and tested that was capable of demarking tilled and untilled soils and cut standing crops. However as further investigation using high-powered computers was needed and various tests on all conditions had to be performed to fool-proof the system the need for an operator at present was needed. Ollis and Stentzi (1997) describe a vision-based perception system to guide an automated harvester cutting fields of alfalfa hay. The

system was designed to track the boundary between cut and uncut crop, identify the end of a crop row, and note the obstacles in the path of travel. A New Holland 2550 speed rover, retrofitted with wheel encoders and servos to control numerous functions like throttle, steering and cutter bar on the machine, was used. It was also equipped with a GPS receiver and two RGB cameras equipped with auto-iris lenses mounted on either side of the cab roof. The system was adapted to varied crop and lighting conditions. The efficiency of the vehicle was comparable to the human operating speeds. An autonomous vehicle for crop and weed registration with a special emphasis on the control system was studied (Nielsen et al. 2002). The problem of wasteful (excess) spraying of fertilizers, pesticides and herbicides was considered. The system incorporated an autonomous vehicle that mapped the field and later used the map to arrive at the appropriate spraying quantities. The frame-work for the control system consisted of a hierarchical four layer model capable of handling steering control propulsion, motion control, and path execution based on the images collected.

2.2.9. Strawberry sorting robot Strawberry cultivation envelopes a major portion of the fresh fruit market. Since harvesting and sorting are done manually, the trend to cultivate strawberry is slowly diminishing as a result of expensive labour. So to automate the process a machine vision system equipped with a five-arm robot was developed and tested using Akihime variety strawberries (Bato 1999). The machine vision system consisted of a colour camera and a conveyor system to present the strawberries to the camera. An accuracy of 98.6 to 100% detection based on the shape and size features

regardless of the orientation of the fruit was achieved. The robot arm based on the feedback from the image analysis sorted the strawberries accordingly.

2.2.10. Fruit grading robot The operations of grading and sorting have become highly advanced with the recent developments in mechatronics and Robotic technology (Kondo 2009). Grading of deciduous fruits (peaches, pears, and apples) was tested using three, three DOF robots (Kondo 2003). There were six suction cups as end-effectors on two robots and 12 suction cups on one robot. This was combined with an image processing system comprising of 12 colour cameras and 28 direct lights. These in combination worked as follows. Containers holding fruits, which were initially arranged in a set fashion inside the container, were sent to the working area by a pusher. The two three DOF Cartesian robots with six suction pads as end-effectors came down and sucked 5-6 fruits each and transferred the fruits to a halfway stage. The third three DOF with 12 suction cups was a cylindrical robot containing one rotational joint and two prismatic joints. The cylindrical robot, which was the grading robot, sucked up the fruits and activated the cameras that took the bottom images of the fruits followed by capturing images of the four sides of the fruits by rotating the fruits by 270°. Once the images were acquired the grading robot placed the fruits in the appropriate trays based on the results from the acquired images. An accuracy of 85 – 90% was achieved, and the decrease in the accuracy was due to short term, less experienced operators who stopped the robot or the lines when a specific line was full with the same grade and size fruits in it.

2.2.11. Fish processing Present fish processing techniques result in material losses due to current practices of feeding, indexing, holding and cutting the fish. An automated fish

processing system was studied (de Silva 1991). The feeding mechanism currently practised has four labourers at the sorting station. Two labourers align a fish from a holding tank, and the other two labourers push the fish on to the flipper gates while aligning the gills of the fish to reference pins. The fish is then dropped on to a conveyor, which then moves the fish to a cutter. Apart from the workload of positioning 60 fish per minute per person, there existed the possibility of introducing errors in positioning the fish (for example, when the fish was dropped on to the conveyor system, the tossing and turning of the fish could impart considerable error in the position of the fish). Aligning the noses of the fish to a guide plate and measuring the length of the fish with a sensor and using a double indexing mechanism to position the fish with a finer tolerance on the conveyor could simplify this problem. The gill position was determined by using a MVS. Two methods: the boxing method and the directional averaging method were suggested for gill positioning. In the boxing method, the image was first intensity reversed to have a higher intensity and avoid double edges. Following this the gill-edge shadow image was enhanced by low-pass filter to reduce high frequency noise and through a directional high-pass filter in the direction perpendicular to the edge to intensify the edge. The image was then thresholded and a connectivity procedure was used to connect the edges, which had a minimum intensity level to result in a single contour segment. The length by the width determined the gill position. In the directional averaging method, the image was filtered and intensity reversed as per the first method. Following that the grey-level of the 2-D image was averaged onto a single line perpendicular to the nominal direction of the gill edge. The nominal direction was known as a result of the manner in which the fish were fed. The projected average provided a 1-D profile of the image. The signature

possessed a peak at a point corresponding to the gill position, as the curvature of the mid region of the gill edge was very small. This method was faster than the boxing method.

The orientation of the fish utilized three methods. The first method employed a direct geometrical definition of orientation. The second method used the second moment of area to establish orientation. The third method used the orientation of the major axis of the fundamental ellipse that fit the border of the object to define the orientation of the object. The prototype consisted of a feeding system that was composed of a mechanical feeding wheel, guide plate, cylindrical holding pans on the conveyor, sensing device to measure the length of each fish, a piston rod arrangement to laterally fix the fish according to its length and double indexing mechanism to position the fish with finer tolerance. A carousel of positioning cells synchronically moved with the conveyor and each cell was equipped with a pneumatic suction device to hold the fish in position and then cut it at the appropriate position. De Silva and Saliba (1992) developed a gripper mechanism to the existing setup where the conveyor chain transported the fish to the cutter and a driver belt held on to the fish that was being conveyed and cut. A CCD camera attached to the system decided the best lateral position for the cutter blades. A displacement sensor measured the thickness of the fish head and this information was used to drive the fish entry platform. However, the setup had a few drawbacks pertaining to holding the fish. Introducing a modified gripper consisting of two fingers made up by four links and controlled by two actuators rectified these problems. The fingers were able to confirm the varied shape of the fish by the autonomous sequential switching of the actuators between the links. A frictional switch protected the actuator from overloading. This setup was more rugged in the fish processing application. Automated fish processing was further

developed and executed to process fresh Cod with a robotic manipulator and a vision sensor increasing the yield of the process (Buckingham et al. 1996).

2.2.12. Fish catching Fish catching using a robotic manipulator equipped with a net was studied (Suzuki and Minami 2002). The behaviour of the fish and the intelligence of the fish to escape the net were noted. The manipulator had visual sensing coupled with a genetic algorithm to process recorded data on the behaviour patterns of the fish and act accordingly and this was an effective technique in catching fish using a robot.

2.2.13. Broiler singulation The poultry industry, similar to the fish industry faces problems posed by the variations and complexity of the product and thereby hampering the automation of various processes. The problem of automating the process of transferring live broilers from a conveyor to a moving shackle was studied (Lee et al. 1998). Singulating and later orienting a bird is essential for the subsequent processes in the poultry industry. Since both the mechanical forces and the birds' natural reflexes make the problem complex a prototype of a singulator was designed and evaluated. It was observed that the singulating of the bird was influenced by drop-off-height, inclination surface roughness of the entry conveyor, conveyor speed, multiple clustered birds at the entry, illumination, and bird's experience.

2.2.14. Robotic milking Milking of cows presented a bottleneck in the dairy industry owing to the high cost and unavailability of skilled labour. A milking station capable of allowing cows to be milked and identifying the cows that have already been milked by recognising the transponder mounted on their collars was integrated with a robot capable

of searching and localizing the udder and teats with the help of infrared laser sensor (Cosmi et al. 1997). The system was also equipped with biosensors to monitor the animals' physiological parameters and issue an appropriate warning to the farmer thereby improving the health of the cow and food safety. The results were validated and commercial systems were to be developed.

2.2.15. Robotic egg handling In the egg processing industry along the packaging lines a small quantity of eggs was found to be cracked. These damaged eggs have to be removed to maintain the quality of the packaging. Using two robots and a mono chrome camera the damaged eggs were removed (Bourely et al. 1986). The camera recorded the bright lines of the cracks and imparted the information to the robots and the robots in turn removed the damaged eggs. Identification accuracies of 90% were recorded. Uneven lighting conditions were identified to be the cause of the reduction in accuracies. Later investigations carried out using colour cameras and monochrome cameras coupled with ANN provided better rates of detection of fertile eggs, cracks, dirt stains and blood spots (Das and Evans 1992; Patel et al. 1993; Patel et al. 1995).

2.3.16. Grain unloading system The graincars, which arrive at the terminal elevators, are unloaded by opening the hopper gates and allowing the grain to flow down by gravity into a pit. The grain falls from the hoppers through a floor gate and onto a conveyor belt which in turn delivers the grain into the elevator. In order to open the hopper gates during unloading of graincars a special tool is used. The tool is operated either using a graincar gate opener (Fig. 1a) or semi-automatically from a remote cabin

using a joy stick and aligning the opener tool to the sprocket with the images from a camera located near the opener (Figs. 1b, 1c).

(a)

(b)

(c)

Figure 1. Graincar opening. a) Graincar gate opener; b) Semi automated graincar gate opener; c) Operator opening graincar gate using semi-automated opener

Though the process of unloading is partially automated there are a few shortcomings that are yet to be addressed, like the merging of the tool with the sprocket in a non-aligned manner. This is due to human intervention where errors are due to fatigue, and lighting conditions. Also the problem in unloading a wrong car is yet to be addressed.

2.3. Classification

2.3.1. Prune sorting The sorting of prunes is done manually which is a laborious and difficult task. Hence, a system consisting of a vibratory trough and a chute to singulate and feed the prunes to a main conveyor was designed and built (Delwiche et al. 1993). The main conveyor consisted of six circular cross-section plastic belts, arranged to form a U-shaped trough. The belts at the top of the trough moved faster than the belts at the bottom causing the prunes to align with their major axes parallel to the direction of travel. The initial feed at the vibratory trough controlled spacing between the prunes. The aligned prunes moved to an inspection chamber consisting of three, 256 pixel line scan cameras positioned at 120° intervals around the chamber aperture. The speed of the main conveyor was set at 2 m/s to correspond to a sorting rate of 20 fruits per second. A combined sample of naturally conditioned prunes, which contained 28% defective prunes, was tested. A sorting error of 5.6% for good prunes and 10.8% for damaged prunes was achieved.

2.3.2. Rice inspection Packers and stock yard administrators require the total broken kernels in an official inspection of milled rice. A premium is usually paid for whole rice (Sistler 1990). The process of identifying the broken kernels is carried out using sieves and visual inspection and the time taken to make an inspection of a 25 g sample is about 20 min. Hence to automate the process of inspection and to reduce the inspection time a commercial automated imaging grain inspection machine was developed and tested. The rice sample entered through a feeder and onto an endless belt with traverse grooves coupled with an image acquisition device. Using artificial neural networks (ANN) the

possibility of using the machine as an official industrial inspection system was studied (Wishna 1999). The results from the system were comparable to human inspectors and the time to inspect a 25 g sample was reduced to 3 min.

2.3.3. Cereal classification Classification of bulk grain is necessary to facilitate automation at many points along the line of grain movement to facilitate decision making for the next operation. The grain at the receiving pit in elevators is sampled for test weight, varietal purity, soundness, foreign material content, insect infestation, and vitreousness by inspectors (Luo et al. 1999). To automate the testing process investigations on automatic classification of bulk grain, single kernels, variations in cultivars of grain, grades, and levels of insect infestation were studied (Cogdill et al. 2004; Choudhary et al. 2008, 2009; Karunakaran 2002; Karunakaran et al. 2004; Maghirang et al. 2003; Mahesh et al. 2008; Majumdar and Jayas 2000 a, b, c, d; Manickavasagan et al. 2008; Paliwal et al. 2004). Using colour, textural and wavelet features extracted from bulk images of barley, Canada western amber durum (CWAD), Canada western red spring (CWRS) wheat, oats and rye, and using ANN namely back propagation neural network, classification accuracies close to 100% were obtained (Visen 2002). Majumdar and Jayas (2000 a, b, c, d) applied image processing to classify clean cereal grains. They extracted and used texture, colour, morphological features of the colour images and their combinations to classify individual kernels of CWRS wheat, CWAD wheat, barley, oats, and rye. They achieved classification accuracies of 99.7 and 99.8%, respectively for CWRS and CWAD when texture, colour, morphological features were combined and 20 most significant features were used as the input to the

discriminant classifier. Paliwal et al. (2003) classified cereal grains (CWRS, CWAD, barley, oats, and rye) and detected the dockage present in the grain. Morphological, colour, and textural features from 42 digital colour images were used to obtain features and develop a model using a BPNN classifier. The model correctly classified most of the cereals with well-defined characteristics; however, particles with irregular and undefined features gave lower classification accuracy (90%).

Choudhary et al. (2008) used a colour machine vision system to classify CWRS wheat, CWAD wheat, barley, oats, and rye. Extracting colour, morphological, textural and wavelet features from the non-touching single kernels, classification algorithms were developed using linear discriminant analysis (LDA) and Quadratic discriminant analysis (QDA) and different combination of extracted features. Using LDA and combining the colour, morphological, textural and wavelet features a classification accuracy ranging between 89.4 to 99.4% was achieved.

2.3.4. Fish grading Many regulations in the European community specify that all fish pertain to marketing standards and have to be graded for freshness, species and size prior to selling. Sorting of fish by size and species is at present done mainly by hand and is a slow and labour-intensive task (Misimi et al. 2008; Sistler 1990; Strachan and Nesvadba 1990). Automation to identify and quantify a large number of fish consistently and accurately for real time purposes is imminent. Invariant moments, minimum mismatch factor, and shape descriptors were used for classification of six species of fish. The shape

descriptors gave the highest classification of 90% followed by invariant moments with an accuracy of 73%, and the mismatch factor with an accuracy of 63%.

2.3.5. Fish counting Migratory fish in certain rivers are disappearing over the years due to the construction of dams or other environmental factors. Introduction of fish-ways allow the fish to swim upstream and reproduce. The usefulness of fish-ways has to be tested for the effectiveness of the system. Presently the counting of fish species is carried out by manually capturing the fish passing through the fish-ways or by introducing a video camera in the fish-ways. Cadieux et al. (2000) discussed an automated system for counting fish by species. The system comprised of a silhouette sensor to acquire the fish silhouettes. These silhouettes were then processed for fish count and species identification. Classification by the system used a combination of statistical and neural network classifiers. The results obtained were comparable to the recognition rate of an expert who surveyed the video images.

2.3.6. Poultry classification A transportable spectrometer system was developed and tested for classification at slaughter plants with an accuracy of 93% classifying the carcasses as normal, septicemic (blood poisoned), and cadaver classes using the visible and the near infrared regions of the spectrum (Chen et al. 1994). This classification was necessary to eliminate real time organoleptic inspection (smelling) of birds, as this was to become mandatory. Employing the inspectors to inspect all birds would make the job tedious and the processing slow. A poultry carcass screening system for implementation

at slaughterhouses, to facilitate an improvement in consumer confidence and to reduce the workload on human inspectors, was proposed based on the results.

3. DESIGN: CONCEPT AND METHOD

3.1. Graincar Hopper Gate

Each graincar transporting grain from the country elevators to the terminal elevators has three or four hoppers at the bottom. The positioning of the hoppers is to ease grain unloading by gravity. The hopper has a sliding gate to keep the grain secure. The opening mechanism of the sliding gate consists of a rack and pinion attachment, a shaft and a sprocket with a key hole. The rack is mounted to the sliding gate, the pinion to the shaft and is attached to the hopper. The shaft is attached in such a way that it could rotate about its own axis. Hence by turning the shaft, the sliding gate is opened or closed. To ease the turning of the shaft, the sprocket with a key hole is fixed to the ends of the shaft. Turning the sprocket by using the opener tool in the key hole opens the hopper (Fig. 2).

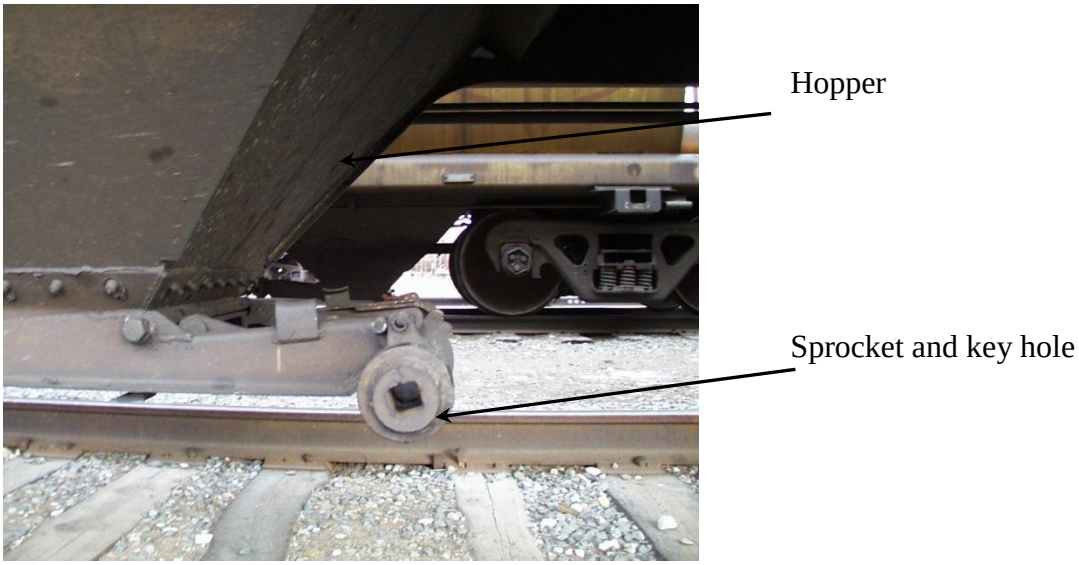


Figure 2. Hopper opening mechanism in graincars

3.2. Hopper Gate: Prototype

A prototype of the hopper assembly was constructed and mounted on a frame (Fig. 3). The frame consisted of 25.4 mm (outer diameter) pipes joined together to form the connecting lines of a rectangular parallelepiped structure. The connections were designed to allow variations in the locations of the sprocket, i.e., the connecting points were designed so that the pipes could slide vertically and collapse into a rectangular structure. The hopper was mounted on the top of two pipes, which were the longest sides of the rectangular structure (Fig. 3).

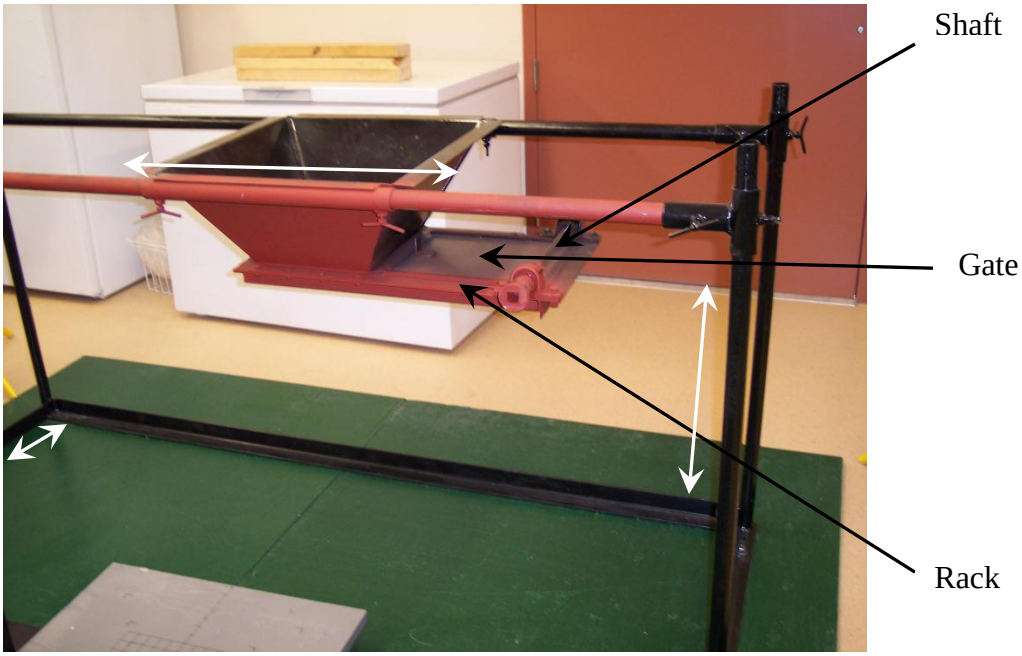


Figure 3. Prototype of the hopper and opening mechanism

The sprocket was fitted to the shaft with a play, so that it would be a loose fit to mimic the variations in sprocket orientation of graincars. The variations in orientations of the sprocket may be due to the aging of the sprockets or could possibly be damage caused by

opening by unconventional methods (e.g., opening using a crowbar). The hopper, sprocket and the hopper gate opening mechanisms were all painted in a reddish brown colour (lead oxide paint), to minimize the contrast.

3.3. Design of "Grain-o-bot" to open the Hopper Gate

A robot capable of fitting a key to the sprocket was designed. The motion of the end-effector, i.e., the key or opener tool was parallel to the axis of the shaft housing the sprocket. A cylindrical or a Cartesian robot could be used for the above motion. However stability, ruggedness and ease in operation for this application made the Cartesian robot an ideal choice. A vision system connected to a computer acted as the eye of the robot and a vision system connected adjacent to the robot acted as a sample identifier to identify the contents of the graincar.

The components of the "Grain-o-bot" are shown in Fig. 4. The three axes of the Cartesian robot namely the x, y and z axes are used to locate and fit the key to the sprocket. Locating the sprocket requires traveling along the rails of the graincar which forms one of the axes and once the sprocket is located the other two axes are used to fit the opener tool to the sprocket. Hence, positioning of the robot (adjacent to the rails) and the orientations of the three axes are fixed (Figs. 5a, 5b, 5c, 5d). Aligning the tool to the sprocket requires linear motion along the three axes. To avoid excess positional errors, the linear motion along the three axes, was made with ball screw assemblies and motors fitted to the ball screws.

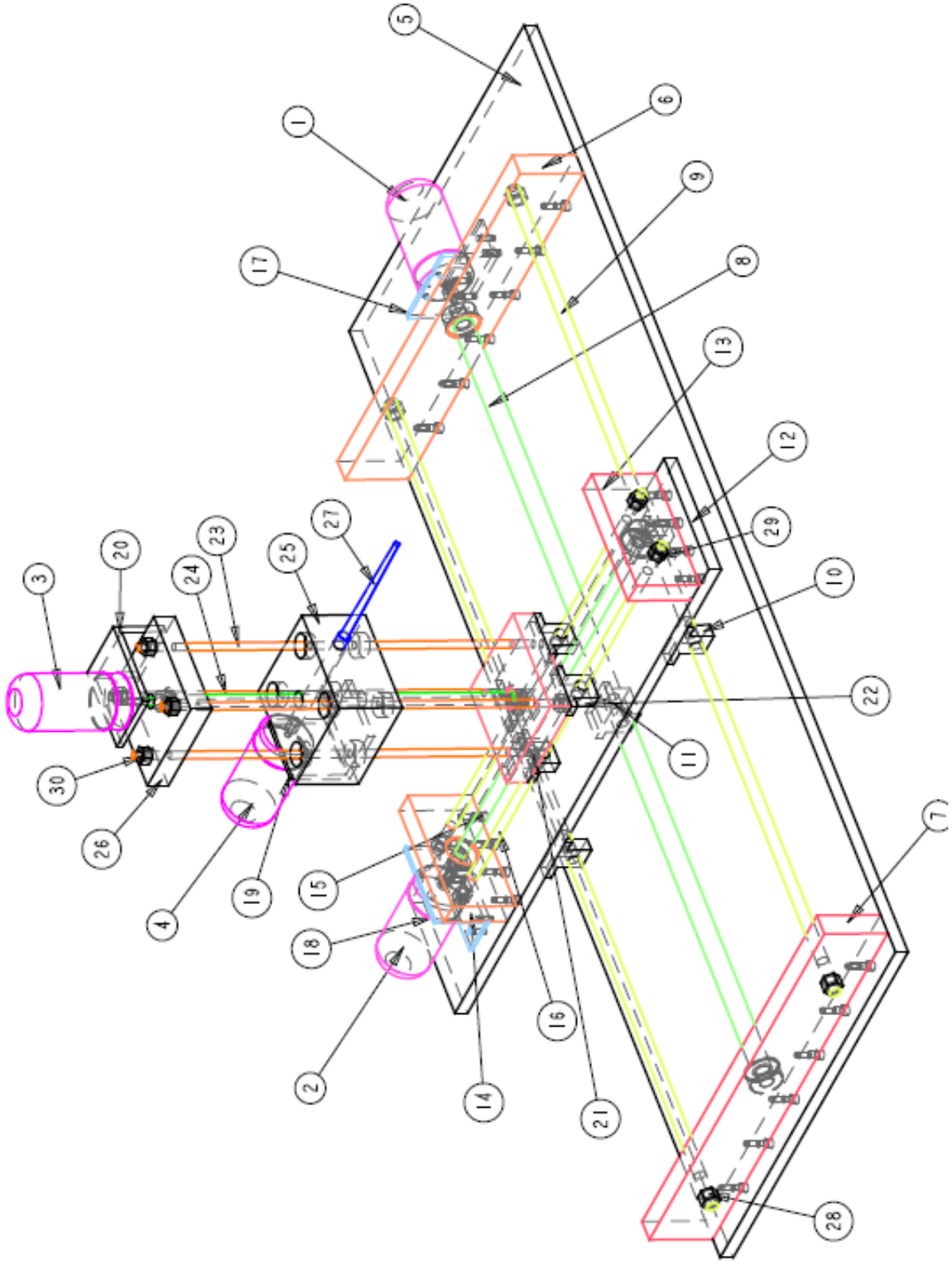
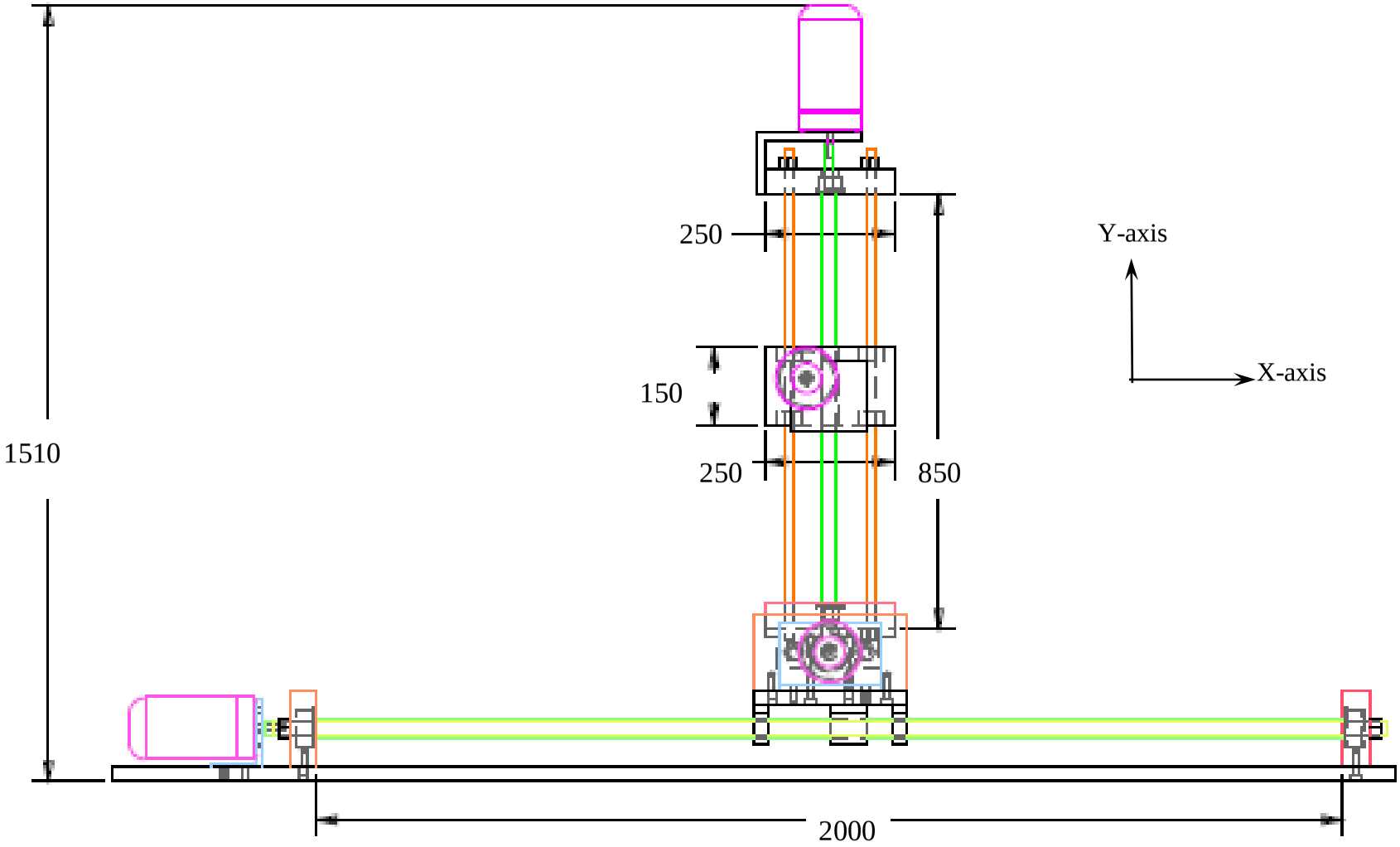
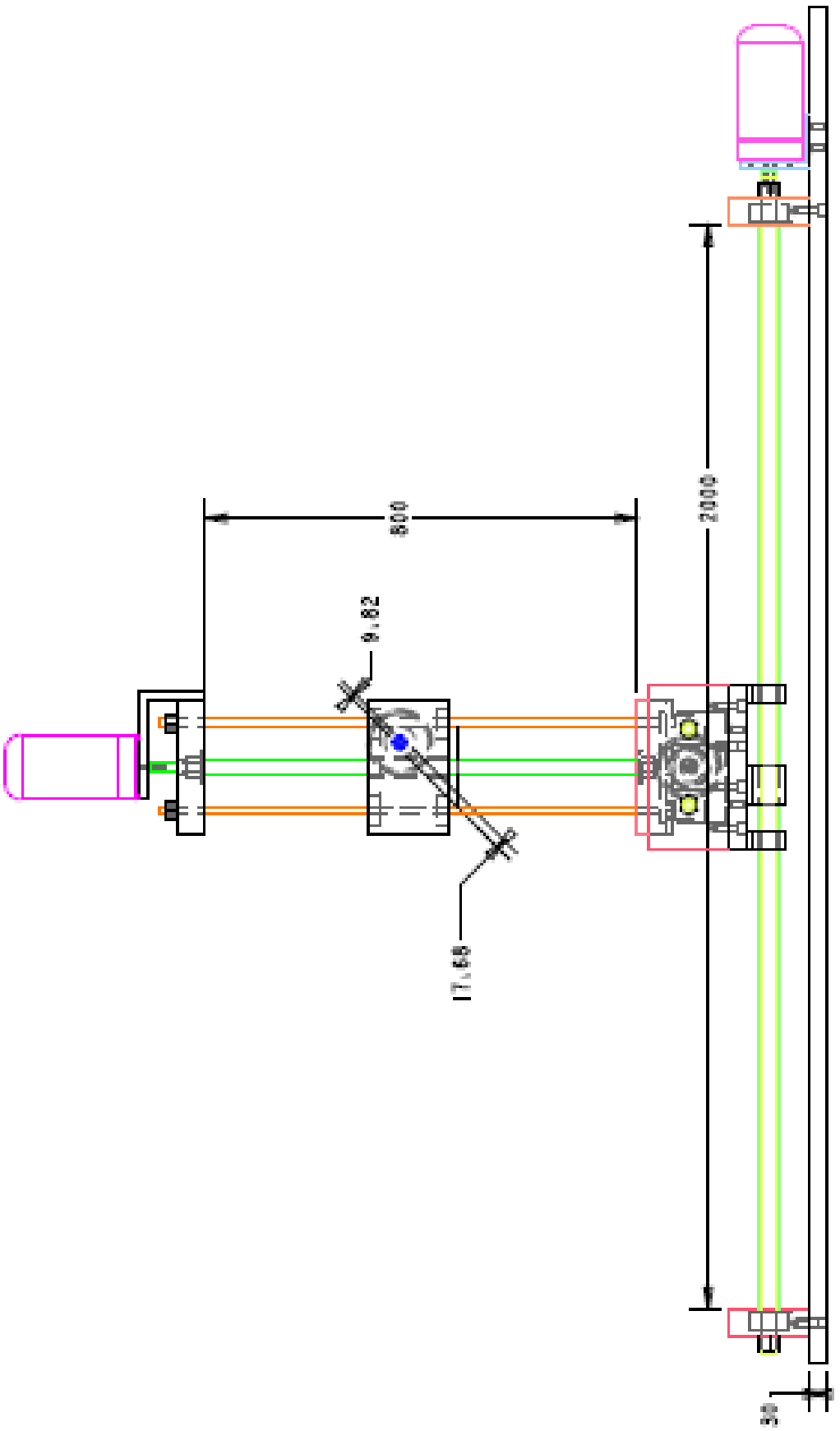


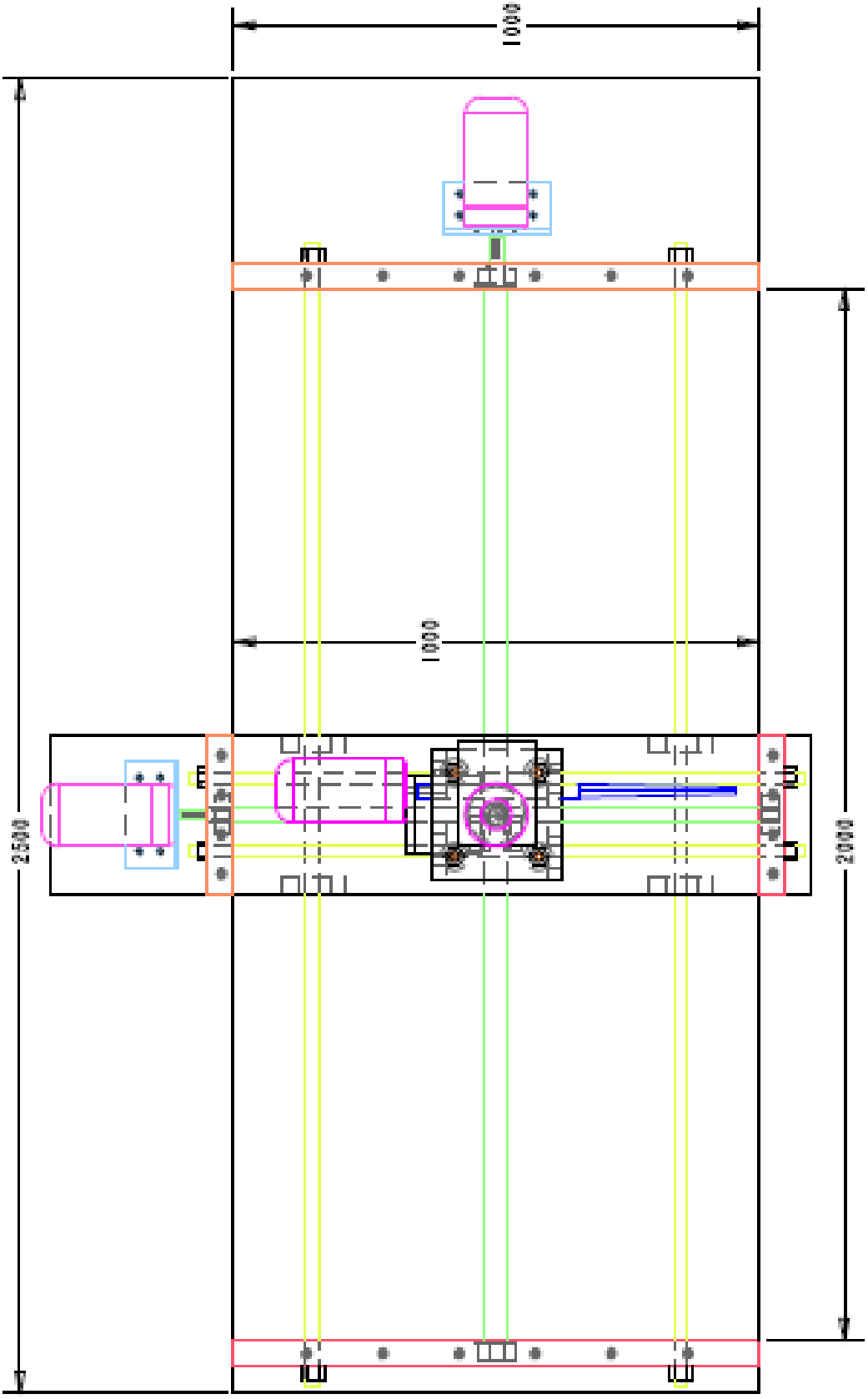
Figure 4. Orthogonal view of "Grain-o-bot" (Numbers are detailed on Page 35 and 36)



(5a)
(Dimensions are in mm)

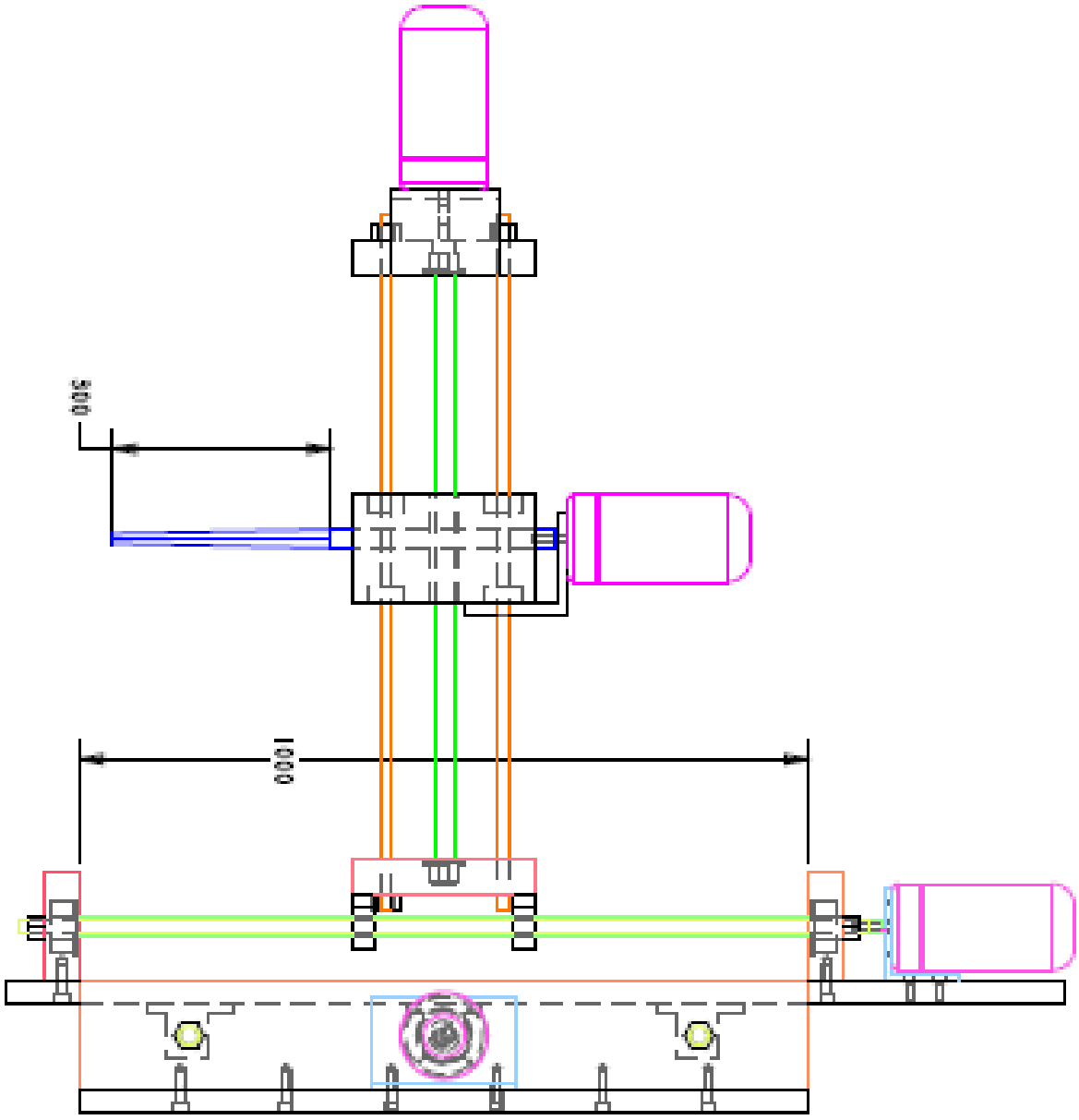


(5b)
(Dimensions are in mm)



(5c)

(Dimensions are in mm)



(5d)
(Dimensions are in mm)

Figure 5. "Grain-o-bot". a) Back View; b) Front View; c) Top View; d) Side View.

The components in the “Grain-o-bot” are as follows (Fig. 4):

1. Motor for the x-axis motion

2. Motor for the z-axis motion

3. Motor for the y-axis motion

4. Motor for the rotational motion of the key

5. Base for the x-axis assembly

6. Support block along the motor end of the x-axis

7. Support block along the non-motor end of the x-axis

8. Ball screw for the motion along x-axis

9. Support rod / feed rod for the motion along x-axis

10. Linear support bearing to slide on x-axis support rods and attach to the z-axis base

11. Nut for the x-axis ball screw

12. Base for the z-axis assembly

13. Support block along the non-motor end of the z-axis

14. Support block along the motor end of the z-axis

15. Ball screw for the motion along z-axis

16. Support rod / feed rod for the motion along z-axis

17. L-Angle bracket to support x-axis motor and connect the motor with the x-axis ball screw

18. L-Angle bracket to support z-axis motor and connect the motor with the z-axis ball screw

19. Linear support bearing to slide on y-axis support rods and attach to the tool casing

20. L-Angle bracket to support y-axis motor and connect the motor with the y-axis ball screw
21. Y-axis support block/base on the non-motor side
22. Linear support bearing to slide on z-axis support rods and attach to the y-axis base
23. Support rod / feed rod for the motion along y-axis
24. Ball screw for the motion along y-axis
25. Casing to house the sprocket key mechanism
26. Y-axis support block/base on the motor side
27. Sprocket key
28. Hex-nut to fasten the x-axis support rod
29. Hex-nut to fasten the z-axis support rod
30. Hex-nut to fasten the y-axis support

The construction of the Cartesian robot was split into four sub-assemblies to facilitate the design process, namely three assemblies to provide motion along the three axes and the fourth comprising the end-effector (the opener tool). The x-axis assembly was used to locate the sprocket and therefore formed the base. This assembly consisted of a base plate to fasten the robot in a fixed position and to provide a support for the support blocks. The ball screw was mounted on the base using two supporting blocks that were fastened to the base. The ball screw was mounted on the support blocks using bearings which were in turn mounted on the support blocks. The bearings allowed the necessary rotary motion to the ball screw while supporting the ball screw on either side. The ball screw on either end had a step machined to rest against the face of the bearings, to ensure the presence of

rotational movement and avoid unwanted lateral movements. One end of the ball screw was extended beyond the supporting structures to connect to a motor to actuate the linear motion. The ball screw and the motor were connected with a rigid L-angle flange bracket to avoid positional errors. The shaft of the motor would slide into a bore in the ball screw and the sliding motion was arrested using a key. There were two support rods or feed rods on either side of the ball screw to help support the z-axis assembly and act as a guide along the motion.

The z-axis assembly was similar to the x-axis assembly in functional construction but was located perpendicular to the x-axis assembly. This construction facilitated in moving the opener tool in and out of the sprocket hole. The base of the z-axis assembly was mounted on to the x-axis assembly, using two linear bearings present in the base of the z-axis assembly, to aid in the sliding over the feed rods in the x-axis assembly. The ball nut in the ball screw assembly mounted on the x-axis assembly was fastened to the lower portion of the z-axis assembly. Hence by actuating the ball screw in the x-axis assembly, the z-axis assembly was moved linearly along the z-axis.

The y-axis assembly was different from the x and z-axis assemblies, as the x and z-axes assemblies facilitated linear motion in the horizontal direction, the y-axis assembly facilitated linear motion in the vertical direction. This assembly comprised of two square support blocks connected by four feed rods which acted as supporting rods and the assembly formed a rectangular cube. Along the center of the supporting blocks a ball screw was connected that facilitated the motion of the end-effector, comprising the

sprocket key and the associated mechanisms. The y-axis assembly was connected to the z-axis assembly by one of the supporting blocks passing through the feed rods of the y-axis assembly, assisted by linear bearings. The ball nut in the z-axis assembly was fastened to the face of the support block of the y-axis assembly and the whole y-axis assembly moved along the z-axis by actuating the ball screw in the z-axis assembly. A motor was connected to the ball screw in the y-axis assembly on the opposite support block. Actuating this motor the end-effector assembly moved along the y-axis.

The end-effector assembly was a steel box housing the sprocket opener tool, a motor to actuate the opener tool and a coupling connecting the motor and the opening tool. The end-effector assembly moved along the support rods of the y-axis assembly on linear bearings fastened to the four corners of the steel box. The centre of the steel box was fastened to the nut of the ball screw assembly of the y-axis assembly. The end-effector assembly also comprised of a clamp attachment on one face to attach the camera in order to guide the tool along all the three axes and orient the tool.

Once the “Grain-o-bot” located the sprocket, it aligned itself and opened the sprocket partially and closed it. A small portion of the contents of the simulated graincar dropped down, which were collected and tested to identify the contents of the simulated graincar. Once the identity of the content was confirmed the robot allowed the hopper gate to fully open and let the contents flow down.

3.4. Vision The vision system that guides the "Grain-o-bot", to locate the sprocket consisted of, a Sony XCD-X700 monochrome camera (Sony of Canada Limited, Toronto, ON) and used a 1/2 type progressive scan charge coupled device sensor with square pixels. The resolution of the camera was 1024 x 768 pixels at 15 frames per second. The camera delivered uncompressed 8-bit monochrome images. It was also equipped with a 10 mm focal length enhanced resolution lens to maintain focus over a varied length ranging over 2–6 times the original focal length. The images acquired by the camera were transferred to a personal computer through a Meteor II-1394 card (Matrox Electronic Systems Ltd, Montreal, PQ) and the information in the images was processed to locate the position of the sprocket. The vision system to identify the grain was a Dalsa 2M30 camera (DALSA Corporation, Toronto, ON), with a resolution of 1600 by 1200 pixels at 30 frames per second. A Matrox Camera-Link frame grabber card (Matrox Electronic Systems Ltd, Montreal, PQ) acted as the interface between the computer and the camera. It was also equipped with a 35 mm modular lens assembly having a 0.1 x 0.1 m field of view at a working distance of 0.37 m.

3.5. Design Selection of the ball screw assembly comprising of the ball screw and the ball nut, mounted on the x-axis assembly was the most critical component as it was to linearly move the other components. The steps in selecting the ball screw are as follows (Anonymous 2004g). Assuming a weight of 100 kg, initial calculations were performed and once all the components were selected (Table.1), their actual weights were tabulated and the selection was made on the actual values.

1. The axial force required to move the load is $F = \mu \times \mathrm{N}$

F = axial force

μ = coefficient of friction of the guidance system

N = load (Weight of the end-effector assembly + y-axis assembly + z-axis assembly and was 95.11 kg)

Considering a factor of safety of 1.5

N = 95.11 * 1.5 = 142.67 kg

μ = 0.0013

F = 142.67 * 0.0013 = 0.18 kg ≈ 0.002 kN

Hence the screw must produce an axial load of 0.002 kN to move the mass of 145 kg.

2. Average velocity was calculated as, distance/time

Distance = 2 m

Total time = 3.5 min = 210 s

Average velocity = 2/210 = 0.0095 m/s.

3. Maximum velocity = 2 * Average velocity = 2 * 0.0095 = 0.019 m/s

(Considering acceleration = deceleration with a constant velocity travel)

4. Total unsupported length = 1.9 m.

5. End fixicity = Type "C" (i.e., both ends were supported by roller bearings).

6. Using the column load chart and the critical speed chart (Fig. 24, 25 in Appendix A) the selected screw was 32 x 5 (32 mm root diameter and 5 mm lead), which was the same as the selected ball screw.

Similar selections were made for the other ball screws, support rods, ball and linear bearings, and couplings.

Table 1. Major components list and specifications

(Source : Faculty of Mechanical Engineering 1978; Anonymous 2011g)

Part	Material/ Specification	Length m	Width / Diameter m	Height m	Quantity	Weight kg	Total kg
End-effector Assembly							
End-effector housing	Steel	0.20	0.20	0.20	1	2.4	2.4
Motor		0.25	0.12 Ø		1	7	7
Opener Tool	Tool steel	0.30	0.02- aver		1	1	1
Double row angular contact ball bearing	SKF 3304	0.0222	0.0200 Bore Ø	0.052	1	0.35	0.35
Self-aligning linear bearings	XLMC16	0.0261	0.016 Bore Ø	0.036	8	0.3	2.4
Vision system		0.116	0.044	0.033	1	0.5	0.5
Clamp attachment	Steel plate	0.10	0.05	0.05	1	0.05	0.05
Y-axis Assembly							
Support blocks	Aluminum	0.2	0.2	0.02	2	2.16	4.4
Support rod / feed rod	Steel	0.85	0.016		4	1.57	6.32
Ball screw assembly	SS	0.85	0.02575		1	2.8	2.8
Motor		0.25	0.12 Ø		1	7	7
Double row angular contact ball bearing	SKF 3304	0.0222	0.0200 Bore Ø	0.052	2	0.35	0.7
Self-aligning linear bearings	XLMC16	0.0261	0.016 Bore Ø	0.036	4	0.3	1.2
Z-axis Assembly							
Base	Aluminum	1	.25	0.05	1	33.75	33.75
Support blocks	Aluminum	0.25	0.05	0.1	2	3.375	6.75
Support rod / feed rod	Steel	1.01	0.016		2	1.59	3.19
Ball screw	SS	1.01	0.02575		1	3.6	3.6

assembly							
Motor	1	0.25	0.12 Ø		1	7	7
Double row angular contact ball bearing	SKF 3304	0.0222	0.0200 Bore Ø	0.052	2	0.35	0.7
Self-aligning linear bearings	XLMC25	0.058	0.025 Bore Ø	0.0401	4	0.5	2
X-axis Assembly							
Base	Aluminum	2.2	1	0.03	1	81	81
Support blocks	Aluminum	1	0.05	0.15	2	20.25	40.5
Support rod / feed rod	Steel	2.3	0.025		2	2.5	5
Ball screw assembly	SS	2.3	0.0421 Ø		1	9.5	19
Motor	1	0.25	0.12 Ø		1	7	7
Double row angular contact ball bearing	SKF 3306	0.0720	0.0300 Bore Ø	0.0302	2	0.35	0.7

3.6. Electrical Drive Four, SIGMA 20-4266TS-22770 (Yaskawa Motoman Canada, Pointe-Claire, PQ) slow synchronized AC motors, actuated the robot. Three of the motors facilitated the control of the linear motions along x, y, and z directions and the fourth motor controlled turning of the opening tool in the end-effector assembly. The motors were rated as 72 rpm, 60 Hz, 120 VAC, 0.52 kg.m torque, capable of achieving the full rpm rating within 3º of rotation and a complete stop within 3º of rotation. The motors were controlled by an Omega PCI-DIO24 card (Omega Engineering Inc, Laval, PQ) and, a control box (to control the starting and stopping of the motor and direction of rotation of the motor). The control box sensed the signals from the interface card and powered the motors.

3.7. Lighting Proper illumination is essential for acquiring images because no image processing technique could process information that is not captured (Paulsen and McClure 1986). Hence the illumination by four different types of lights was tested namely: a fluorescent ring light, two incandescent light systems, and an AI RL36120 red LED (light emitting device) ring light. The fluorescent ring light was a ring bulb of 32 W fixed on a wooden base and was housed in a concave container painted inside with white paint to focus all reflected light in one direction. The first incandescent light system consisted of eight bulbs, 100 W each, arranged in a circular fashion in a circular drum. It was designed to provide diffuse light using shades. The second system consisted of eight bulbs, 46 W each, arranged in a circular fashion as the first system. The beam was directed using a semicircular stainless container painted white using magnesium oxide. The fluctuations in the systems were controlled using a photodiode light sensor. The LED ring light was red ring light connected to a strobe controller and was capable of producing 710 Lux at a working distance of 0.1 m and was used to increase contrast on edges. The above sources with the exception of the LED were connected to a voltage stabilizer. The lights were tested with the variations (Table 2), and arranged as shown in Fig. 6, to mimic the ambient lighting condition.

A Kodak white card was used as a reference to calibrate the camera. The image of the white card was acquired and five areas, one from each corner and one from the centre containing 100 X 100 pixels were tested and illumination within ± 1 grey value on the viewing area was obtained.

Table 2. Lighting Experiments for the Variations

Light sources	**Variations**
1. Fluorescent ring light 2. Incandescent light a. Direct b. Diffuse 3. LED ring light	1. Human intervention: Presence or absence 2. Position of light source: At a distance of 1 m with respect to the sprocket: Straight, top left and top right, bottom left and bottom right. 3. Stray Light: 1. Light from back right of sprocket; 2. Light from back left of sprocket; 3. Light from both back right and back left of sprocket; 4. Light from front right of sprocket; 5. Light from front left of sprocket; 6. Light from both front right and front left of sprocket; 7. Light from the ceiling 4. Background variation: 1. White Sheet; 2. Galvanized Steel Sheet; 3. Lead-Oxide; 4. Stainless Steel Sheet;

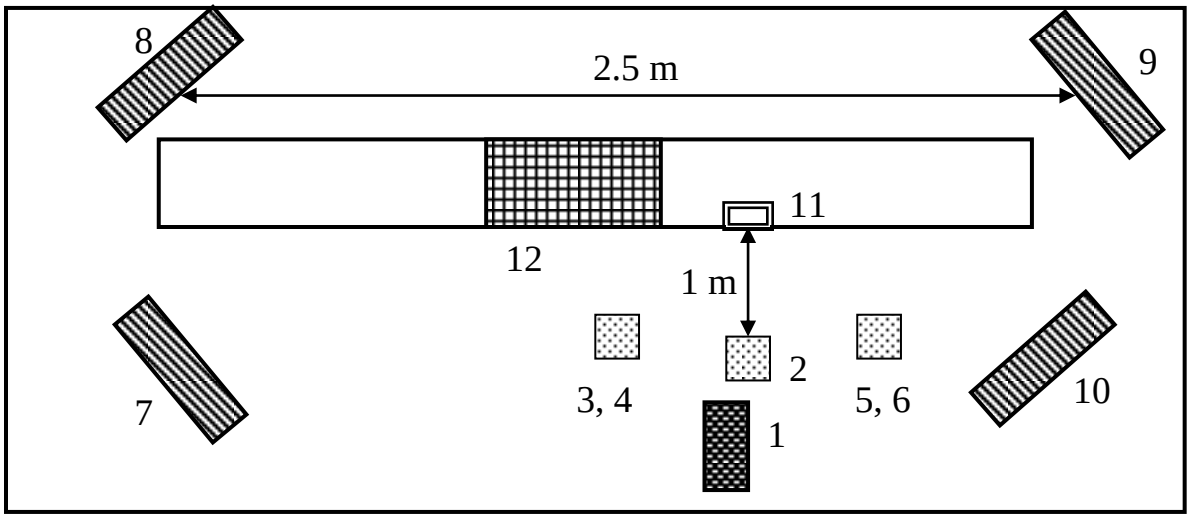


Figure 6. **A Schematic showing different location of the light source and stray lights: 1 = camera; 2 = position of light source in front of the sprocket; 3 and 4 = alternate positions of light source to the top and bottom right of the sprocket; 5 and 6 = alternate positions of light source to the top and bottom left of the sprocket, 7 through 10 = positions of stray light source from the front right of the sprocket, back right of the sprocket, back left of the sprocket, and front left of the sprocket, respectively; 11 = sprocket; and 12 = hopper.**

A total of 1440 images were collected with all combinations and tested.

3.8. Sprocket Identification

Algorithms were developed in MATLAB 7.0 (The MathWorks, Inc., Natick, MA) on a windows platform and were used to determine the presence of the sprocket in a given image. Initially a template was chosen by visual inspection from an image, which had good contrast, and the region encompassing the sprocket was selected. The template image was 101 X 101 pixels. Some of the images acquired with backlight or excess light reflected from the background contained bright patches and stray light. The grey levels of the bright patches were over 240 and were identified and eliminated by pre-processing the regions with an averaging filter (Gonzales and Woods 1998). The edges of objects in both the target and the template images were detected using the Canny operator and the

target images were further analysed using three types of algorithms. The three types of algorithms developed to identify the sprocket are:

1. Algorithm I: Correlation
2. Algorithm II: Correlation followed by regional correlation on sub images
3. Algorithm III: Correlation followed by shape detection on sub images

Algorithm I

The edge detected template image was correlated with the edge detected target image by scanning the template over the target image (Gonzalez and Woods 1998). The highest pixel value of the correlated image represented the existence of the template in the target image. Pixels with values ranging between the highest pixel value and the highest pixel value minus 10 from the correlated images were recorded as the sprocket locations. However, the highest value was also obtained when the edges of the template were correlated with a bright patch in the target image. Hence, additional processing was required to confirm the existence of the sprocket.

Algorithm II

The images were first analysed using algorithm I. Sub images of 100 X 100 pixels around the pixels identified by algorithm were separated from the original image. Edge detection was performed on the sub images. A second template containing only the sprocket was correlated with the sub images. The highest pixel value in each correlated sub image was compared and the sub image with the highest correlated pixel value was selected as the image containing the sprocket.

Algorithm III

Though the accuracy of the sprocket identification by algorithm II was higher than algorithm I, there were many false positives (objects other than the sprocket were identified as the sprocket). To prevent the detection of false positives, the process of identification by using the second correlation was replaced by a shape detection function. The shape detection function was used to determine the presence or absence of both the ellipse and the square in the sub images.

The top 20 points were selected by performing the first correlation and were treated as the possible locations of the sprocket. The area encompassing the selected pixels were separated from the original image and the recorded as individual sub image (101 X 101 pixels). Edge detection was performed on the sub images using a canny operator. Individual objects within the sub images were identified and labelled using the eight-connectivity function. Following labelling, objects with less than 10 pixels were eliminated. Objects within 35 pixels from the centre of the sub-images were tested for the sprocket hole, which was a square, and objects that were 40 pixels from the boundary were tested for an ellipse. To test the presence of the square, a straight-line equation was fitted for adjacent pixel locations. If a change in slope was observed for more than four pixel pairs consecutively, then it was said to represent a corner and the location was noted. By obtaining three such locations, a square was constructed. The location of the pixels from the object was compared to the connected square. If the deviation of the pixels in the object and the constructed square was within 3 pixels, the identified object was considered to be the square. The location of the centre of the square was recorded.

The objects within 40 pixels from the edges were tested by constructing an ellipse with the locations of the pixels in the object. The object was divided into three sections. The pixel locations connecting the sections were recorded as four locations to fit the ellipse using the determinant method and the centroid of the ellipse was recorded. If, the centroid of the fitted ellipse, and the centroid of the square, were within 3 pixels of each other, then, it was determined that the sprocket was identified in the sub image. Otherwise the subsequent sub images were tested.

3.9. Training and Operation of the "Grain-o-bot"

The process of operating the robot included a training process where images were acquired in the following order to act as templates.

1. When the tool was properly fit inside the sprocket
2. The instance, the tool was initially disengaged from the sprocket and moved linearly along the z-axis towards the 0^{th} position in the z-axis.
3. When the tool was disengaged from the sprocket and moved linearly along the z-axis to the 0^{th} point in the z-axis.
4. When the tool was moved linearly along the y-axis to the 0^{th} point in the y-axis.
5. The first instance when the sprocket was fully available inside the frame when moving along the x-axis, from the 0^{th} position of the x-axis.

Once these frames were recorded, the robot was initially allowed to move along the x-axis from the 0^{th} position. Initial investigations showed, the frame captured along the x-axis at a distance of 1.5 m, encompassed a length of 0.6 m. So images were recorded every 0.3 m to capture the sprocket fully in the first frame. Once the sprocket was identified, the location of the sprocket with respect to the x-y plane was determined with

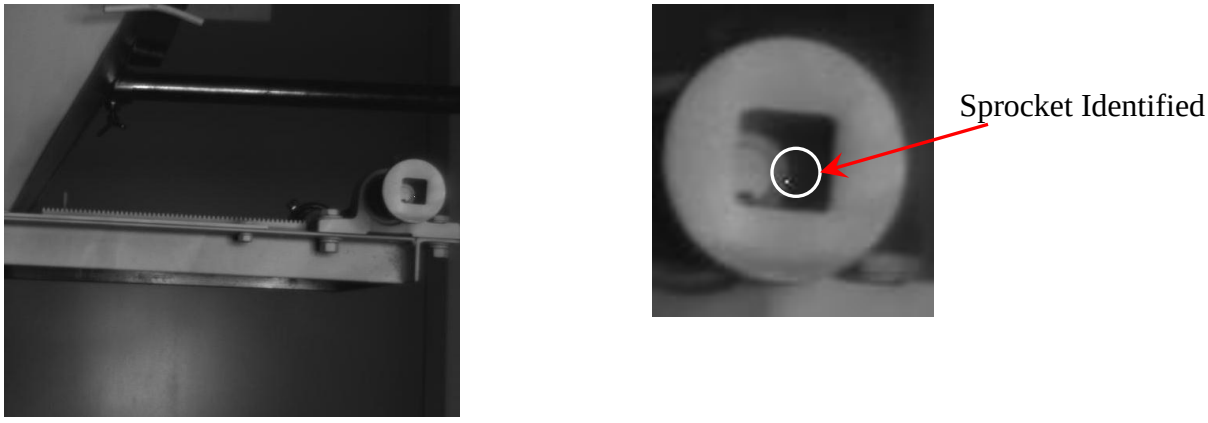
template-5. The motion of the robot along the x-axis was fine tuned to bring the sprocket to the exact location as in the template-5. Since the other positions were already known, using the time taken for moving from one position to another and using templates 2, 3, and 4 as references the tool was moved until it reached its position. Here the orientation of the tool and the sprocket was aligned by fine tuning the tool to the acquired image with template 5 and the fit was made. The tool was now turned in a clockwise direction for 3 s and immediately in an anticlockwise direction to open and shut the hopper gate. This resulted in a sample of the material in the hopper to fall down. This was collected in an inclined trough, which was transported, to a plate for identification. Once the identification was made and the material was found to be the required material (matching information on the I-90 tag) then the robot opened the hopper gate to allow material to flow.

3.10. Grain Identification Previous studies have indicated the classification of bulk grain with very high classification accuracy (Cogdill et al. 2004; Choudhary et al. 2008, 2009; Karunakaran 2002; Karunakaran et al. 2004; Maghirang et al. 2003; Mahesh et al. 2008; Majumdar and Jayas 2000 a, b, c, d; Manickavasagan et al. 2008; Paliwal et al. 2004; Visen 2002). The images, features and network from these studies was integrated and utilized.

4. RESULTS AND DISCUSSION ON SPROCKET IDENTIFICATION

4.1. Sprocket Identification

Different types of images acquired using different lights and variables were tested for the identification of the sprocket location as shown in Fig. 6. The original and the edge detected images are shown in Fig. 7. Direct lighting sometimes produced reflections from nearby objects. The intensity of the light beams sometimes smoothed information, which would have been helpful in the identification. The images analyzed by the algorithms were visually inspected to confirm the correct identification. The results obtained by applying the three algorithms and the results confirmed visually were tabulated. The number of instances where the manual verification reported a positive identification was referred to here as the reported identification.



(a) (b)

Figure 7. Sprocket identification. a) Image with illumination from incandescent light source; b) Sprocket location correctly identified

4.1.1. Sprocket identification using Algorithm I

The influences of the light positions, namely, light source at top right with respect to the sprocket, light source at the top left with respect to sprocket, light source at the bottom

right with respect to sprocket, and light source at the bottom left with respect to sprocket and the light source centred and directly facing the sprocket were analyzed. Using Algorithm I and testing for sprocket identification provided 100% accurate results with all four lighting types (Table 3a). When examined manually there were instances where there was more than one point identified as the sprocket. The presence of false positives had increased the efficiency of identification.

Table 3a. Influence of light position on sprocket classification accuracy using Algorithm I.

Light Type	Light Position				
	1	2	3	4	5
Incandescent-direct	100%	100%	100%	100%	100%
Incandescent-diffuse	100%	100%	100%	100%	100%
Fluorescent	100%	100%	100%	100%	100%
LED	100%	100%	100%	100%	100%

* Twenty replicates were tested in all cases

1. Light source at top right with respect to sprocket; 2. Light source at top left with respect to sprocket; 3. Light source at bottom right with respect to sprocket; 4. Light source at bottom left with respect to sprocket; 5. Light source centred at the sprocket.

The influences of varied background (behind the sprocket) namely, white, lead-oxide, galvanised steel and stainless steel, on sprocket identification were tested using Algorithm I and recorded (Table 3b). The incandescent-direct light source proved to be a better choice than the other three light systems and LED proved to be the worst choice. All instances where there were more than one point were identified.

Table 3b. Influence of varied background on sprocket classification accuracy using Algorithm I.

Light Type	Background			
	White	Lead-Oxide	Galvanized Steel	Stainless Steel (Star Pattern)
Incandescent-direct	100%	100%	100%	95%
Incandescent-diffuse	90%	100%	95%	100%
Fluorescent	100%	100%	100%	70%
LED	70%	100%	95%	90%

- Twenty replicates were tested in all cases

The influence of personnel present in the background (behind the sprocket) was tested as a variable for the classification of the sprocket and the classification accuracies were recorded (Table 3c). The incandescent-direct light source proved to be a better light source between the tested sources and fluorescent light source performed badly.

Table 3c. Influence of human presence in the background on sprocket classification accuracy using Algorithm I.

Light Type	Person in Background	
	Present	Absent
Incandescent-Direct	90%	100%
Incandescent-Diffuse	55%	100%
Fluorescent	20%	100%
LED	35%	100%

* Twenty replicates were tested in all cases

The influence of stray light was tested as a variable and the classification accuracy of the sprocket was recorded (Table 3d). Identification of the sprocket when the stray light was

directed towards the image from the front (along the direction of the light coming from the lighting systems), adding to the original lighting systems was the cause of the low identification accuracies. Identification of the sprocket using the LED was better in the case of stray light compared to the other lighting systems with a slight dip in the accuracy of identification when the stray light came from the front left and both front right and front left of the sprocket.

Table 3d. Influence of stray light on sprocket classification accuracy using Algorithm I.

Light Type	Light Variations						
	1	2	3	4	5	6	7
Incandescent-Direct	100%	100%	100%	100%	0%	0%	100%
Incandescent-Diffuse	100%	100%	85%	100%	100%	0%	80%
Fluorescent	20%	85%	90%	100%	100%	95%	100%
LED	100%	100%	100%	100%	75%	50%	100%

* Twenty replicates were tested in all cases

1. Stray light from back right of sprocket; 2. Stray light from back left of sprocket;
2. Stray light from both back right and back left of sprocket; 4. Stray light from front right of sprocket; 5. Stray light from front left of sprocket; 6. Stray light from both front right and front left of sprocket; 7. Stray light from the ceiling

This probably happened because the stray lighting systems with their higher luminance flushed the information in the image.

4.1.2. Sprocket identification using Algorithm II

To improve on the identification of the sprocket and to eliminate the false positives, Algorithm II was used on the images. The accuracy of LED light source dropped down when the light was held at the top right corner with respect to the sprocket (Table 3e). However there were a few false positives, indicating the sprocket location to be at the correct position and also at a different location.

Table 3e. Influence of light position on sprocket classification accuracy using Algorithm II.

Light Type	Light Position				
	1	2	3	4	5
Incandescent-Direct	100%	100%	100%	100%	100%
Incandescent-Diffuse	100%	100%	100%	100%	100%
Fluorescent	100%	100%	100%	100%	100%
LED	30%	100%	100%	100%	100%

* Twenty replicates were tested in all cases
1. Light source at top right with respect to sprocket; 2. Light source at top left with respect to sprocket; 3. Light source at bottom right with respect to sprocket; 4. Light source at bottom left with respect to sprocket; 5. Light source centred at the sprocket.

Applying Algorithm II to the set of images with varied backgrounds indicated that the incandescent-direct light source and the fluorescent light source had a sprocket identification accuracy of 100% in all the variations provided by the different backgrounds, but the LED light source performed poorly when used with a white background (Table 3f). This could be attributed to the fact that the LED was red and when the light impinged on the white background flushed the sprocket information from the resulting image as the sprocket was also red.

Table 3f. Influence of varied background on sprocket classification accuracy using Algorithm II

Light Type	Background			
	White	Lead-Oxide	Galvanized Steel	Stainless Steel (Star Pattern)
Incandescent-Direct	100%	100%	100%	100%
Incandescent-Diffuse	95%	100%	90%	100%
Fluorescent	100%	100%	100%	100%
LED	65%	100%	100%	100%

* Twenty replicates were tested in all cases

The images with the presence and absence of personnel in the background of the sprocket were tested with Algorithm II. The images from the diffuse and LED light sources when processed with Algorithm II, dropped in the identification accuracy to 50% and the fluorescent light source fared poorly (25% accuracy) when a person was present in the background (Table 3g). Here again a few false positives were observed, which would hamper in the decision of the exact location of the sprocket.

Table 3g. Influence of human presence in the background on sprocket classification accuracy using Algorithm II

Light Type	Person in Background	
	Present	Absent
Incandescent-Direct	100%	100%
Incandescent-Diffuse	50%	100%
Fluorescent	25%	100%
LED	50%	100%

* Twenty replicates were tested in all cases

When Algorithm II was tested for identification of the sprocket, on images acquired with stray light, identification of the sprocket when the stray light entered from both directions in the front namely right and the left, the identification capacity dropped drastically compared to the other variations, similar to the performance of Algorithm I. Images from all the light sources on an average performed similarly in the identification of the sprocket in the other conditions (Table 3h). Here again false positives were observed, when the resulting images were manually verified.

Table 3h. Influence of stray light on sprocket classification accuracy using Algorithm II.

Light Type	Light Variations						
	1	2	3	4	5	6	7
Incandescent-Direct	100%	100%	100%	90%	80%	0%	100%
Incandescent-Diffuse	100%	100%	100%	95%	80%	0%	100%
Fluorescent	100%	95%	100%	100%	100%	0%	100%
LED	100%	100%	100%	80%	100%	5%	100%

* Twenty replicates were tested in all cases

1. Stray light from back right of sprocket; 2. Stray light from back left of sprocket; 3. Stray light from both back right and back left of sprocket; 4. Stray light from front right of sprocket; 5. Stray light from front left of sprocket; 6. Stray light from both front right and front left of sprocket; 7. Stray light from the ceiling

4.1.3. Sprocket identification using Algorithm III

To further enhance the identification of the sprocket, Algorithm III was developed and tested. Identification accuracies on the image where the light sources were placed in five locations in front of the sprocket were recorded (Table 3i). The identification had dropped down from the 100% identification in Algorithm I and II to a range from 0 –

95%. On an average the fluorescent and the incandescent-diffuse light source performed better than the LED and the incandescent-direct light source (Table 3i). This was due to the use of shape detection techniques, which failed to identify the sprocket when the minimum conditions were not met. However in the images where the sprocket was identified, there were no false positives.

Table 3i. Influence of light position on sprocket classification accuracy using Algorithm III.

Light Type	Light Position				
	1	2	3	4	5
Incandescent-direct	15%	45%	0%	45%	50%
Incandescent-diffuse	30%	60%	5%	80%	50%
Fluorescent	10%	95%	25%	85%	75%
LED	0%	75%	0%	65%	45%

* Twenty replicates were tested in all cases

1. Light source at top right with respect to sprocket; 2. Light source at top left with respect to sprocket; 3. Light source at bottom right with respect to sprocket; 4. Light source at bottom left with respect to sprocket; 5. Light source centred at the sprocket.

Algorithm III was applied on images acquired with varied backgrounds and the results are tabulated in Table 3j. The results indicate that stainless steel and galvanized steel had low effect in influencing the identification of the sprocket, whereas the white and lead-oxide backgrounds had an effect on the details of the image and thereby hampering the identification process.

Table 3j. Influence of varied background on sprocket classification accuracy using Algorithm III.

Light Type	Background			
	White	Lead-Oxide	Galvanized Steel	Stainless Steel (Star Pattern)
Incandescent-Direct	90%	50%	70%	70%
Incandescent-Diffuse	70%	50%	85%	80%
Fluorescent	0%	75%	55%	80%
LED	60%	45%	100%	90%

* Twenty replicates were tested in all cases

On an average all the light sources performed equally in capturing the details of the sprocket to assist in identification of the sprocket in the images.

Images with and without personnel in the background were processed with Algorithm III and the results indicated that images with personnel in the background were better for the identification of the sprocket with all light sources (Table 3k). This could be attributed to the fact that the plain background merged with the features of the sprocket and when a person was introduced into the frame, the features of the sprocket were better highlighted because of the non-uniformity of the persons' image in the background. LED light source performed better than the other light sources in highlighting the features of the sprocket for identification but performed poorly when the personnel was absent in the background.

Table 3k. Influence of human presence in the background on sprocket classification accuracy using Algorithm III.

Light Type	Person in Background	
	Present	Absent
Incandescent-Direct	80%	50%
Incandescent-Diffuse	75%	50%
Fluorescent	80%	75%
LED	95%	45%

* Twenty replicates were tested in all cases

The fluorescent light source fell to second position for identifying the sprocket in both instances where a person was present or absent.

Testing the influence of stray light on the capability of Algorithm III to identify the sprocket showed results, which were not very promising (Table 3l).

Table 3l. Influence of stray light on sprocket classification accuracy using Algorithm III.

Light Type	Light Variations						
	1	2	3	4	5	6	7
Incandescent-Direct	50%	0%	0%	0%	0%	5%	75%
Incandescent-Diffuse	35%	100%	55%	0%	0%	0%	70%
Fluorescent	10%	20%	0%	0%	0%	0%	0%
LED	75%	25%	45%	0%	0%	50%	10%

* Twenty replicates were tested in all cases

1. Stray light from back right of sprocket; 2. Stray light from back left of sprocket; 3. Stray light from both back right and back left of sprocket; 4. Stray light from front right of sprocket; 5. Stray light from front left of sprocket; 6. Stray light from both front right and front left of sprocket; 7. Light from the ceiling

Further investigation needs to be carried out to minimize the influences of stray light. Since Algorithm III used shape detection as its base for processing the images and identifying the sprocket, the images where stray light was introduced flushed the features and limited the capacity of the algorithm in the sprocket identification.

No identification was made when the stray light was introduced from the front right and from the front left of the sprocket. However when the stray light was introduced from both sides the LED light source was able to pick up features by cancelling out the interferences and making identification in 50% of the instances.

5. FABRICATION OF THE "GRAIN-O-BOT"

The fabrication of the "Grain-o-bot" was completed as per the description in Chapter 3 with a few modifications.

5.1. Fabrication of the "Grain-o-bot" structure

The frames for the x-axis and z-axis were made of 6 gauge square steel tubing with guide rails mounted on the top face to assist motion of the other assemblies as shown in Fig. 8. Plummer blocks were used on the ends to support the ball screw assemblies.

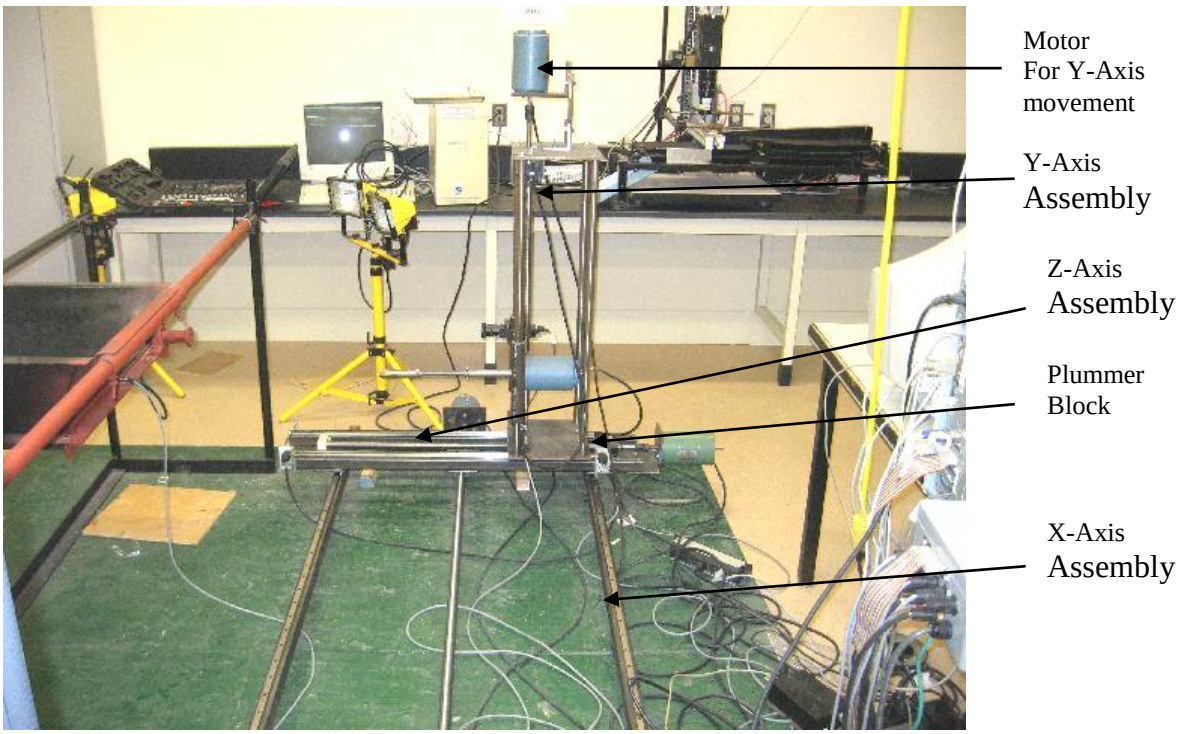
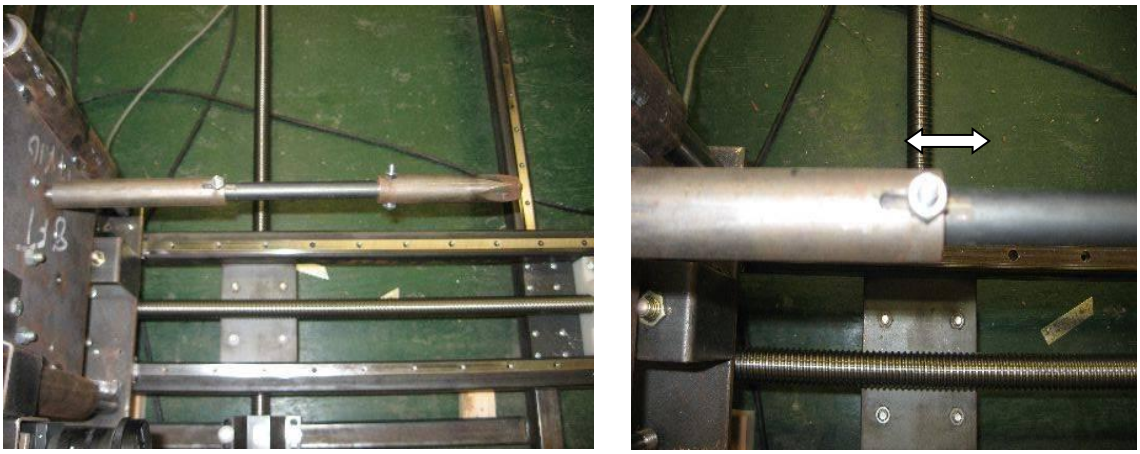


Figure 8. Structure of "Grain-o-bot"

The end-effector was changed from a one piece tool as described in the design conceptualization (Chapter 3) to a four piece tool, primarily consisting of

1. a tapered part to mate with the sprocket on one end and cylindrical on the other end,
2. a cylindrical shaft with bores on either ends,
3. a connector shaft, and
4. a spring.

This change was made so that the tapered tool can move linearly within the shaft by 12 mm when a pressure was applied at the end of the tool where the tool was not properly fitted in the sprocket hole. The extended position was held by the force of the spring located between them.



(a) (b)

Figure 9. End-effector. a) Image showing the full end-effector; b) End-effector retracting mechanism

The setup without the linear movement of the tool could accommodate the mating of the sprocket with the tapered tool in all instances with the exception when the corner of the square in the tapered tool mated with the face of the square in the sprocket. In this instance if the end-effector was actuated, would produce a hammering effect both on the

sprocket and the "Grain-o-bot". To eliminate the hammering effect the tool was thus modified as shown in Fig. 9.

5.2. Limit Switch design and assembly

The motors were connected with clamps designed to specifically position the motors in line with the ball screw rods for motion along the axes and the end-effector.

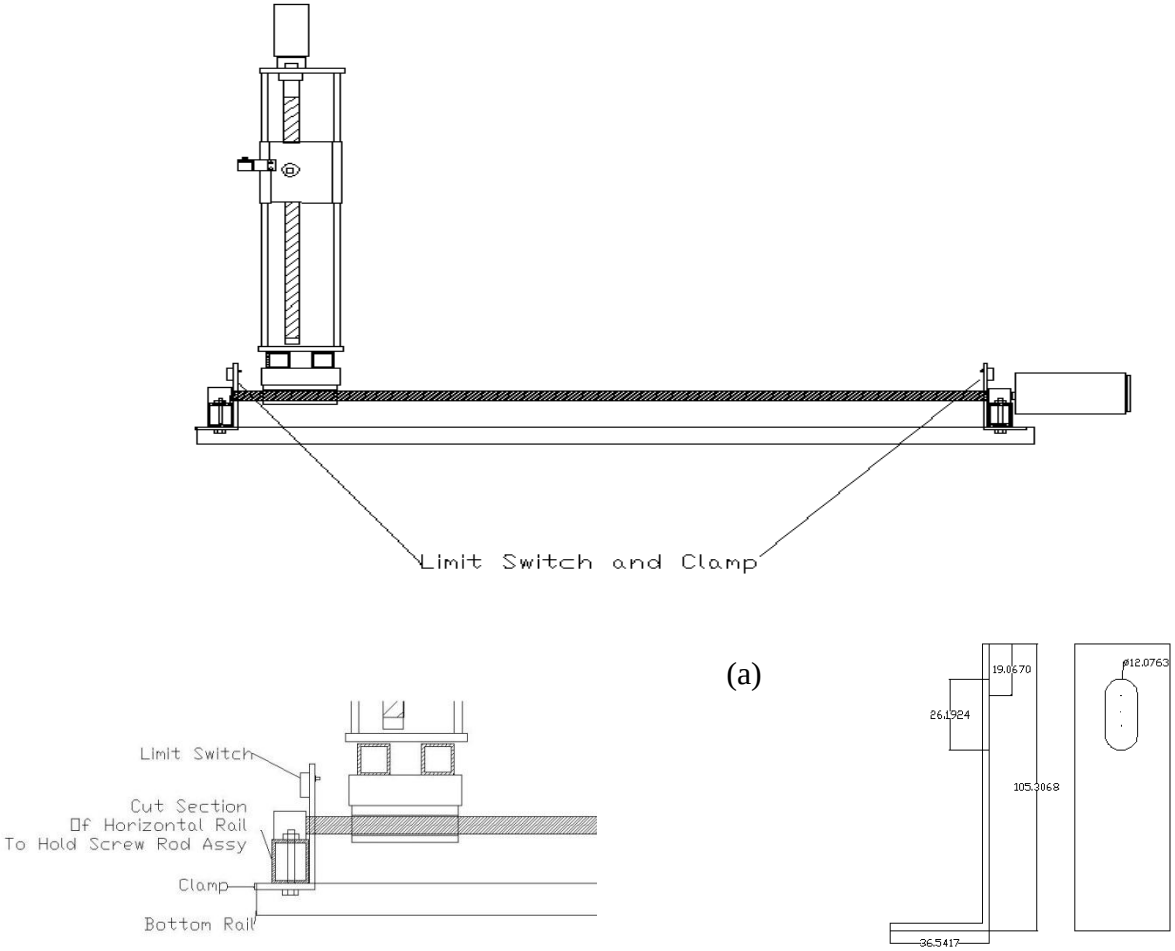
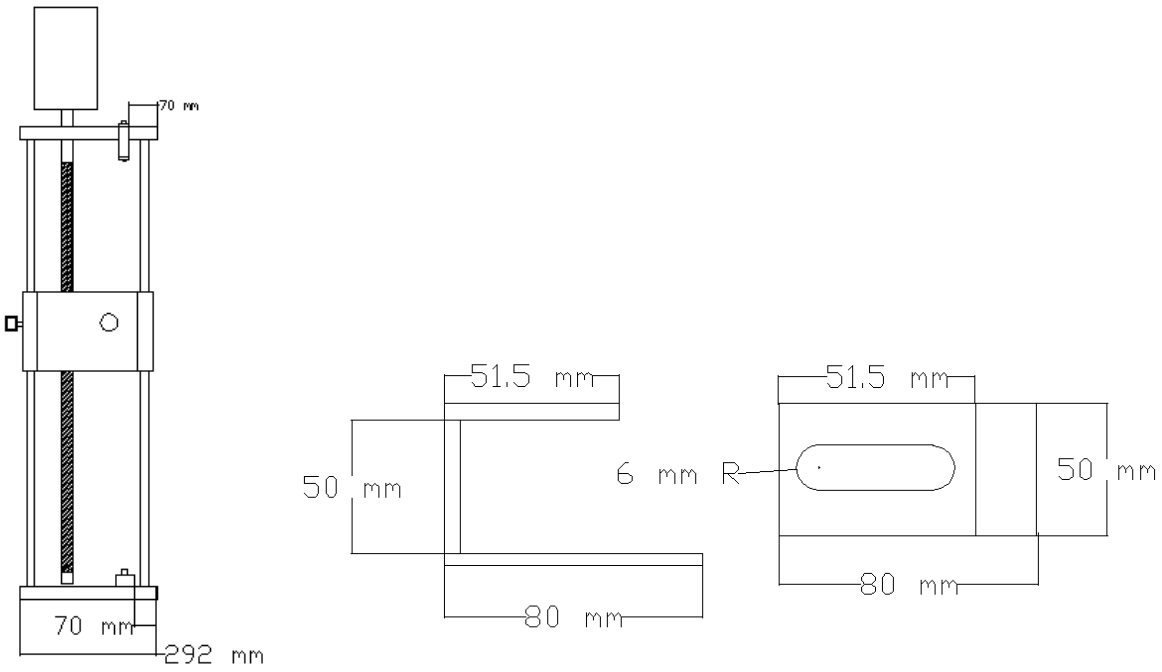


Figure 10. X-axis limit switch design and assembly. a) Image showing limit switch locations in the x-axis assembly; b) Image showing detailed view of limit switch location; c) clamp for limit switch assembly (Dimensions are in mm)

A second set of clamps were designed and mounted on the frame to position 8 extended straight plunger limit switches capable of withstanding an operating force of 0.0034 kN, return force of 0.001 kN, and with a pre-travel of 0.4 mm, an operating travel of 5.5 mm, movement differential of 0.05 mm and an operating position at 21.5 mm to control motion.

The limit switches to control the x-axis movement were mounted on a clamp designed as shown in Fig. 10 (b) using a seven gauge metal sheet and assembled on the shorter side of the x-axis assembly frame as shown in Fig. 10 (a) and Fig. 10 (c). The inclusion of the limit switches controlled the range of motion along the x-axis to not travel beyond 1950 mm and assisted in powering off the motors when the switches were activated.

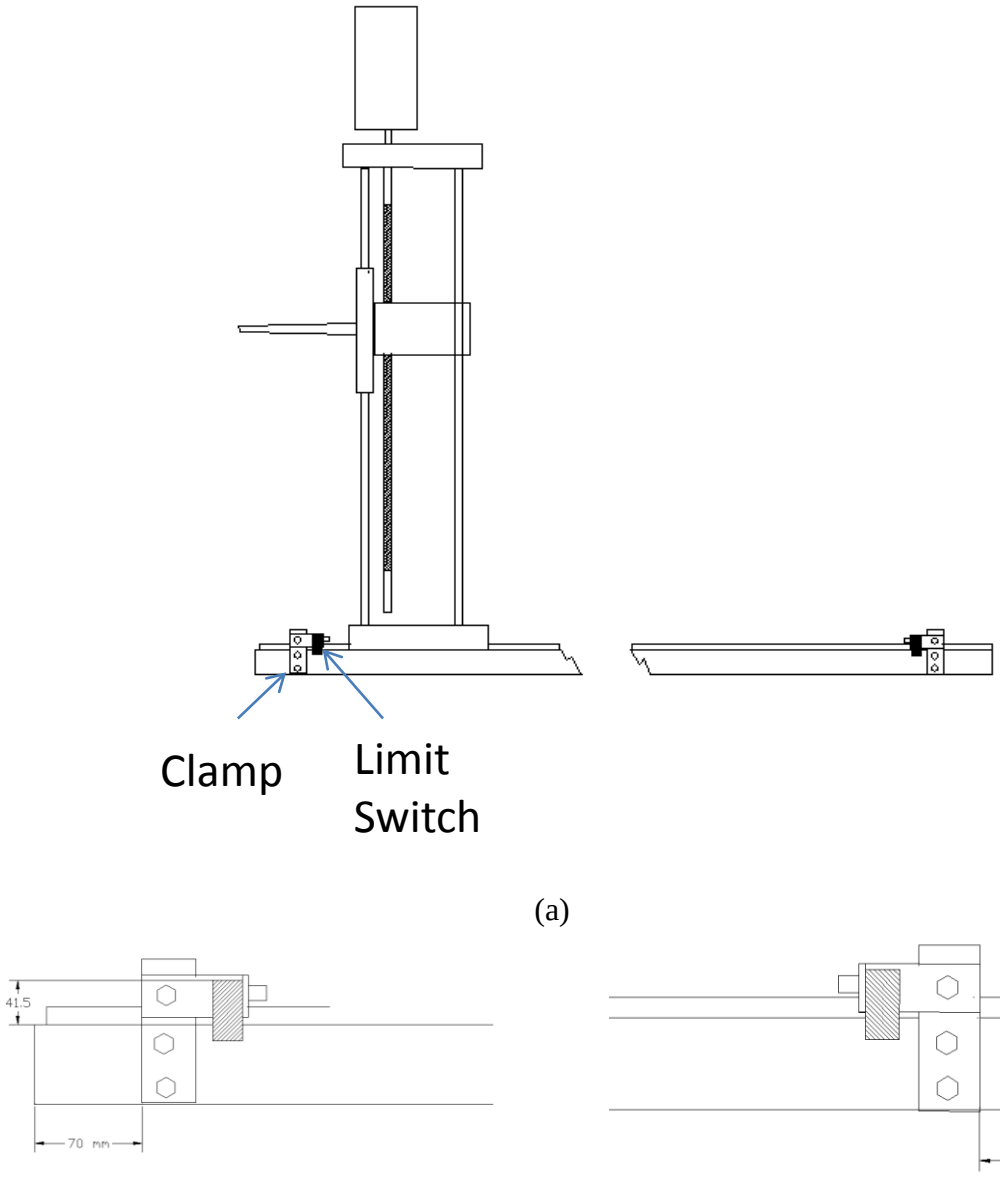


(a)

(b)

Figure 11. Y-axis limit switch design and assembly. a) Image showing limit switch locations in the y-axis assembly; b) clamps for limit switch assembly in y-axis

A similar set up was designed, fabricated and assembled for the y-axis to control motion up to 900 mm and for the z-axis to control motion up to 900 mm and the hopper to control motion up to 275 mm as shown in Fig. 11, 12, and 13, respectively.

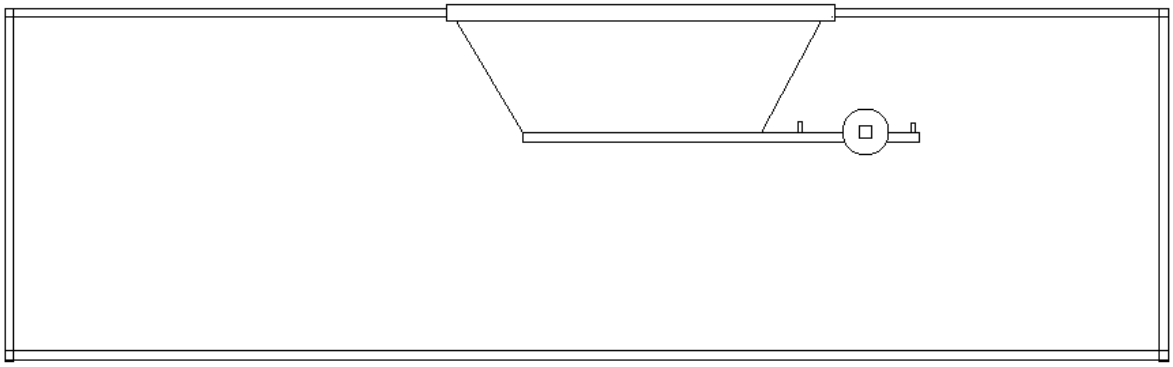


(a)

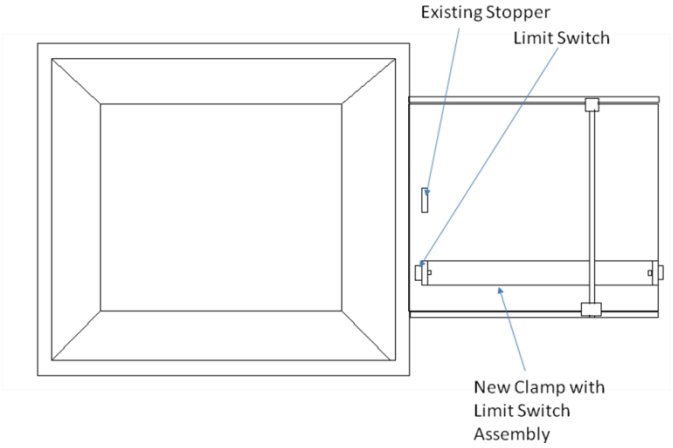
(b)

(c)

Figure 12. Z-axis limit switch design and assembly. a) Image showing limit switch locations in the z-axis assembly; b) clamp location and limit switch assembly on z-axis forward movement; c) clamp location and limit switch assembly in reverse movement (Dimensions are in mm)



(a)



(b)

Figure 13. Limit switch design and assembly on the hopper. a) image showing limit switch locations in the hopper elevation; b) image showing limit switch locations in the hopper plan

5.3. Control Assembly

The control box (Fig. 14) to control the motor consisted of 16 solid state input/output modules mounted on a 24 position backplane with LED indicators to indicate signal status, and equipped with pull-up resistors to avoid undefined states, and power fuses for overload protection on each channel. The control box was connected to the PCI-DIO 24 card (Omega Engineering Inc, Laval, PQ) using a 1 m ribbon cable with 50 pin connector, and was housed in an enclosure equipped with an emergency stop switch.

(a) (b)

Figure 14. Control Box. a) image showing control box enclosure with emergency switch indicated by the arrow; b) image showing solid state input and output devices

5.4. Motor Control

A motor control program was designed in VB.Net using universal library to externally control the motors, and the interface is shown in Fig. 15. The interface controlled the motors using the clicking of check boxes. It was designed to have two check boxes for each motor motion, namely x-axis motor clockwise motion, x-axis motor counter clockwise motion, y-axis motor clockwise motion, y-axis motor counter clockwise motion, z-axis motor clockwise motion, z-axis motor counter clockwise motion, hopper

opening motor clockwise motion, hopper opening motor counter clock wise motion as shown in Fig. 15.

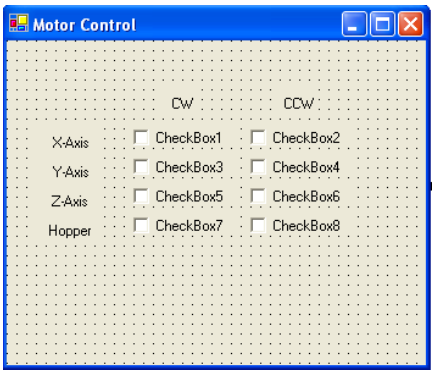


Figure 15. Motor control interface

Checking the check box activated the motor to rotate in the appropriate direction, and checking it off switched the motor activation. The program also deactivated the control if the limit switches were activated, and alerted the users by changing the color of the text box of the control. The program during launch switched off all the motors by writing zeros to all the bits in the ports. Port A was used to control the motors and port B was used to read the input from the limit switches.

This was designed to control and study the speeds of motion along the different axes, and to record the template image when the tool was at the fully retracted position in the z-axis. A housing to position the camera was constructed and attached to the y-axis assembly so that the camera could navigate along with the tool to record the tools position.

6. IMPLEMENTATION OF GRAIN IDENTIFICATION

Three grain types namely CWRS wheat, barley, and canola, were acquired for the classification algorithm development. The samples for bulk image analysis were obtained by pouring 1 kg of grain kernels into a large plastic bag and shaking it to mix the grain thoroughly. The grain was then slowly poured into a petri dish until it was completely filled. Excess grain was removed from the dish so that the top level of grain was almost horizontal and matched the rim of the petri dish. The position of the petri dish was marked so that the placing of the petri dish would be consistent throughout the sample collecting process. This process was repeated 1000 times for each grain type resulting in a total of 3000 bulk sample images.

6.1. Image acquisition setup

The camera set up consisted of a mono-chrome CCD camera (Model no.: DS-21-02M30 DALSA Corporation, Toronto, ON) with a 1600 x 1200 resolution.

Figure 16. Camera setup to identify grain

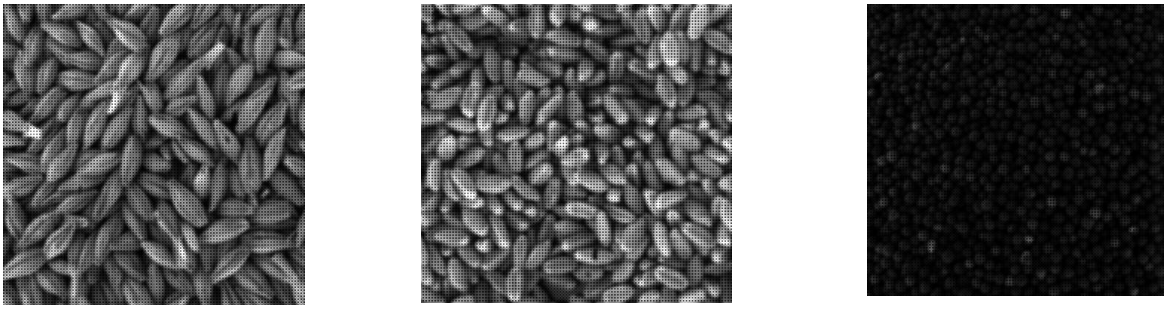
The camera was connected to the PC for image acquisition using a Matrox Solios Single-Base 85MHz, Camera Link® PCI-X frame grabber (Matrox Electronic Systems Ltd, Montreal, PQ) with 64 Mb DDR SDRAM (Fig. 16).

The camera was mounted on a camera stand (M3, Bencher Inc., Chicago, IL). A circular fluorescent tube light with a 305 mm diameter 32-W circular lamp (FC12T9/CW, Philips, Singapore) with a rated voltage of 120 V was in a lighting enclosure.

An enclosure was made up of a semi-spherical steel bowl of 390 mm diameter. The inner side of the bowl was painted white and smoked with magnesium oxide to reflect the light and produce diffuse reflectance.

6.2. Image Acquisition and Algorithm Development

As described above 1000 images each of CWRS wheat, barley, and canola were collected for algorithm development.

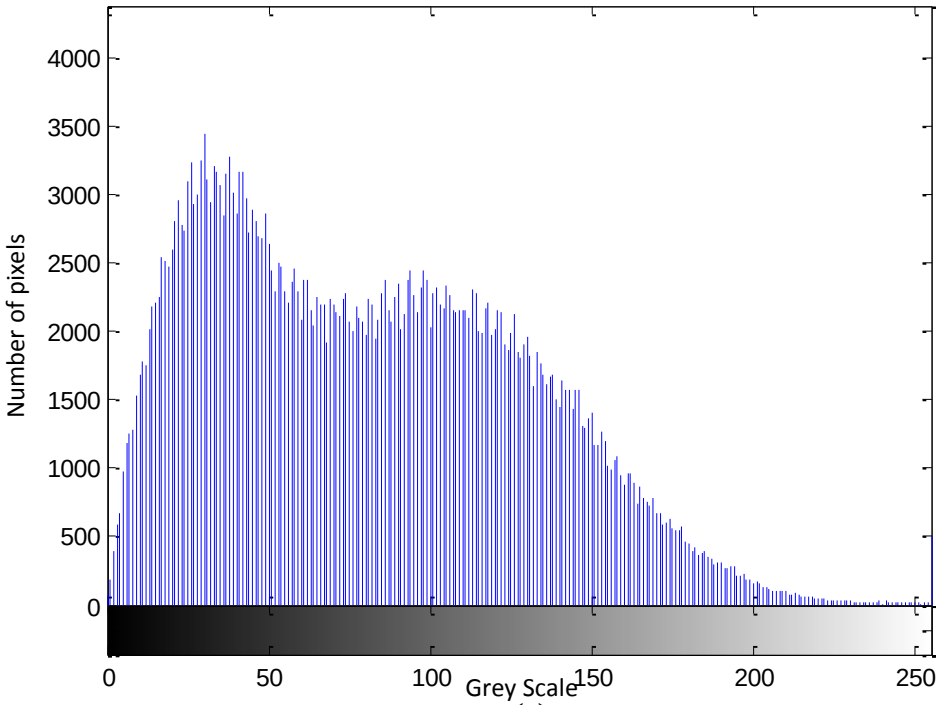


(a) **(b)** **(c)**

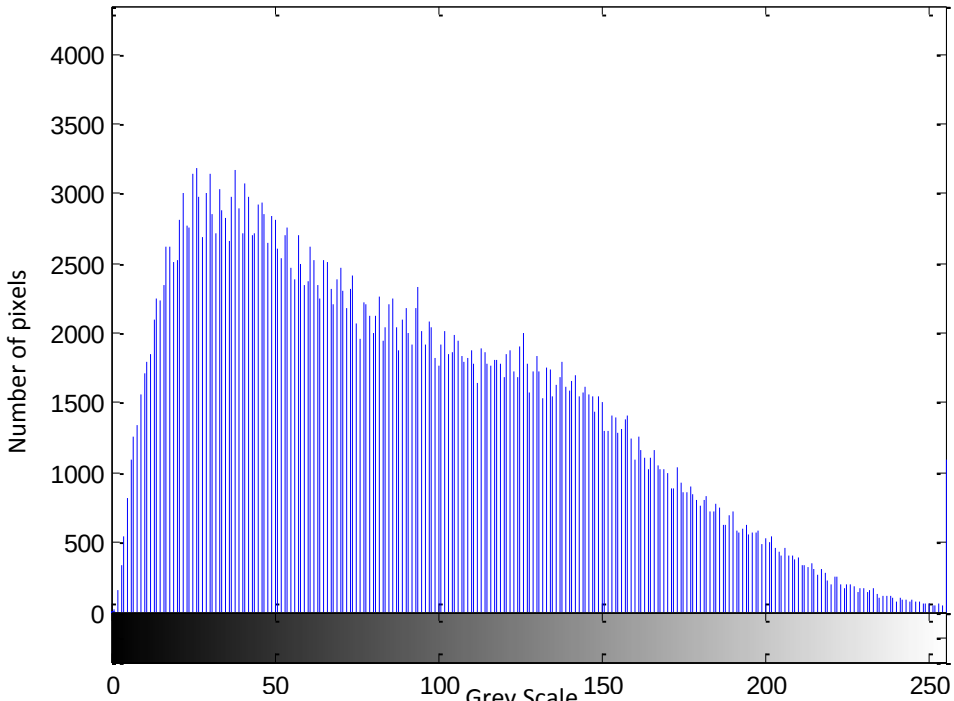
Figure 17. Raw images. a) barley; b) CWRS wheat; c) canola

To eliminate the petri dish boundary an area of 600 pixels x 600 pixels was selected. The area encompassing the selected pixels were separated from the original image and the recorded as individual sub image (600 X 600 pixels) Fig. 17.

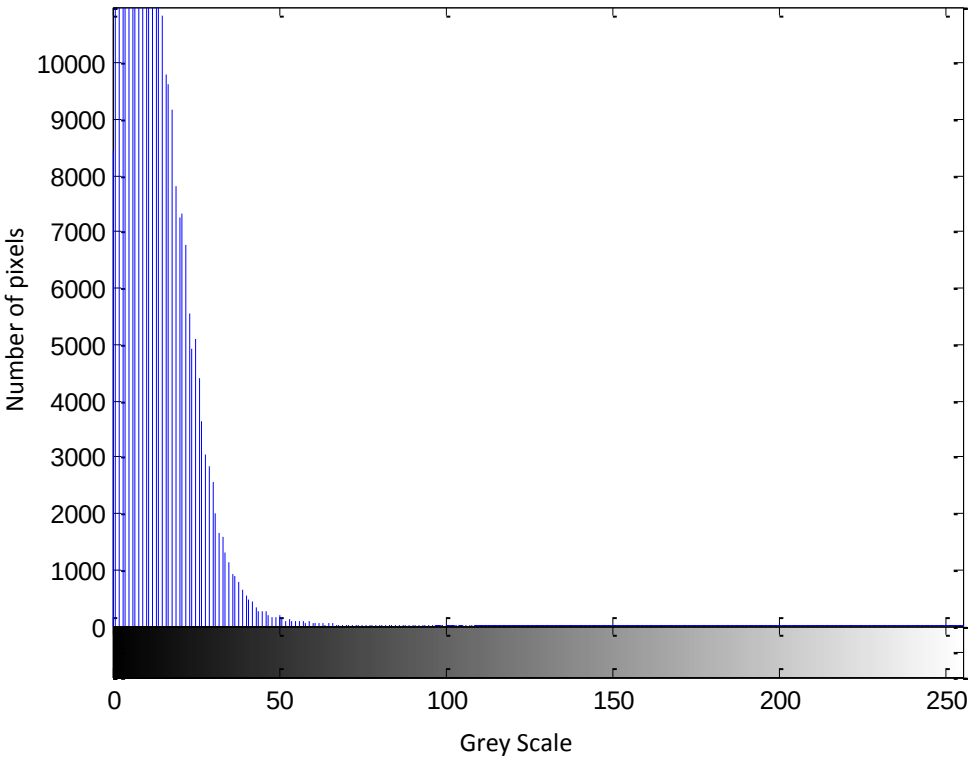
The histograms of the raw images were processed to visually identify the differences in pixel intensities (Fig. 18).



(a)



(b)

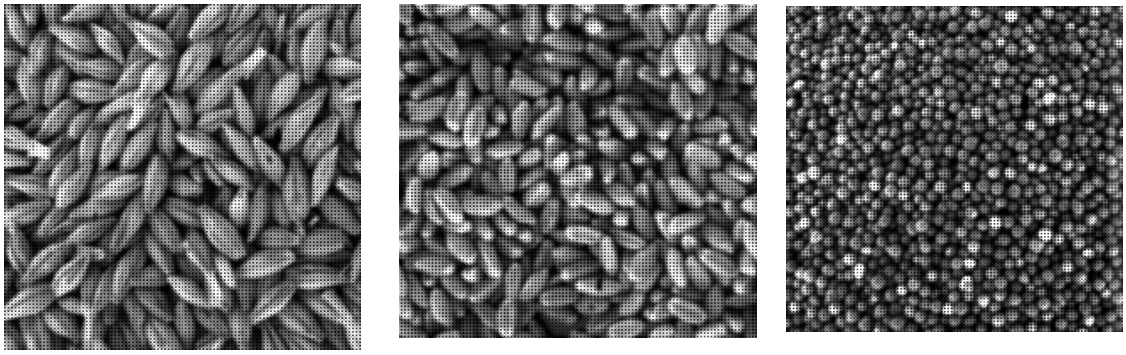


(c)

Figure 18. Raw image histograms. a) barley; b) CWRS wheat; c) canola

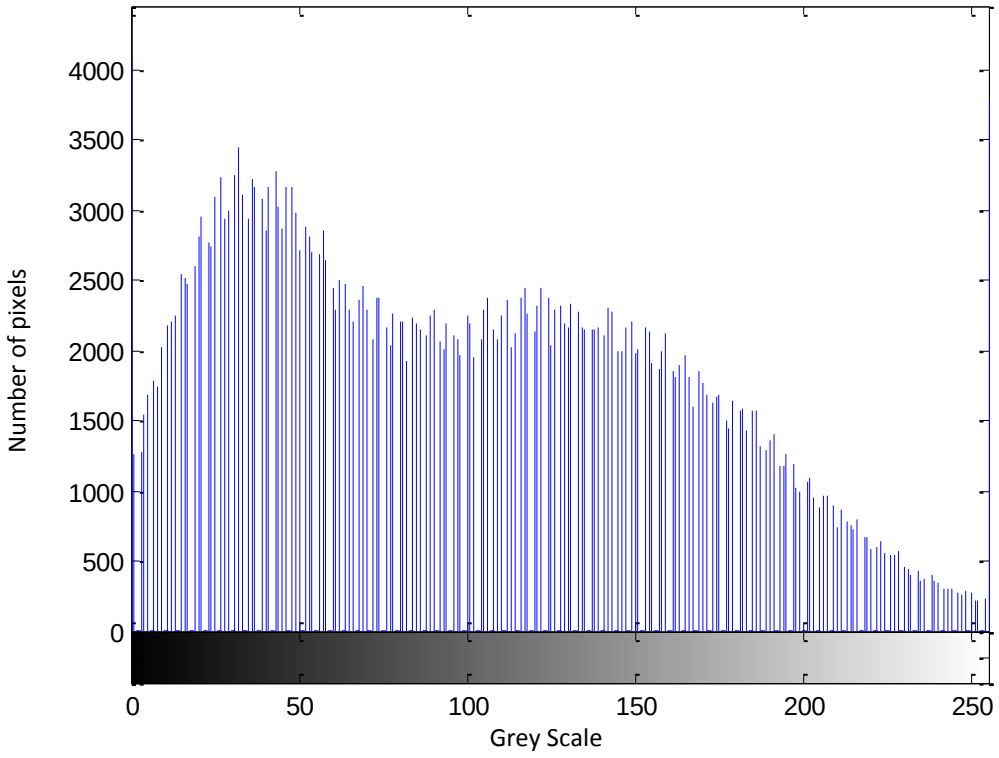
The histograms displayed in Fig. 18, show a significant change in the pixel intensity distribution ranging from 175 to 250 grey values between barley, CWRS wheat, and canola.

However to further enhance the classification and extract additional features the images were processed using the "imadjust" function where the function mapped the intensity values in the selected image to new values in J such that 1% of data was saturated at low and high intensities of original image. This increased the contrast of the output image (Fig. 19). The respective histograms are shown in Fig. 20.

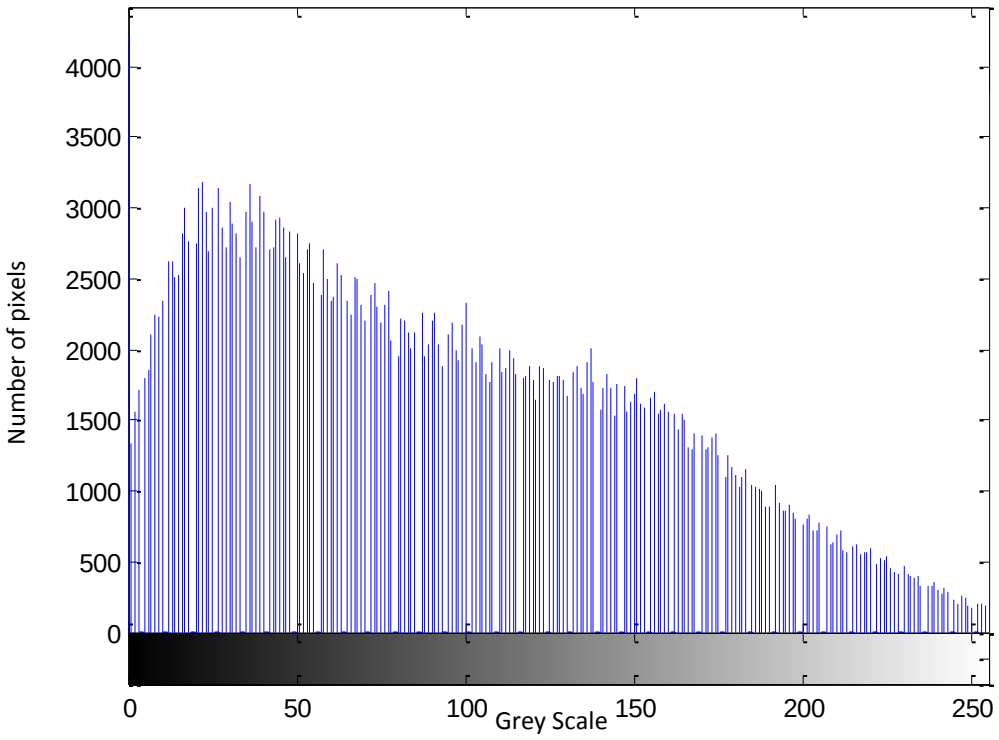


(a) **(b)** **(c)**

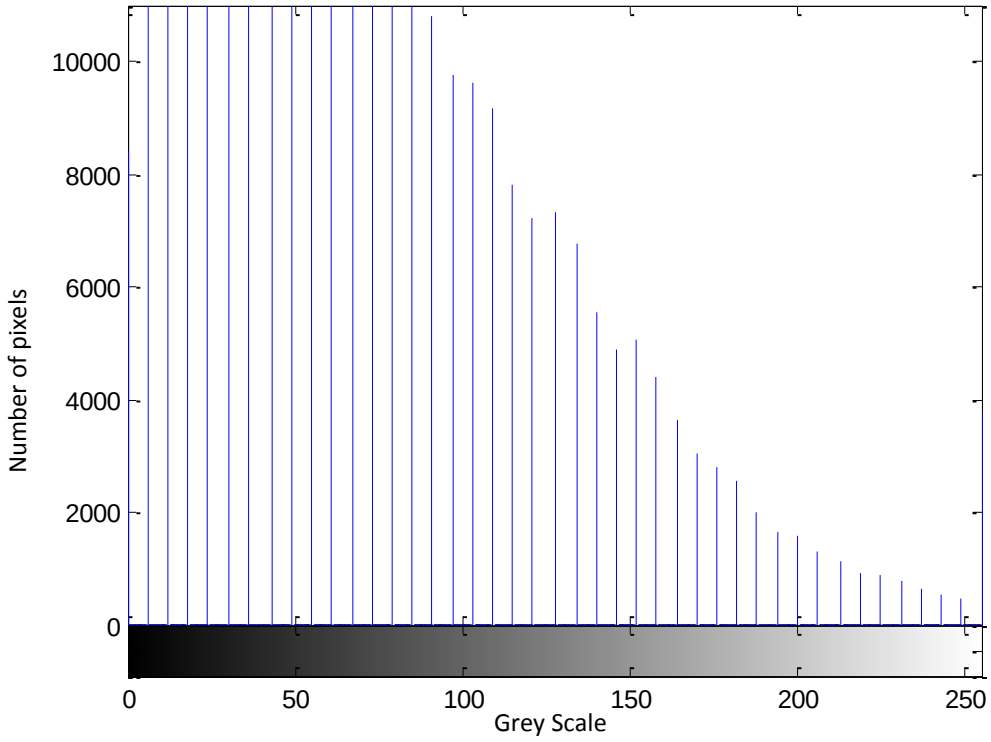
Figure 19. Intensity adjusted images. a) barley; b) CWRS wheat; c) canola



(a)



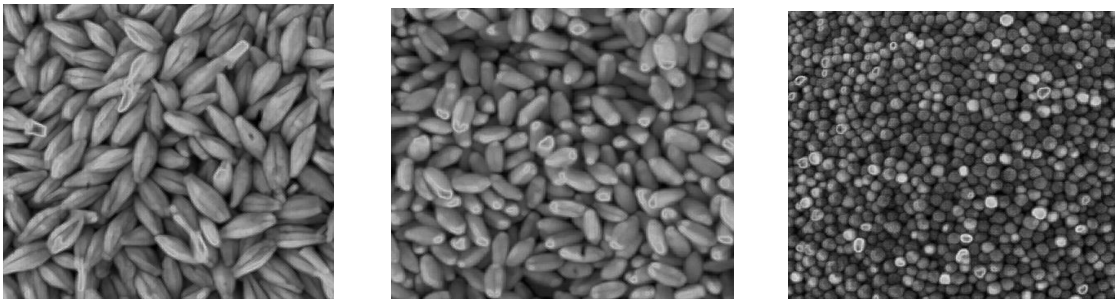
(b)



(c)

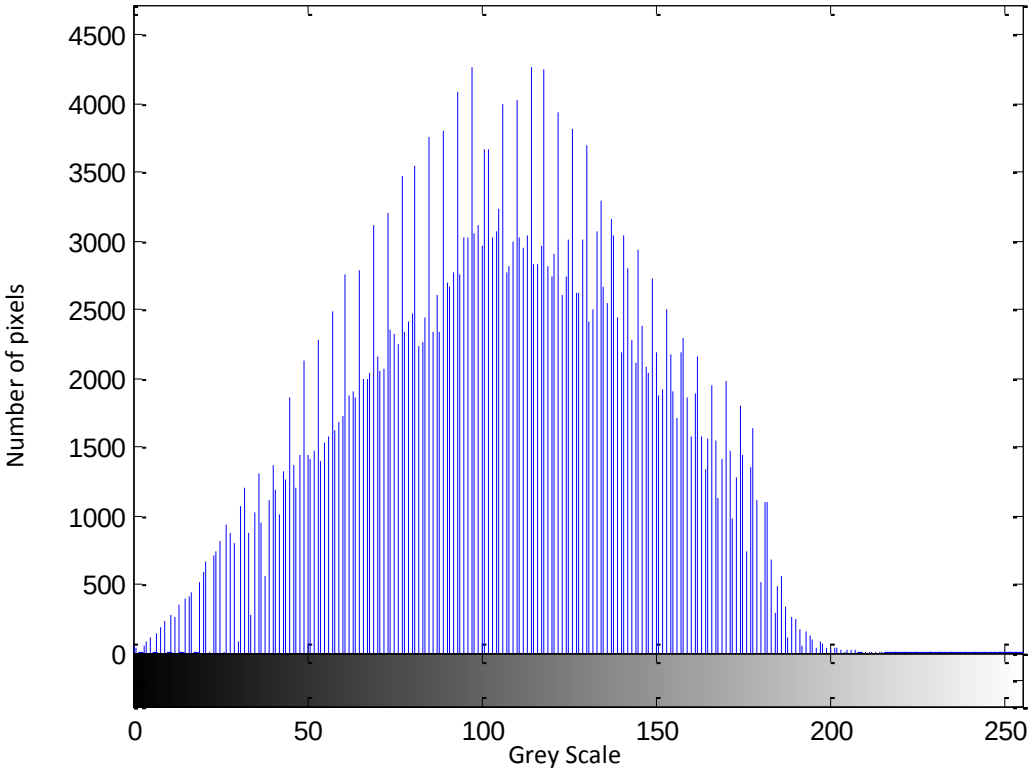
Figure 20. Intensity adjusted image histograms. a) barley; b) CWRS wheat; c) canola

This process introduced a change in the pixel intensity distribution of the canola image but reduced the variation between CWRS wheat and barley in the 175 – 200 pixel intensity range. So a further processing of the histogram was done prior to finalizing the level of processing required, thus the images were further processed using a range filter function, where each output pixel contained the range value (maximum value − minimum value) of the 3-by-3 neighbourhood around the corresponding pixel in the input image (Fig. 21). The process however further reduced the ease of classification capability between CWRS wheat and barley as seen in the histograms (Fig. 22).

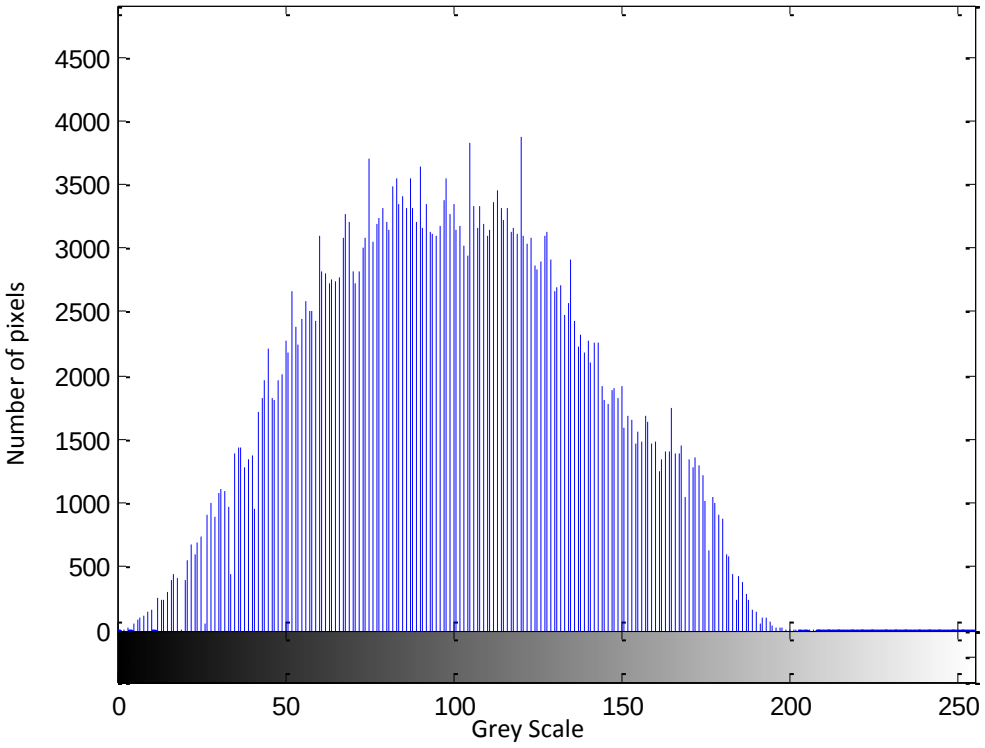


(a) **(b)** **(c)**

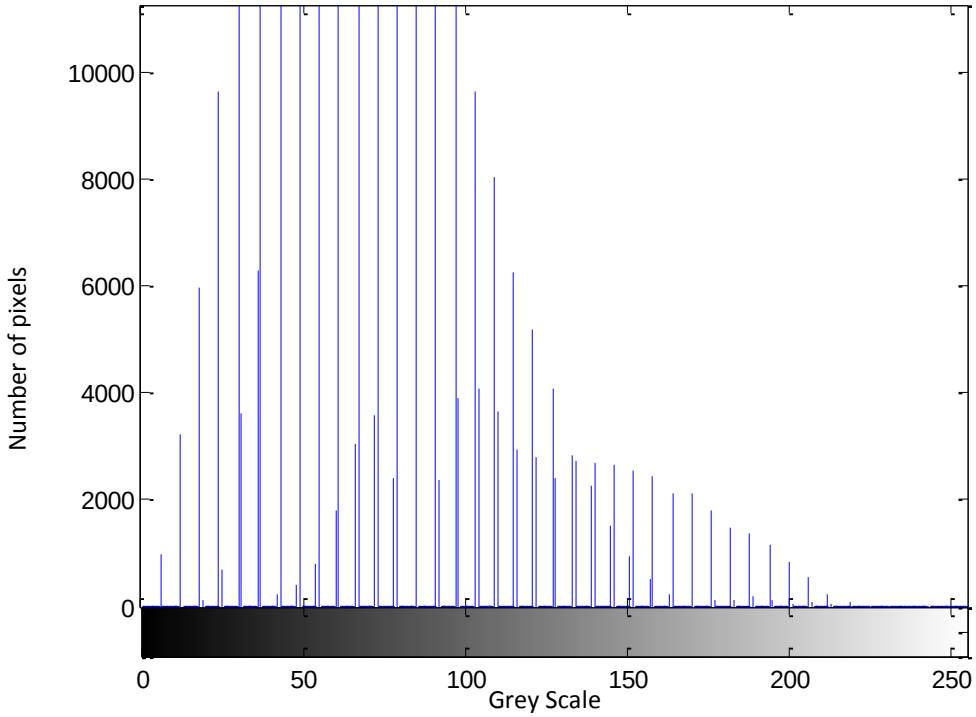
Figure 21. Histogram equalized images. a) barley; b) CWRS wheat; c) canola


Number of pixels
4500
4000
3500
3000
2500
2000
1500
1000
500
0
0
50
100
150
200
250
Grey Scale

(a)


Number of pixels
4500
4000
3500
3000
2500
2000
1500
1000
500
0
0
50
100
150
200
250
Grey Scale

(b)



(c)

Figure 22. Histogram equalized image histograms. a) barley; b) CWRS wheat; c) canola

Five features were collected for the classification namely, count of pixels ranging between 45 and 55, 175-185, 195-205, 205-215 and the count of pixels with an intensity of 200. If the count of the pixels in the intensity of 200 was greater than 0 or less than 250, and the count of pixels in the intensity range of 205 – 215 was greater than 100 or less than 1000, and the count of the pixels in the intensity range of 175-185 was greater than 1000 or less than 5000 and the count of the pixels in the intensity range of 195-205 was greater than 100 or less than 2000, the image was classified as barley. If the count of the pixels in the intensity of 200 was greater than 250 or less than 700, and the count of pixels in the intensity range of 205 – 215 was greater than 2500 or less than 4000, and the

count of the pixels in the intensity range of 175-185 was greater than 6000 or less than 9000 and the count of the pixels in the intensity range of 195-205 was greater than 3500 or less than 6000, the image was classified as CWRS wheat. If the count of the pixels in the intensity of 200 was equal to zero, and the count of pixels in the intensity range of 205 – 215 was equal to zero, and the count of the pixels in the intensity range of 175-185 was equal to zero and the count of the pixels in the intensity range of 195-205 was equal to 0, and the count of the pixels in the intensity range of 45-55 was greater than 100 or less than 5000, the image was classified as canola. Otherwise it was classified as unknown.

Twenty images were used as training images and the rest were used as test images. The results of the classifications are given in Table 4.

Table 4. Initial classification accuracy of the grain identification algorithm using raw and processed images.

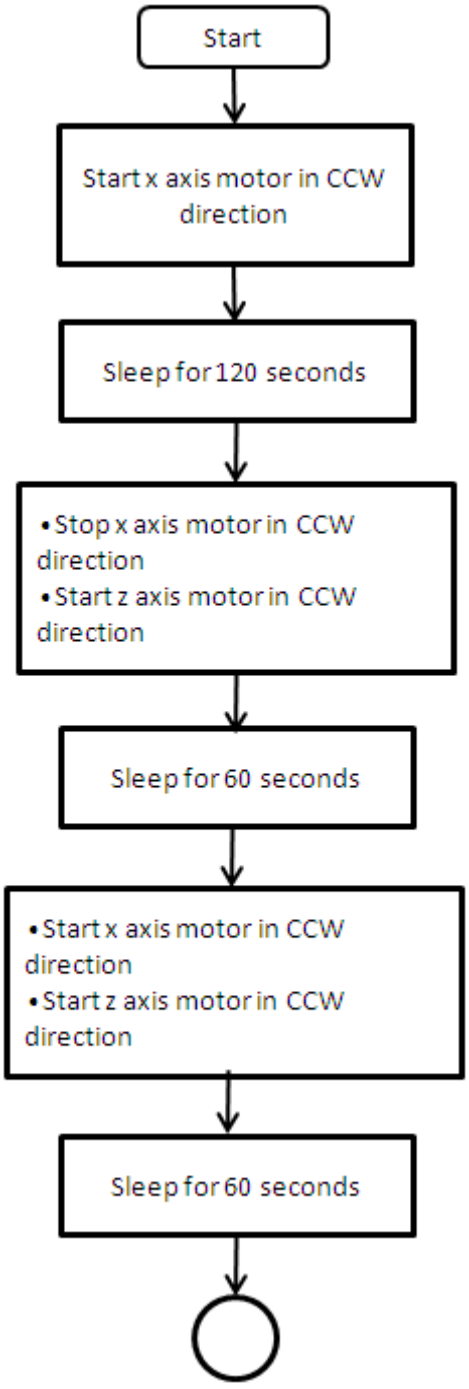
Grain	Classification Accuracy		
	Raw-Image	Image adjusted Image	Range filtered image
Barley	100%	78%	62%
CWRS	100%	32%	28%
Canola	99.9%	82%	36%

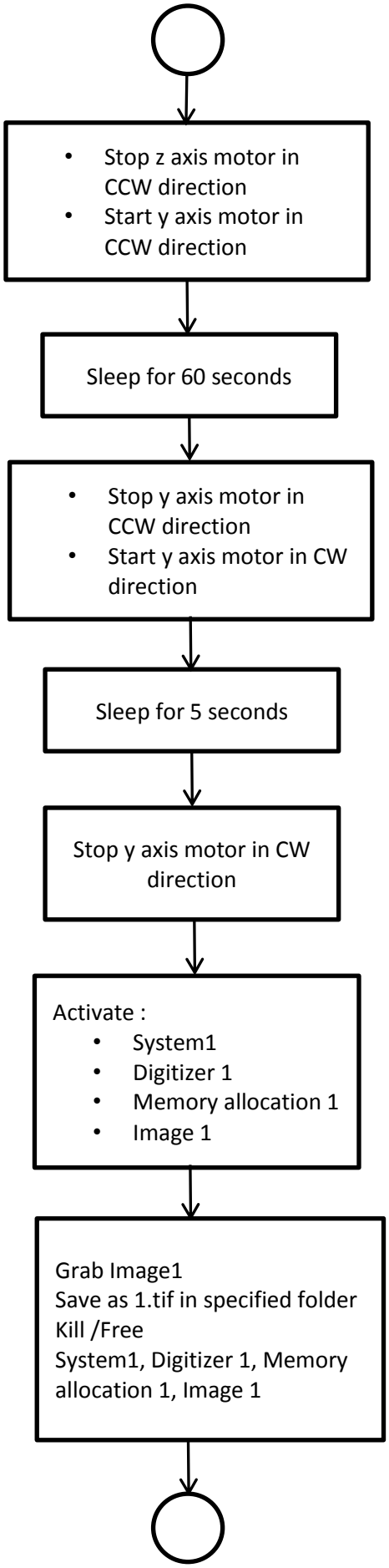
Similar to the inspection of the visual information from the histograms the results using the above features had dropped.

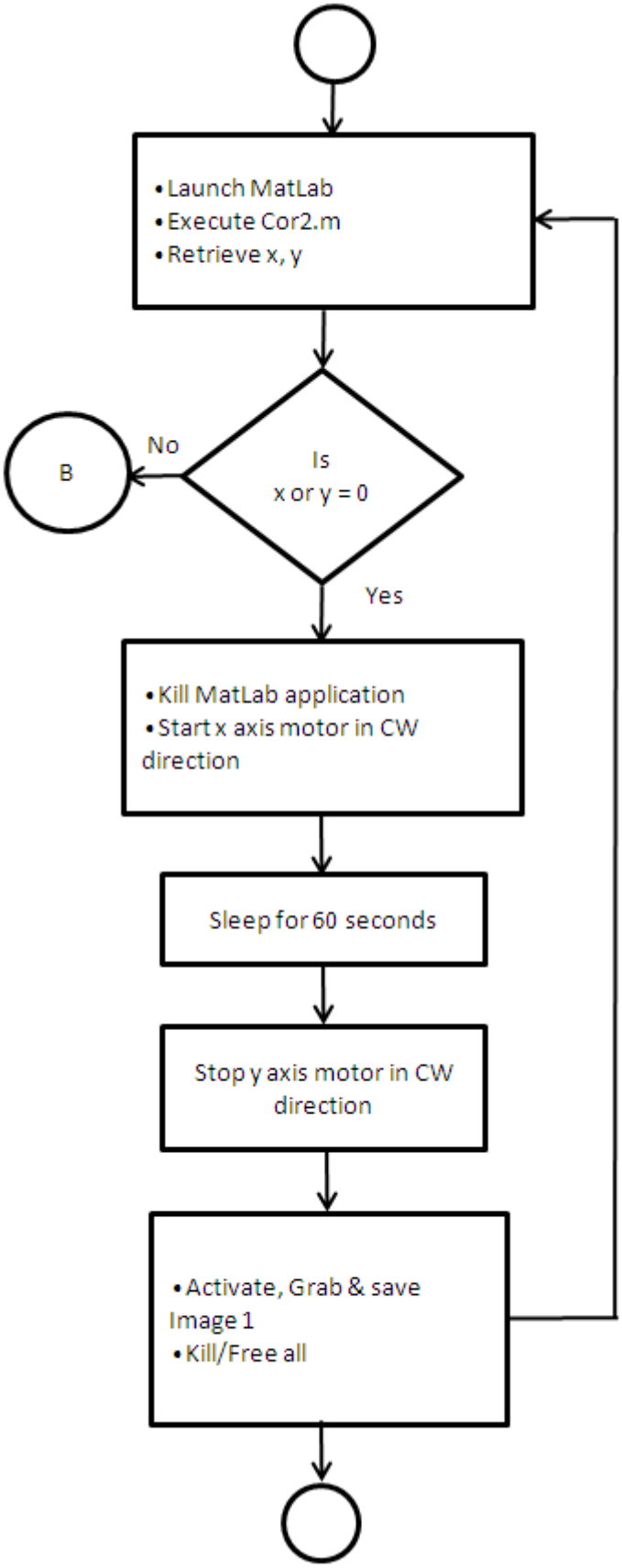
Following the initial study, an additional 1000 images of each grain were collected and tested with the algorithm for raw images, and the results were a 100% classification for barley, 99.9% for CWRS wheat and 99.9% for canola.

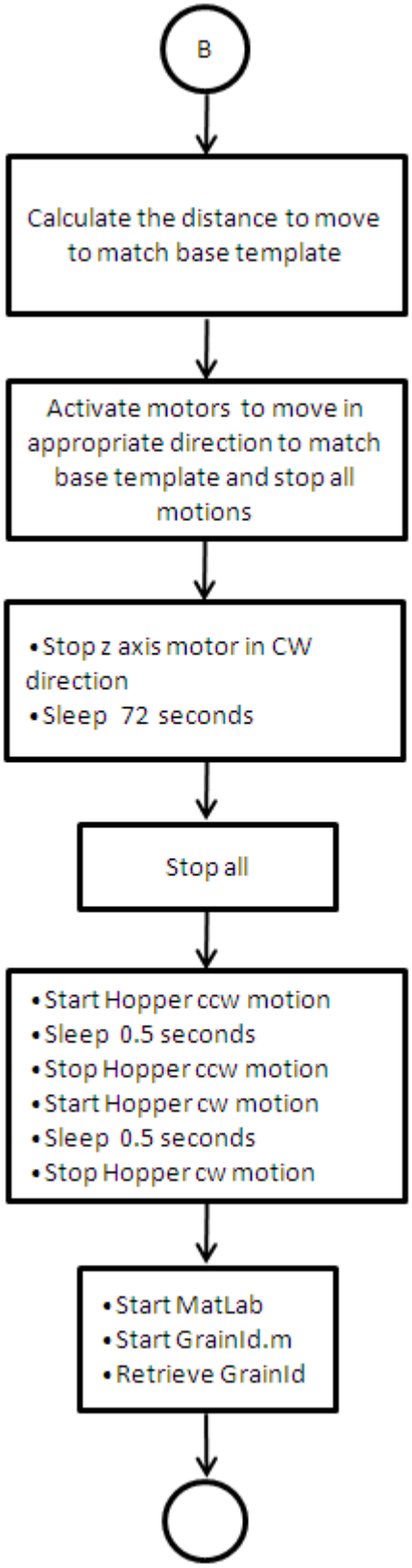
7. IMPLEMENTATION OF THE "GRAIN-O-BOT" FOR GRAINCAR UNLOADING

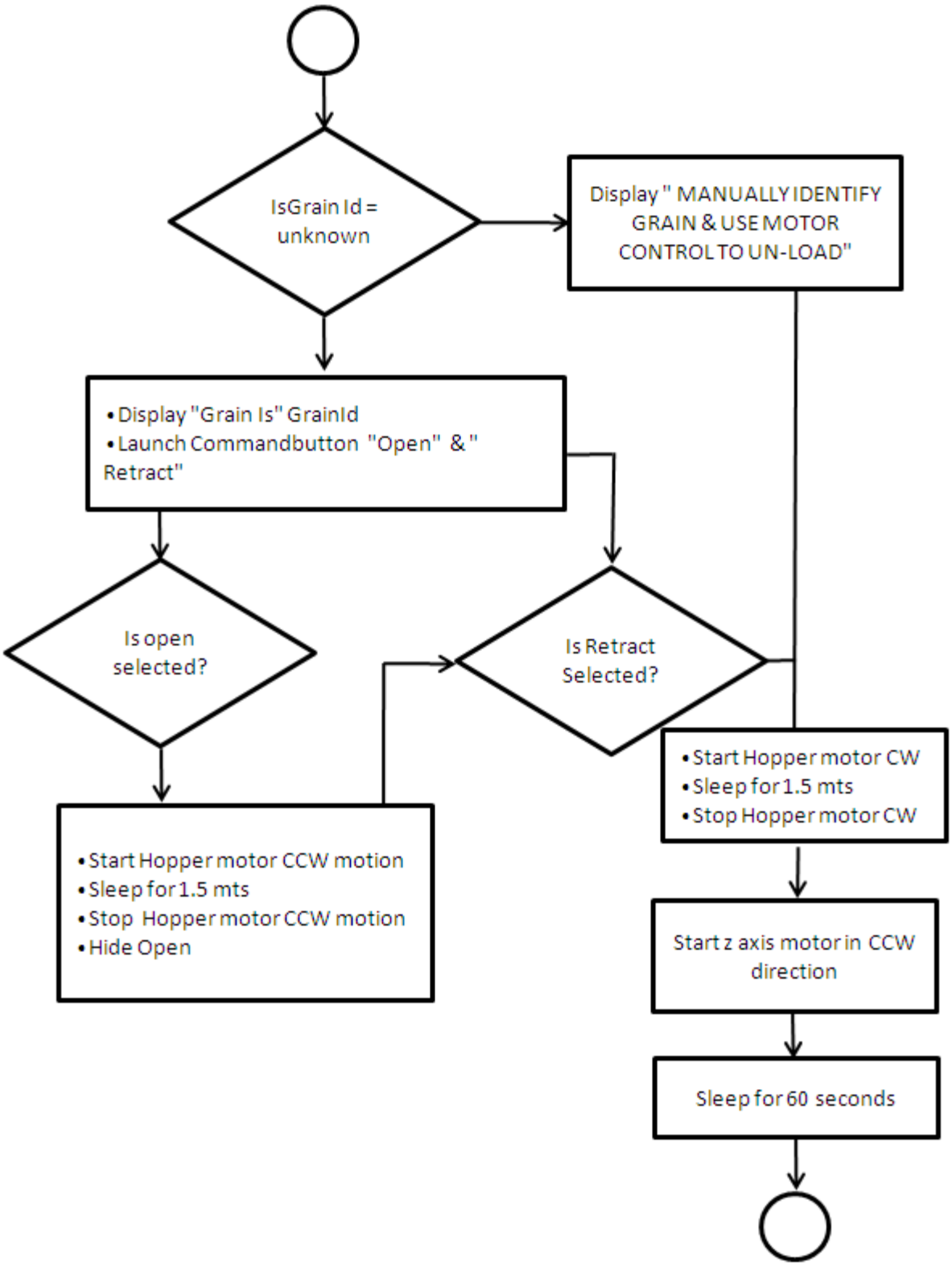
The fabrication of the "Grain-o-bot" structure was followed by the integration of the programs to control the "Grain-o-bot". The Vb.net program was designed as per the flow chart shown. The programs are given in Appendix B.




• Stop z axis motor in CCW direction
• Start y axis motor in CCW direction
Sleep for 60 seconds
• Stop y axis motor in CCW direction
• Start y axis motor in CW direction
Sleep for 5 seconds
Stop y axis motor in CW direction
Activate :
• System1
• Digitizer 1
• Memory allocation 1
• Image 1
Grab Image1
Save as 1.tif in specified folder
Kill /Free
System1, Digitizer 1, Memory allocation 1, Image 1


• Launch MatLab
• Execute Cor2.m
• Retrieve x, y
Is
x or y = 0
No
B
Yes
• Kill MatLab application
• Start x axis motor in CW direction
Sleep for 60 seconds
Stop y axis motor in CW direction
• Activate, Grab & save Image 1
• Kill/Free all


B
Calculate the distance to move to match base template
Activate motors to move in appropriate direction to match base template and stop all motions
• Stop z axis motor in CW direction
• Sleep 72 seconds
Stop all
• Start Hopper ccw motion
• Sleep 0.5 seconds
• Stop Hopper ccw motion
• Start Hopper cw motion
• Sleep 0.5 seconds
• Stop Hopper cw motion
• Start MatLab
• Start GrainId.m
• Retrieve GrainId


IsGrain Id = unknown
Display " MANUALLY IDENTIFY GRAIN & USE MOTOR CONTROL TO UN-LOAD"
• Display "Grain Is" GrainId
• Launch Commandbutton "Open" & " Retract"
Is open selected?
Is Retract Selected?
• Start Hopper motor CCW motion
• Sleep for 1.5 mts
• Stop Hopper motor CCW motion
• Hide Open
• Start Hopper motor CW
• Sleep for 1.5 mts
• Stop Hopper motor CW
Start z axis motor in CCW direction
Sleep for 60 seconds

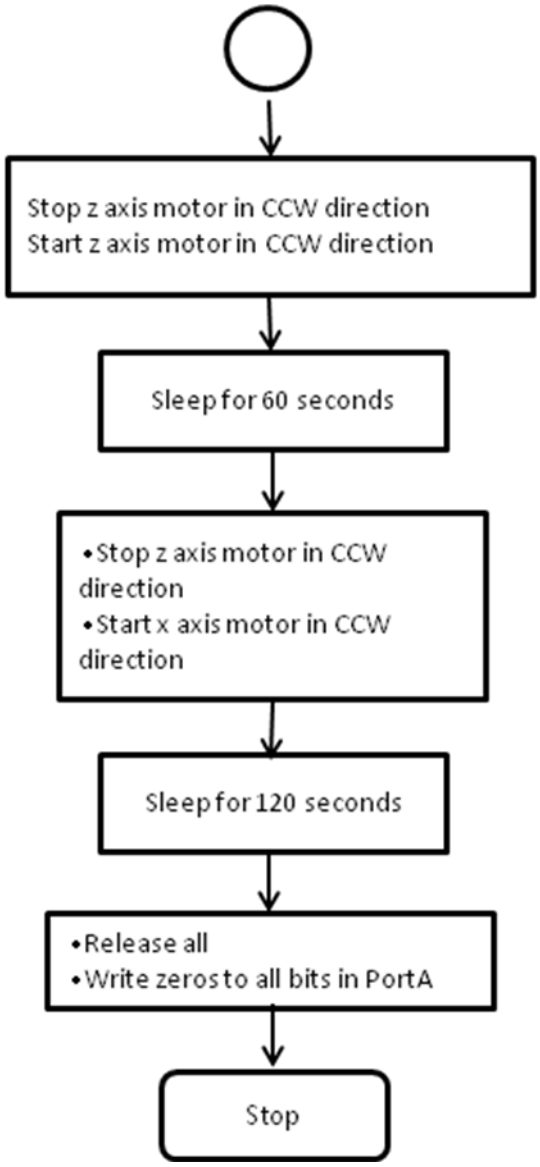


Figure 23. Flowchart indicating the flow of "Grain-o-bot" logic

As illustrated in the flow chart (Fig. 23), the steps were followed and the program for the "Grain-o-bot" was built and tested. The "Grain-o-bot" initially returns to the start position irrespective of where it was positioned. This step was necessary to avoid the "Grain-o-bot" from starting from an unknown position, in the event of a failure or an emergency stop in the previous run. The length of time it moved in all axes to arrive at

the known location was determined by the time it took the "Grain-o-bot" to travel along the respective axes to reach the other ends of the axes. Once the "Grain-o-bot" arrived in the known position, it moves upward along the y-axis for 5 s. The camera at the start of the y-axis was not able to capture the sprocket in its field of view. Hence it was necessary to raise the camera to a position where the sprocket would be visible, during the scan. At this point motion along all axes was stopped and the Matrox - Metero II 1394 digitizer was initialized and the Sony XCD-X 700 camera was activated to capture an image. The captured image was saved as a tiff file in a pre-determined location for the sprocket identification using the Matlab program. The Matlab program was started as an object and the captured image was tested for the presence of the sprocket using the program. If the sprocket was not identified, a "0" was returned by the Matlab program and the Matlab object was killed. The x-axis motor was activated to move in the clockwise direction which in turn moved the "Grain-o-bot" along the x-axis for a period of 60 s. The 60 s was calculated and later confirmed by the time taken to move the end-effector across 50% of the camera's previous view. This was necessary to ensure that a partially exposed sprocket in the 1st field of view would be completely visible in the second shot. This process was repeated until the sprocket was identified. If the sprocket was not identified within 6 iterations a message was programmed to appear requesting the end user to check for the existence of the sprocket manually. If the Matlab-program identified the sprocket then the location of the sprocket was returned and the Matlab object was killed.

The position the end-effector needed to be in order to mate exactly with the sprocket was identified based on first manually mating the end-effector to the sprocket and retracting the "Grain-o-bot" back along the z-axis to the zeroth position of the z-axis. An image was

captured and stored and the co-ordinates were recorded as the location the sprocket identified by the Matlab program had to be in order to get a prefect lock. The difference between the location returned by the Matlab program and the location identified by the pre-determined image was used to move the grain-o-bot along the x-axis and followed by moving along the y-axis by actuating the motors in the appropriate direction. Once the location was reached the z-axis motor was actuated to travel toward the sprocket for 73 s. The time was calculated as the rate of feed and confirmed. Following this the end-effector motor turned in the clockwise direction for 0.5 s and then turned counter clockwise for 0.5 s. This motion was to facilitate the contents of the grain car dropping into a predetermined area for content identification.

The Matrox Solios frame grabber was activated to trigger the Dalsa camera, which in turn captured the content image and saved it to a known location. Following this, frame grabber and the memory were released and the content identification program was launched using the Matlab object. The program identified the content as CWRS wheat, barley, or canola or returned ”not identifiable” as a message. A button needed to be clicked to open the hopper gate or another button to return to the base location if the content was not acceptable.

8. CHALLENGES IN IMPLEMENTATION

The integration of the program with the hardware posed challenges unanticipated in the initial plan.

Thorough analysis of lighting conditions that affected sprocket detection and using the findings in the implementation greatly assisted in the subsequent steps.

The Matrox Meteor II frame grabbers during the integration of the sub programs stopped functioning. To solve this problem the frame grabbers were removed and the software was re-installed. However this was met with compatibility issues with Kaspersky Antivirus program. Uninstalling the antivirus program resolved the issue of loading the software and connecting them with their respective cameras to function as expected. Following the identification of the issue with the Meteor II frame grabbers were replaced with Meteor Solios frame grabbers to completely resolve the imaging errors.

The location of the Sony camera which was used to detect the sprocket was lower than the field of view to capture the sprocket. The issue was eliminated by raising the start location of the y-axis, so that the sprocket would be visible when moved on the x-axis. This is acceptable as the graincar will always be at a fixed height with respect to the tracks and so the sprocket location along the y-axis will vary marginally and the range of 2-4 cm will be visible within the field of view at this height.

The rate of feed which was designed to be the same across all the axes was actually different between the different axes. This was identified as a result of the non-coordination of the movement using the calculated values. This was eliminated by initially capturing an image and running each motor for a fixed time and re-capturing an image. A known object was selected and its displacement over the two images was

measured over the time of travel to calculate the rate of travel. This was repeated for each axis in both directions to arrive at the exact feed.

The “Grain-o-bot” was completely controlled by “on” and “off” of motors in one direction or the other. One of the solid state relays that controlled the motor to open the hopper gate was stuck on “On” and had to be replaced. However the failing of the hardware hammered the gate.

The Matlab programs that were initially constructed to detect the sprocket had used an image where the camera was placed around 1 m from the sprocket. The size of the sprocket obtained from this image was 98 x 98 pixels. However in real time the camera was much closer to the sprocket and the size of the sprocket obtained was 125 x 125 pixels. The Matlab programs were altered to work with the new size.

The closeness of the camera to the sprocket brought out more features than the previous images. This was addressed by adding a pre-processing function which eliminated the noise prior to identifying the sprocket.

Performance as a result of multiple programs running at the same time was an issue. The main program started to slow its responses as a result of all the frame grabbers and the images that were stored. This was addressed by using the programs and hardware resources as needed and once they had completed their intended operation, they were immediately released and the memory was freed up.

9. CONCLUSIONS AND RECOMMENDATIONS

The research work conducted in this thesis explored the potential of robotics and image processing in automating the unloading of grain in elevators. A Cartesian robot equipped with a camera acting as an eye and another camera acting as a grain identifyier was integrated as a "Grain-o-bot" to perform this automation.

Using three types of light sources, three algorithms were evaluated for their performance in detecting the hopper gate sprocket on a model of a graincar hopper gate. Three algorithms were developed based on correlation and shape detection techniques. The algorithm using correlation followed by shape detection performed better than the other two algorithms in detecting the sprocket with no false positives. The images acquired using the diffuse incandescent light source were less influenced by external variations. Under proper illumination, the sprocket was correctly identified at all times, and external variables had minimal influence on the identification. Although the processing speed of Algorithm III was three times longer than Algorithms I and II, it is beneficial to use Algorithm III as it produced no false identifications.

The "Grain-o-bot" was able to accurately reach, open and close the hopper gate to release a small portion of the contents. The grain identification was possible using five features. A classification accuracy of 99-100% was achieved between CWRS wheat, barley, and canola.

The programs are written for the test setup which replicates the real time situations. However, implementing this project at a work site needs studying of the site carefully and fine-tuning the programs and the lighting conditions to the particular site. The

occurrence of extreme lighting conditions needs to be investigated at each location, and precautionary methods relevant to the site need to be applied.

The rate of motion of the “Grain-o-bot” had been intentionally reduced for the study. This needs to be improved in the future. The “Grain-o-bot” currently uses a computer to process the programs. Furture studies need to transfer the programs and computing to a micro controller to optimize design.

The “Grain-o-bot” can also be used to unload all granular products that are discharged through a hopper. Hence studying other granular materials that are transported and incorporating their identification algorithms will make the “Grain-o-bot” more useable.

10. REFERENCES

Allotta. B., G. Buttazzo, P. Dario, P. Levi, and F. Quaglia. 1990. A force/torque sensor-based technique for robot harvesting of fruits and vegetables. In *Proceedings of the IEEE International workshop on intelligent robots and systems, 231*-235. Tsuchiura, Ibaraki, Japan, July 5-6

Anonymous. 2011a. The role of variety identification in Canadian grain quality assurance. http://www.grainscanada.gc.ca/technology-technologie/vicgq-ivqgc-eng.htm. (2011/07/18)

Anonymous. 2011b. Monitoring the Canadian grain handling and transportation system. http://www.tc.gc.ca/eng/policy/report-acg-grainmonitoringprogram-ghts_appendix-228.htm. (2011/07/18).

Anonymous. 2011c. Grain elevators . http://www.grainscanada.gc.ca/wa-aw/geic-sgc/search-recherche-eng.asp. (2011/07/18).

Anonymous. 2011d. Grain handling and marketing. http://www.thecanadianencyclope-dia.com/index.cfm?PgNm=TCE&Params=A1ARTA0003364. (2011/07/18).

Anonymous. 2011e. Country elevator guide. 2011-12 Elevator guide. http://www.cwb-.ca/public/en/farmers/contracts/pdf/1112_elevator_guide.pdf (2011/07/18).

Anonymous. 2011f. Using producer cars to ship prairie grain. http://www1.agric.gov-.ab.ca/$department/deptdocs.nsf/all/sis12325 (2011/07/18).

Anonymous. 2011g. Ball screw assembly selection charts. http://www.nookindustries .com/ball/BallCharts.cfm (2011/07/18).

Anonymous. 2011h. Trends in the automation of agricultural field machinery. http://www.clubofbologna.org/ew/documents/KNR_Sherear.pdf . (2011/07/18).

Bato M.P., M. Nagata, Q. Cao, B.P. Shrestha, and R. Nakashima. 1999. Strawberry sorting using machine vision. ASAE Paper No. 993162, St. Joseph, MI: ASAE.

Bourely. A.J., T.C. Hsia, and S.K. Upadhyaya. 1986. Investigation of a robotic egg candling system. In *Proceedings of the Agri-Mation 2*, 53-62. St. Joseph, MI: ASAE.

Bye, P. and J.J. Chanaron. 1986. Economic prospects of agricultural robots. In *Proceedings of the Agri-Mation 2*, 91-98. St. Joseph, MI: ASAE.

CIGI. 1993. *Grains and Oilseeds – Handling, Marketing, Processing,* 4th ed. Winnipeg, MB: Canadian International Grains Institute.

Cadieux, S., F. Michaud, and F. Lalonde. 2000. Intelligent system for fish sorting and counting. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems,* 1279-1284. Takamatsu, Japan. November 1.

Choudhary, R., J. Paliwal, and D.S. Jayas. 2008. Classification of cereal grains using wavelet, morphological, colour, and textural features of non-touching kernel images. *Biosystems Engineering* 99(3): 330-337.

Choudhary, R., S. Mahesh, J. Paliwal, and D.S. Jayas. 2009. Identification of wheat classes using wavelet features from near infrared hyperspectral images of bulk samples. *Biosystems Engineering* 102(2): 115-127.

Chen,Y.R., R.W. Huffman, B. Park, and M. Nguyen. 1994. A transportable spectrometer system for on-line classification of poultry carcasses. ASAE Paper No. 94-3577, St. Joseph, MI: ASAE.

Cogdill, R.P., C.R. Hurburgh, G.R. Rippke, S.J. Bajic, R.W. Jones, J.F. McClelland, T.C. Jensen, and J. Liu. 2004. Single kernel maize analysis by near-infrared hyperspectral imaging. *Transactions of the ASAE* 47(1): 311-320.

Cosmi, F., C. Fabro, P. Susmel, and G. Zoppello. 1997. Automation in dairy farms: A robotic milking system. In *Proceedings of IEEE International Conference on Advanced Robotics* 33-37. Monterey, CA. July 7-9.

Das, K. and M.D. Evans. 1992. Detecting fertility of hatching eggs using machine vision I. Histogram characterization method. *Transactions of the ASAE* 35(4):1335-1341.

Delwiche, M.J., S.Tang, and J.F. Thomson. 1993. A high-speed sorting system for dried prunes. *Transactions of the ASAE* 36(1):195-200.

de Silva, C.W. 1991. Gauging and positioning issues of robotic fish processing. In *Proceedings of the IEEE 5th International Conference on advanced robotics, 'Robots in Unstructured Environments'* 1: 291-296. Pisa, Italy. June 19-22.

de Silva, C.W. and M. Saliba, 1992. Instrumentation issues in the handling of fish for automated processing. In *Proceedings of the IEEE Conference on Industrial Electronics, Control, Instrumentation and Automation* 2: 789-794. San Diego. November 9-13.

Edan, Y., I. Wolf, J. Grinshpun, Y. Dobrusin, and V. Rogozin. 1994. Robotic melon harvesting: Prototype and field tests. ASAE Paper No. 943073, St. Joseph, MI: ASAE.

Edan, Y., V. Rogozin, T. Flash, and G.E.Miles. 2000. Robotic melon harvesting. In *IEEE Transactions on Robotics and Automation 16:* 831-835.

FAOSTAT, 2009. Food and Agriculture Organization of United Organization, Rome. http://faostat.fao.org/ (2009/03/16)

Faculty of Mechanical Engineering. 1978. *Design Data: Data book of Engineers*. 2nd Edition, Coimbatore, India: Kalaikathir Achagam.

Gonzalez, R.C. and R.E. Woods. 1998. *Digital Image Processing*, 2nd Ed. New York, NY: Addison-Wesley.

Gotou, K., T. Fujiura, Y. Nishiura, H. Ikeda, and M. Dohi. 2003. 3-D Vision system of tomato production robot. In *Proceedings of IEEE International Conference on Advanced Intelligent Mechatronics,* 2: 1210-1215. Kobe, Japan. July 20-24.

Grasso, G. M. and M. Recce.1997. Scene analysis for an orange harvesting robot. *Artificial Intelligence Applications* 11(3): 9-15.

Hannan, M.W., T. F. Burks, and D.M. Bulanon. 2009. A machine vision algorithm combining adaptive segmentation and shape analysis for orange fruit detection. *Agricultural Engineering International: The CIGR Ejournal*, Manuscript 1281, vol. XI.

Huang, Y.G. and F.F. Lee. 2009. An automatic machine vision-guided grasping system for Phalaenopsis tissue culture plantlets. *Computers and Electronics in Agriculture* 70 (1): 42-51

Ito, N. 1990. Agricultural robots in Japan. In *Proceedings of IEEE International Workshop on Intelligent Robots and Systems*, 1: 249-253. Ibaraki, Japan. July 3-6.

Jimenez, A.R., R. Ceres, and J. L. Pons. 1999. A machine vision system using a laser radar applied to robotic fruit harvesting. In *Proceedings of the IEEE Workshop on Computer Vision Beyond the Visible Spectrum: Methods and Applications*, 110-119. Fort Collins, CO. June 6-7.

Karunakaran, C. 2002. Soft X-ray inspection of wheat kernels to detect infestations by stored-grain insects. Unpublished Ph.D. thesis. Winnipeg, MB: Department of Biosystems Engineering, University of Manitoba.

Karunakaran, C., D.S. Jayas, and N.D.G. White. 2004. Soft X-rays: A potential insect detection method in cereals in grain handling facilities. In *Proceedings of International Quality Grain Conference*. Indianapolis, Indiana. July 19-22.

Klassen, N.D., R.J. Wilson, and J.N. Wilson. 1993. Agricultural vehicle guidance sensor. ASAE Paper No. 931008, St. Joseph, MI: ASAE.

Kondo, N. 2003. Fruit grading robot. In *Proceedings of the IEEE/ASME International Conference on Advanced Intelligent Mechatronics.* 2: 1366-1371. Kobe, Japan. July 20-24.

Kondo, N. 2010. Automation on fruit and vegetable grading system and food traceability. *Trends in Food Science & Technology* 22(3): 145-152.

Lee, K., R. Gogate, and R. Carey. 1998. Automated singulating system for transfer of live broilers. In *Proceedings of the IEEE International Conference on Robotics and Automation,* 3356-3361. Leuven, Belgium. May 16-21.

Li, M., K.Imou, K.Wakabayashi, and S.Yokoyama. 2009. Review of research on agricultural vehicle autonomous guidance. *International Journal of Agricultural & Biological Engineering* 2(3): 1−16.

Lu. W., A. Song, J. Cai, H. Sun, and X. Chen. 2011. Structural design and kinematics algorithm research for orange harvesting robot . *Journal of Southeast University, (Natural Science Edition)* 41(1): 95-100.

Luo, X.Y., D.S. Jayas, and S.J. Symons. 1999. Identification of damaged kernels in wheat using a color machine vision system. *Journal of Cereal Science* 30(1):49-59.

Maghirang, E.B., F.E. Dowell, J.E. Baker, and J.E. Throne. 2003. Automated detection of single wheat kernels containing live or dead insect using near-infrared reflectance spectroscopy. *Transactions of the ASAE* 46(4): 1277-1282.

Mahesh, S., A. Manickavasagan, D.S. Jayas, J. Paliwal, and N.D.G. White. 2008. Feasibility of near-infrared hyperspectral imaging to differentiate Canadian wheat classes. *Biosystems Engineering* 101: 50-57.

Majumdar, S. and D.S. Jayas. 2000a. Classification of cereal grains using machine vision: I. Morphology models. *Transactions of the ASAE* 43(6): 1669-1675.

Majumdar, S. and D.S. Jayas. 2000b. Classification of cereal grains using machine vision: II. Color models. *Transactions of the ASAE* 43(6): 1677-1680.

Majumdar, S. and D.S. Jayas. 2000c. Classification of cereal grains using machine vision: III. Texture models. *Transactions of the ASAE* 43(6): 1681-1687.

Majumdar, S. and D.S. Jayas. 2000d. Classification of cereal grains using machine vision: IV. Morphology, color, and texture models. *Transactions of the ASAE* 43(6): 1689-1694.

Manickavasagan, A., D.S. Jayas, and N.D.G. White. 2008. Thermal imaging to detect infestation by *Cryptolestes ferrugineus* inside wheat kernels. *Journal of Stored Products Research* 44: 186-192.

Misimi, E., U. Erikson, and A. Skavhaug. 2008. Quality grading of atlantic salmon (Salmosalar) by computer vision. *Journal of Food Science* 73(5): E211–E217

Monta, M., N. Kondo, and Y. Shibano. 1995. Agricultural robot in grape production system. In *Proceedings of the IEEE International Conference on Robotics and Automation,* 2504-2509. Aichi, Japan. August 5-10.

Nielsen, K.M., P. Andersen, T.S. Pedersen, T. Bak, and D. Nielsen. 2002. Control of an autonomous vehicle for registration of weed and crop in precision agriculture. In *Proceedings of the IEEE International Conference on Control Applications,* 2: 909-914. Glasgow, Scotland. September 18-20.

Ollis, M. and A. Stentz. 1997. Vision-based perception for an automated harvester. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robotic Systems,* 1838-1844. Grenoble, France. September 7-11.

Paliwal, J., S.J. Symons, and C. Karunakaran. 2004. Insect species and infestation level determination in stored wheat using near-infrared spectroscopy. *Canadian Biosystems Engineering* 46: 7.17-7.24.

Patel, V.C., R.W. McClendon, and J.W. Goodrum. 1993. Egg defect detection using computer vision and neural networks. ASAE Paper No. 933051, St. Joseph, MI: ASAE.

Patel, V.C., R.W. McClendon, and J.W. Goodrum. 1995. Detection of cracks in eggs using color computer vision and artificial neural networks. ASAE Paper No. 953258, St. Joseph, MI: ASAE.

Paulsen, M.R. and W.F. McClure. 1986. Illumination for computer vision systems. *Transactions of the ASAE* 29(5): 1398-1404.

Recce, M., J. Taylor, A. Plebe, and G. Tropiano. 1996. Vision and neural control for an orange harvesting robot. In *Proceedings of the IEEE International Workshop on Neural Networks for Identification, Control, Robotics, and Signal/Image processing,* 467-475. Venice, Italy. August 21-23.

Richey, J.H. and J.H. Richey. 1986. Tractorbot restructures farming. In *Proceedings of the Agri-Mation 2*, 111-118. St. Joseph, MI: ASAE.

Rosier, J.C., R. Snel, and E.J. Goedvolk. 1996. Automated harvesting of flowers and

cuttings. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics,* 3006-3008. New York, NY: IEEE Systems, Man, and Cybernetics Society.

Schempf, H. and T. Graham. 2002. Junior: A robot for outdoor container nurseries. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, (IROS '02),* 1:750-755 October 02-03.

Sistler, F.E. 1987. Robotics and intelligent machines in agriculture. *IEEE Journal of Robotics and Automation* 3(1): 3-6.

Sistler, F.E. 1990. Grading agricultural products with machine vision. In *Proceedings of the International Conference on Intelligent Robots and Systems,* 255-261. Ibaraki, Japan. July 3-6.

Slaughter, D.C. and R.C. Harrel. 1987. Color vision in robotic fruit harvesting. *Transactions of the ASAE* 30(4): 1144-1148.

Slaughter, D.C. and R.C. Harrel. 1989. Discriminating fruit for robotic harvest using color in natural outdoor scenes. *Transactions of the ASAE* 32(2): 757-763.

Strachan, N.J.C and P. Nesvadba. 1990. Fish species recognition by shape analysis of images. *Pattern Recognition Society* 23(5): 539-544.

Suzuki, H. and M. Minami. 2002. Fish catching by visual servoing and observed intelligence of the fish. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems,* 1: 287-292. Lausanne, Switzerland. October 2-4.

Takahashi, Y., J. Ogawa, and K. Saeki. 2001. Automatic tomato picking robot system with human interface using image processing. In *Proceedings of the 27th Annual Conference of the IEEE,* 1: 433-438. Denver, CO. November 29- December 3.

Toda, M., O.Kitani, T. Okamoto, and T. Torii. 1993. Studies on autonomous vehicles for agricultural robots. ASAE Paper No. 993091, St. Joseph, MI: ASAE.

Visen, N.S. 2002. Machine vision based grain handling system. Unpublished Ph.D. thesis. Winnipeg, MB: Department of Biosystems Engineering, University of Manitoba.

Wishna, S. 1999. Image processing for official inspection of rice brokens. ASAE Paper No. 993197, St. Joseph, MI: ASAE.

Appendix A

Ball Screw Selection Charts

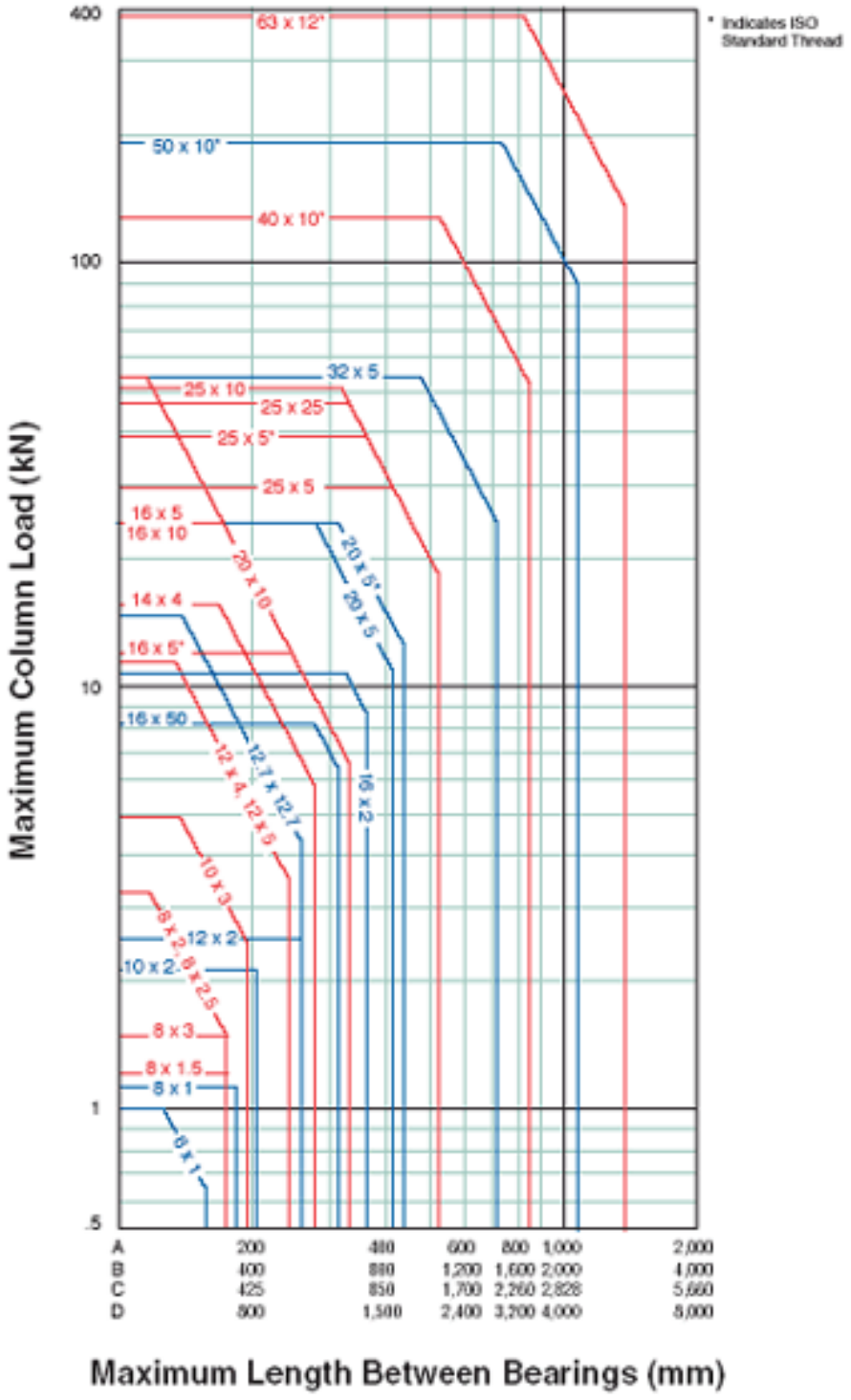


Figure 24. Critical Column Strength (Source: (Anonymous 2011g)

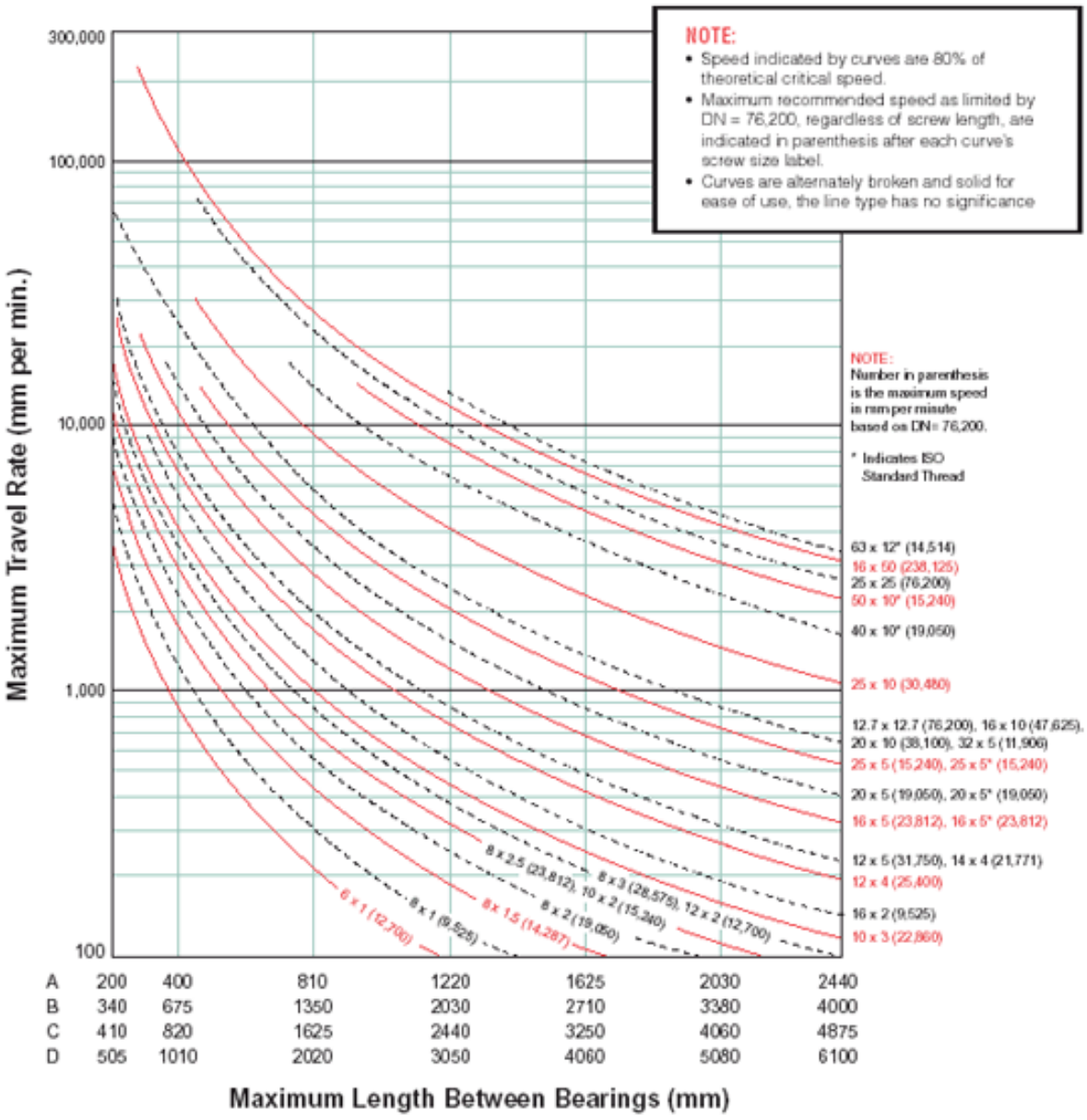


Figure 25. Critical speed chart (Source: Anonymous 2011g)

Appendix B

Programs

1. Vb.Net Program to Control Motor

```
Public Class Form1
    Inherits System.Windows.Forms.Form
    Private DaqBoard As MccDaq.MccBoard = New MccDaq.MccBoard(0)
    Public CheckBox As System.Windows.Forms.CheckBox()

#Region " Windows Form Designer generated code "

    Public Sub New()
        MyBase.New()

        'This call is required by the Windows Form Designer.
        InitializeComponent()

        'Add any initialization after the InitializeComponent() call

    End Sub

    'Form overrides dispose to clean up the component list.
    Protected Overloads Overrides Sub Dispose(ByVal disposing As
Boolean)
        If disposing Then
            If Not (components Is Nothing) Then
                components.Dispose()
            End If
        End If
        MyBase.Dispose(disposing)
    End Sub

    'Required by the Windows Form Designer
    Private components As System.ComponentModel.IContainer

    'NOTE: The following procedure is required by the Windows Form
Designer
    'It can be modified using the Windows Form Designer.
    'Do not modify it using the code editor.
    Friend WithEvents CheckBox1 As System.Windows.Forms.CheckBox
    Friend WithEvents CheckBox2 As System.Windows.Forms.CheckBox
    Friend WithEvents CheckBox3 As System.Windows.Forms.CheckBox
    Friend WithEvents CheckBox4 As System.Windows.Forms.CheckBox
    Friend WithEvents CheckBox5 As System.Windows.Forms.CheckBox
    Friend WithEvents CheckBox6 As System.Windows.Forms.CheckBox
    Friend WithEvents CheckBox7 As System.Windows.Forms.CheckBox
    Friend WithEvents CheckBox8 As System.Windows.Forms.CheckBox
    Friend WithEvents Label1 As System.Windows.Forms.Label
    Friend WithEvents Label2 As System.Windows.Forms.Label
    Friend WithEvents Label3 As System.Windows.Forms.Label
    Friend WithEvents Label4 As System.Windows.Forms.Label
    Friend WithEvents Label5 As System.Windows.Forms.Label
    Friend WithEvents Label6 As System.Windows.Forms.Label
    Friend WithEvents Timer1 As System.Windows.Forms.Timer
    <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()
        Me.components = New System.ComponentModel.Container
        Me.CheckBox1 = New System.Windows.Forms.CheckBox
```

```
Me.CheckBox2 = New System.Windows.Forms.CheckBox
Me.CheckBox3 = New System.Windows.Forms.CheckBox
Me.CheckBox4 = New System.Windows.Forms.CheckBox
Me.CheckBox5 = New System.Windows.Forms.CheckBox
Me.CheckBox6 = New System.Windows.Forms.CheckBox
Me.CheckBox7 = New System.Windows.Forms.CheckBox
Me.CheckBox8 = New System.Windows.Forms.CheckBox
Me.Label1 = New System.Windows.Forms.Label
Me.Label2 = New System.Windows.Forms.Label
Me.Label3 = New System.Windows.Forms.Label
Me.Label4 = New System.Windows.Forms.Label
Me.Label5 = New System.Windows.Forms.Label
Me.Label6 = New System.Windows.Forms.Label
Me.Timer1 = New System.Windows.Forms.Timer(Me.components)
Me.SuspendLayout()
'
'CheckBox1
'
Me.CheckBox1.Location = New System.Drawing.Point(104, 72)
Me.CheckBox1.Name = "CheckBox1"
Me.CheckBox1.Size = New System.Drawing.Size(88, 16)
Me.CheckBox1.TabIndex = 0
Me.CheckBox1.Text = "CheckBox1"
'
'CheckBox2
'
Me.CheckBox2.Location = New System.Drawing.Point(200, 72)
Me.CheckBox2.Name = "CheckBox2"
Me.CheckBox2.Size = New System.Drawing.Size(88, 16)
Me.CheckBox2.TabIndex = 1
Me.CheckBox2.Text = "CheckBox2"
'
'CheckBox3
'
Me.CheckBox3.Location = New System.Drawing.Point(104, 96)
Me.CheckBox3.Name = "CheckBox3"
Me.CheckBox3.Size = New System.Drawing.Size(88, 16)
Me.CheckBox3.TabIndex = 2
Me.CheckBox3.Text = "CheckBox3"
'
'CheckBox4
'
Me.CheckBox4.Location = New System.Drawing.Point(200, 96)
Me.CheckBox4.Name = "CheckBox4"
Me.CheckBox4.Size = New System.Drawing.Size(88, 16)
Me.CheckBox4.TabIndex = 3
Me.CheckBox4.Text = "CheckBox4"
'
'CheckBox5
'
Me.CheckBox5.Location = New System.Drawing.Point(104, 120)
Me.CheckBox5.Name = "CheckBox5"
Me.CheckBox5.Size = New System.Drawing.Size(88, 16)
Me.CheckBox5.TabIndex = 7
Me.CheckBox5.Text = "CheckBox5"
'
'CheckBox6
```

```
        '
        Me.CheckBox6.Location = New System.Drawing.Point(200, 120)
        Me.CheckBox6.Name = "CheckBox6"
        Me.CheckBox6.Size = New System.Drawing.Size(88, 16)
        Me.CheckBox6.TabIndex = 6
        Me.CheckBox6.Text = "CheckBox6"
        '
        'CheckBox7
        '
        Me.CheckBox7.Location = New System.Drawing.Point(104, 144)
        Me.CheckBox7.Name = "CheckBox7"
        Me.CheckBox7.Size = New System.Drawing.Size(88, 16)
        Me.CheckBox7.TabIndex = 5
        Me.CheckBox7.Text = "CheckBox7"
        '
        'CheckBox8
        '
        Me.CheckBox8.Location = New System.Drawing.Point(200, 144)
        Me.CheckBox8.Name = "CheckBox8"
        Me.CheckBox8.Size = New System.Drawing.Size(88, 16)
        Me.CheckBox8.TabIndex = 4
        Me.CheckBox8.Text = "CheckBox8"
        '
        'Label1
        '
        Me.Label1.Location = New System.Drawing.Point(128, 40)
        Me.Label1.Name = "Label1"
        Me.Label1.Size = New System.Drawing.Size(32, 24)
        Me.Label1.TabIndex = 8
        Me.Label1.Text = "CW"
        Me.Label1.TextAlign =
System.Drawing.ContentAlignment.MiddleCenter
        '
        'Label2
        '
        Me.Label2.Location = New System.Drawing.Point(224, 40)
        Me.Label2.Name = "Label2"
        Me.Label2.Size = New System.Drawing.Size(32, 24)
        Me.Label2.TabIndex = 9
        Me.Label2.Text = "CCW"
        Me.Label2.TextAlign =
System.Drawing.ContentAlignment.MiddleCenter
        '
        'Label3
        '
        Me.Label3.Location = New System.Drawing.Point(24, 72)
        Me.Label3.Name = "Label3"
        Me.Label3.Size = New System.Drawing.Size(56, 24)
        Me.Label3.TabIndex = 10
        Me.Label3.Text = "X-Axis"
        Me.Label3.TextAlign =
System.Drawing.ContentAlignment.MiddleCenter
        '
        'Label4
        '
        Me.Label4.Location = New System.Drawing.Point(24, 96)
        Me.Label4.Name = "Label4"
```

```
        Me.Label4.Size = New System.Drawing.Size(56, 24)
        Me.Label4.TabIndex = 11
        Me.Label4.Text = "Y-Axis"
        Me.Label4.TextAlign =
System.Drawing.ContentAlignment.MiddleCenter
        '
        'Label5
        '
        Me.Label5.Location = New System.Drawing.Point(24, 120)
        Me.Label5.Name = "Label5"
        Me.Label5.Size = New System.Drawing.Size(56, 24)
        Me.Label5.TabIndex = 12
        Me.Label5.Text = "Z-Axis"
        Me.Label5.TextAlign =
System.Drawing.ContentAlignment.MiddleCenter
        '
        'Label6
        '
        Me.Label6.Location = New System.Drawing.Point(24, 144)
        Me.Label6.Name = "Label6"
        Me.Label6.Size = New System.Drawing.Size(56, 24)
        Me.Label6.TabIndex = 13
        Me.Label6.Text = "Hopper"
        Me.Label6.TextAlign =
System.Drawing.ContentAlignment.MiddleCenter
        '
        'Timer1
        '
        Me.Timer1.Enabled = True
        Me.Timer1.Interval = 1000
        '
        'Form1
        '
        Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
        Me.ClientSize = New System.Drawing.Size(344, 266)
        Me.Controls.Add(Me.Label6)
        Me.Controls.Add(Me.Label5)
        Me.Controls.Add(Me.Label4)
        Me.Controls.Add(Me.Label3)
        Me.Controls.Add(Me.Label2)
        Me.Controls.Add(Me.Label1)
        Me.Controls.Add(Me.CheckBox5)
        Me.Controls.Add(Me.CheckBox6)
        Me.Controls.Add(Me.CheckBox7)
        Me.Controls.Add(Me.CheckBox8)
        Me.Controls.Add(Me.CheckBox4)
        Me.Controls.Add(Me.CheckBox3)
        Me.Controls.Add(Me.CheckBox2)
        Me.Controls.Add(Me.CheckBox1)
        Me.Name = "Form1"
        Me.RightToLeft = System.Windows.Forms.RightToLeft.No
        Me.Text = "Motor Control"
        Me.ResumeLayout(False)

    End Sub

#End Region
```

```
    Private Sub CheckBox1_Checkstatechanged(ByVal sender As Object,
ByVal e As System.EventArgs) Handles CheckBox1.CheckStateChanged
        Dim PortNum1 As MccDaq.DigitalPortType =
MccDaq.DigitalPortType.FirstPortA
            Dim ULStat1 As MccDaq.ErrorInfo
            Dim PortType1 As MccDaq.DigitalPortType
            Dim DataValue1 As UInt16
        'DataValue1 = Convert.ToUInt16(254)
            ULStat1 =
DaqBoard.DConfigPort(MccDaq.DigitalPortType.FirstPortA,
MccDaq.DigitalPortDirection.DigitalOut)
            PortType1 = PortNum1
        If CheckBox1.Checked = True Then
            CheckBox2.Checked = False
            'CheckBox3.Checked = False
            'CheckBox4.Checked = False
            'CheckBox5.Checked = False
            'CheckBox6.Checked = False
            'CheckBox7.Checked = False
            'CheckBox8.Checked = False
            DataValue1 = Convert.ToUInt16(254)
            ULStat1 = DaqBoard.DOut(PortType1, DataValue1)

        Else
            CheckBox1.Checked = False
            DataValue1 = Convert.ToUInt16(255)
            ULStat1 = DaqBoard.DOut(PortType1, DataValue1)
        End If


    End Sub
    Private Sub CheckBox2_Checkstatechanged(ByVal sender As Object,
ByVal e As System.EventArgs) Handles CheckBox2.CheckStateChanged
        Dim PortNum1 As MccDaq.DigitalPortType =
MccDaq.DigitalPortType.FirstPortA
        Dim ULStat1 As MccDaq.ErrorInfo
        Dim PortType1 As MccDaq.DigitalPortType
        Dim DataValue1 As UInt16
        'DataValue1 = Convert.ToUInt16(253)
        ULStat1 =
DaqBoard.DConfigPort(MccDaq.DigitalPortType.FirstPortA,
MccDaq.DigitalPortDirection.DigitalOut)
        PortType1 = PortNum1

        If CheckBox2.Checked = True Then

            CheckBox1.Checked = False
            'CheckBox3.Checked = False
            'CheckBox4.Checked = False
            'CheckBox5.Checked = False
            'CheckBox6.Checked = False
            'CheckBox7.Checked = False
            'CheckBox8.Checked = False
            DataValue1 = Convert.ToUInt16(253)
            ULStat1 = DaqBoard.DOut(PortType1, DataValue1)
        Else
            DataValue1 = Convert.ToUInt16(255)
```

```
            ULStat1 = DaqBoard.DOut(PortType1, DataValue1)
        End If

    End Sub
    Private Sub CheckBox3_Checkstatechanged(ByVal sender As Object,
ByVal e As System.EventArgs) Handles CheckBox3.CheckStateChanged
        Dim PortNum1 As MccDaq.DigitalPortType =
MccDaq.DigitalPortType.FirstPortA
        Dim ULStat1 As MccDaq.ErrorInfo
        Dim PortType1 As MccDaq.DigitalPortType
        Dim DataValue1 As UInt16
        'DataValue1 = Convert.ToUInt16(251)
        ULStat1 =
DaqBoard.DConfigPort(MccDaq.DigitalPortType.FirstPortA,
MccDaq.DigitalPortDirection.DigitalOut)
        PortType1 = PortNum1

        If CheckBox3.Checked = True Then
            'CheckBox1.Checked = False
            'CheckBox2.Checked = False
            CheckBox4.Checked = False
            'CheckBox5.Checked = False
            'CheckBox6.Checked = False
            'CheckBox7.Checked = False
            'CheckBox8.Checked = False
            DataValue1 = Convert.ToUInt16(251)
            ULStat1 = DaqBoard.DOut(PortType1, DataValue1)
        Else
            DataValue1 = Convert.ToUInt16(255)
            ULStat1 = DaqBoard.DOut(PortType1, DataValue1)
        End If



    End Sub
    Private Sub CheckBox4_Checkstatechanged(ByVal sender As Object,
ByVal e As System.EventArgs) Handles CheckBox4.CheckStateChanged

        Dim PortNum1 As MccDaq.DigitalPortType =
MccDaq.DigitalPortType.FirstPortA
        Dim ULStat1 As MccDaq.ErrorInfo
        Dim PortType1 As MccDaq.DigitalPortType
        Dim DataValue1 As UInt16
        DataValue1 = Convert.ToUInt16(247)
        ULStat1 =
DaqBoard.DConfigPort(MccDaq.DigitalPortType.FirstPortA,
MccDaq.DigitalPortDirection.DigitalOut)
        PortType1 = PortNum1

        If CheckBox4.Checked = True Then
            'CheckBox1.Checked = False
            'CheckBox2.Checked = False
            CheckBox3.Checked = False
            'CheckBox5.Checked = False
            'CheckBox6.Checked = False
            'CheckBox7.Checked = False
```

```
            'CheckBox8.Checked = False
            DataValue1 = Convert.ToUInt16(247)
            ULStat1 = DaqBoard.DOut(PortType1, DataValue1)
        Else
            DataValue1 = Convert.ToUInt16(255)
            ULStat1 = DaqBoard.DOut(PortType1, DataValue1)
        End If



    End Sub
    Private Sub CheckBox5_Checkstatechanged(ByVal sender As Object,
ByVal e As System.EventArgs) Handles CheckBox5.CheckStateChanged

        Dim PortNum1 As MccDaq.DigitalPortType =
MccDaq.DigitalPortType.FirstPortA
        Dim ULStat1 As MccDaq.ErrorInfo
        Dim PortType1 As MccDaq.DigitalPortType
        Dim DataValue1 As UInt16
        'DataValue1 = Convert.ToUInt16(239)
        ULStat1 =
DaqBoard.DConfigPort(MccDaq.DigitalPortType.FirstPortA,
MccDaq.DigitalPortDirection.DigitalOut)
        PortType1 = PortNum1

        If CheckBox5.Checked = True Then

            'CheckBox1.Checked = False
            'CheckBox3.Checked = False
            'CheckBox4.Checked = False
            'CheckBox2.Checked = False
            CheckBox6.Checked = False
            'CheckBox7.Checked = False
            'CheckBox8.Checked = False
            DataValue1 = Convert.ToUInt16(239)
            ULStat1 = DaqBoard.DOut(PortType1, DataValue1)
        Else
            DataValue1 = Convert.ToUInt16(255)
            ULStat1 = DaqBoard.DOut(PortType1, DataValue1)
        End If



    End Sub
    Private Sub CheckBox6_Checkstatechanged(ByVal sender As Object,
ByVal e As System.EventArgs) Handles CheckBox6.CheckStateChanged
        Dim PortNum1 As MccDaq.DigitalPortType =
MccDaq.DigitalPortType.FirstPortA
        Dim ULStat1 As MccDaq.ErrorInfo
        Dim PortType1 As MccDaq.DigitalPortType
        Dim DataValue1 As UInt16
        'DataValue1 = Convert.ToUInt16(223)
        ULStat1 =
DaqBoard.DConfigPort(MccDaq.DigitalPortType.FirstPortA,
MccDaq.DigitalPortDirection.DigitalOut)
        PortType1 = PortNum1
```

```
        If CheckBox6.Checked = True Then

            'CheckBox1.Checked = False
            'CheckBox3.Checked = False
            'CheckBox4.Checked = False
            CheckBox5.Checked = False
            'CheckBox2.Checked = False
            'CheckBox7.Checked = False
            'CheckBox8.Checked = False
            DataValue1 = Convert.ToUInt16(223)
            ULStat1 = DaqBoard.DOut(PortType1, DataValue1)
        Else
            DataValue1 = Convert.ToUInt16(255)
            ULStat1 = DaqBoard.DOut(PortType1, DataValue1)
        End If



    End Sub
    Private Sub CheckBox7_Checkstatechanged(ByVal sender As Object,
ByVal e As System.EventArgs) Handles CheckBox7.CheckStateChanged
        Dim PortNum1 As MccDaq.DigitalPortType =
MccDaq.DigitalPortType.FirstPortA
        Dim ULStat1 As MccDaq.ErrorInfo
        Dim PortType1 As MccDaq.DigitalPortType
        Dim DataValue1 As UInt16
        'DataValue1 = Convert.ToUInt16(191)
        ULStat1 =
DaqBoard.DConfigPort(MccDaq.DigitalPortType.FirstPortA,
MccDaq.DigitalPortDirection.DigitalOut)
        PortType1 = PortNum1

        If CheckBox7.Checked = True Then
            'CheckBox1.Checked = False
            'CheckBox3.Checked = False
            'CheckBox4.Checked = False
            'CheckBox5.Checked = False
            'CheckBox6.Checked = False
            'CheckBox2.Checked = False
            CheckBox8.Checked = False
            DataValue1 = Convert.ToUInt16(191)
            ULStat1 = DaqBoard.DOut(PortType1, DataValue1)
        Else
            DataValue1 = Convert.ToUInt16(255)
            ULStat1 = DaqBoard.DOut(PortType1, DataValue1)
        End If


    End Sub
    Private Sub CheckBox8_Checkstatechanged(ByVal sender As Object,
ByVal e As System.EventArgs) Handles CheckBox8.CheckStateChanged
        Dim PortNum1 As MccDaq.DigitalPortType =
MccDaq.DigitalPortType.FirstPortA
        Dim ULStat1 As MccDaq.ErrorInfo
        Dim PortType1 As MccDaq.DigitalPortType
        Dim DataValue1 As UInt16
        'DataValue1 = Convert.ToUInt16(127)
```

```
        ULStat1 =
DaqBoard.DConfigPort(MccDaq.DigitalPortType.FirstPortA,
MccDaq.DigitalPortDirection.DigitalOut)
        PortType1 = PortNum1

        If CheckBox8.Checked = True Then

            'CheckBox1.Checked = False
            'CheckBox3.Checked = False
            'CheckBox4.Checked = False
            'CheckBox5.Checked = False
            'CheckBox6.Checked = False
            CheckBox7.Checked = False
            'CheckBox2.Checked = False
            DataValue1 = Convert.ToUInt16(127)
            ULStat1 = DaqBoard.DOut(PortType1, DataValue1)
        Else
            DataValue1 = Convert.ToUInt16(255)
            ULStat1 = DaqBoard.DOut(PortType1, DataValue1)
        End If


    End Sub
    Private Sub Form1_Closing(ByVal sender As Object, ByVal e As
System.ComponentModel.CancelEventArgs) Handles MyBase.Closing
        Dim ULStat As MccDaq.ErrorInfo
        Dim DataValue As UInt16
        Dim DataValue1 As UInt16
        Dim PortNum As MccDaq.DigitalPortType =
MccDaq.DigitalPortType.FirstPortB
        Dim PortNum1 As MccDaq.DigitalPortType =
MccDaq.DigitalPortType.FirstPortA
        Dim ULStat1 As MccDaq.ErrorInfo
        Dim I As Integer
        ULStat1 =
DaqBoard.DConfigPort(MccDaq.DigitalPortType.FirstPortA,
MccDaq.DigitalPortDirection.DigitalOut)
        DataValue1 = Convert.ToUInt16(255)
        DataValue = Convert.ToUInt16(0)

        ULStat = DaqBoard.DIn(PortNum, DataValue)
        ULStat1 = DaqBoard.DOut(PortNum1, DataValue1)
        If ULStat1.Value <> MccDaq.ErrorInfo.ErrorCode.NoErrors Then
Stop
        If ULStat.Value <> MccDaq.ErrorInfo.ErrorCode.NoErrors Then
Stop
        End
    End Sub
    Private Sub Form1_Load(ByVal sender As Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        Dim ULStat As MccDaq.ErrorInfo
        Dim ULStat1 As MccDaq.ErrorInfo
        Dim DataValue As UInt16
        Dim DataValue1 As UInt16
        Dim PortNum As MccDaq.DigitalPortType =
MccDaq.DigitalPortType.FirstPortB
```

```
        Dim PortNum1 As MccDaq.DigitalPortType =
MccDaq.DigitalPortType.FirstPortA
        Dim I As Integer

        ULStat =
DaqBoard.DConfigPort(MccDaq.DigitalPortType.FirstPortB,
MccDaq.DigitalPortDirection.DigitalIn)
        ULStat1 =
DaqBoard.DConfigPort(MccDaq.DigitalPortType.FirstPortA,
MccDaq.DigitalPortDirection.DigitalOut)
        DataValue = Convert.ToUInt16(0)
        DataValue1 = Convert.ToUInt16(255)
        ULStat = DaqBoard.DIn(PortNum, DataValue)
        ULStat1 = DaqBoard.DOut(PortNum1, DataValue1)
        'If ULStat1.Value <> MccDaq.ErrorInfo.ErrorCode.NoErrors Then
Stop
        'If ULStat.Value <> MccDaq.ErrorInfo.ErrorCode.NoErrors Then
Stop
    End Sub

    Private Sub Timer1_Tick(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Timer1.Tick
        Timer1.Stop()
        Dim ULStat As MccDaq.ErrorInfo
        Dim ULStat1 As MccDaq.ErrorInfo
        Dim BitValue As MccDaq.DigitalLogicState
        Dim BitNum As Integer 'Changed from Short to Integer
        Dim J As Integer
        Dim PortType As MccDaq.DigitalPortType
        Dim PortNum As MccDaq.DigitalPortType =
MccDaq.DigitalPortType.FirstPortB
        Dim Direction As MccDaq.DigitalPortDirection =
MccDaq.DigitalPortDirection.DigitalIn
        Dim DataValue As UInt16
        Dim I As Integer
        PortType = PortNum
        J = BitNum

        ULStat = DaqBoard.DIn(PortNum, DataValue)
        If (Convert.ToInt32(DataValue) And CInt((2 ^ 0))) <> 0 Then
            CheckBox1.Checked = False
            Me.CheckBox1.ForeColor = System.Drawing.Color.Red
        Else
            Me.CheckBox1.ForeColor = System.Drawing.Color.Black
        End If

        If (Convert.ToInt32(DataValue) And CInt((2 ^ 1))) <> 0 Then
            CheckBox2.Checked = False
            Me.CheckBox2.ForeColor = System.Drawing.Color.Red
        Else
            Me.CheckBox2.ForeColor = System.Drawing.Color.Black
        End If


        If (Convert.ToInt32(DataValue) And CInt((2 ^ 2))) <> 0 Then
            CheckBox3.Checked = False
            Me.CheckBox3.ForeColor = System.Drawing.Color.Red
```

```
        Else
            Me.CheckBox3.ForeColor = System.Drawing.Color.Black
        End If


        If (Convert.ToInt32(DataValue) And CInt((2 ^ 3))) <> 0 Then
            CheckBox4.Checked = False
            Me.CheckBox4.ForeColor = System.Drawing.Color.Red
        Else
            Me.CheckBox4.ForeColor = System.Drawing.Color.Black
        End If


        If (Convert.ToInt32(DataValue) And CInt((2 ^ 4))) <> 0 Then
            CheckBox5.Checked = False
            Me.CheckBox5.ForeColor = System.Drawing.Color.Red
        Else
            Me.CheckBox5.ForeColor = System.Drawing.Color.Black
        End If


        If (Convert.ToInt32(DataValue) And CInt((2 ^ 5))) <> 0 Then
            CheckBox6.Checked = False
            Me.CheckBox6.ForeColor = System.Drawing.Color.Red
        Else
            Me.CheckBox6.ForeColor = System.Drawing.Color.Black
        End If


        If (Convert.ToInt32(DataValue) And CInt((2 ^ 6))) <> 0 Then
            CheckBox7.Checked = False
            Me.CheckBox7.ForeColor = System.Drawing.Color.Red
        Else
            Me.CheckBox7.ForeColor = System.Drawing.Color.Black
        End If


        If (Convert.ToInt32(DataValue) And CInt((2 ^ 7))) <> 0 Then
            CheckBox8.Checked = False
            Me.CheckBox8.ForeColor = System.Drawing.Color.Red
            'BitNum = 7
            'Dim PortNum1 As MccDaq.DigitalPortType =
MccDaq.DigitalPortType.FirstPortA
            'Dim ULStat1 As MccDaq.ErrorInfo
            'Dim Direction1 As MccDaq.DigitalPortDirection =
MccDaq.DigitalPortDirection.DigitalOut
            'Dim DataValue1 As UInt16
            'BitValue = MccDaq.DigitalLogicState.High
            'ULStat1 =
DaqBoard.DBitOut(MccDaq.DigitalPortType.FirstPortA, BitNum, BitValue)
        Else
            Me.CheckBox8.ForeColor = System.Drawing.Color.Black
        End If

        Timer1.Start()
    End Sub
End Class
```

2.a. Cor2.m (Program to Identify sprocket location)

```
clc;
clear all;
close all;
tic
dirname = 'C:\Documents and
Settings\Aravind\Desktop\Grain_o_bot\EndY';

out = 'C:\Documents and Settings\Aravind\Desktop\Grain_o_bot\EndY';

n= length(dirname);
index=strfind(dirname,'\');
index_len=length(index);
cur_path = ''; %Current Path
cur_path = strcat(cur_path,dirname(1:index(index_len)-1));
cur_dir  = ''; %Current directory
cur_dir  = strcat(cur_dir,dirname(index(index_len)+1:n));
files = dir(fullfile(cur_path,cur_dir));
temp_name = ' ';
temp_name = strcat(temp_name, cur_path,'\',out);
disp('Writing the data to the file');disp(temp_name);
fid = fopen(temp_name,'W');
ftr.name = 'blank';
ftr.num_points = 0;
ftr.locationx = 0;
ftr.locationy = 0;


ftr(2).name = 'blank';
ftr(2).num_points = 0;
ftr(2).locationx = 0;
ftr(2).locationy = 0;

A=imread('C:\Documents and
Settings\Aravind\Desktop\Grain_o_bot\Template\Template.tif');
C= A(25:145, 28:148);
F= edge(C, 'canny');
for i = 3:length(files) % the files start from the 3rd index

 disp ('Processing file:'); disp(files(i).name);
 [B, map]=imread('C:\Documents and
Settings\Aravind\Desktop\Grain_o_bot\EndY\1.tif');
%
img_size = size(A);
xx=median(median(B));
E = edge(B, 'canny');
D = real(ifft2(fft2(E) .* fft2(F,768,1024)));
thresh = max(D(:))-10;
H=(D > thresh);
img_size = size(A);
temp_size = size(F);

points = sum(sum(H));
```

```
location = zeros(points,2);

ii=1;
    for l=1:img_size(1)
        for m=1:img_size(2)
             if (H(l,m)==1)
                 location(ii,1)= round(l-(temp_size(1)/2));
                 location(ii,2)= round(m-(temp_size(2)/2));
%                  xxx=location(ii,1);

                 if (location(ii,1)< 1)
                     location(ii,1)= 1;
                     %disp('Invaild X indentification!');
                 end
                 if (location(ii,2)< 1)
                     location(ii,2)= 1;
                     %disp('Invaild Y indentification!');
                 end


                ii= ii+1;
             end
        end
    end


ftr(i).name=files(i).name;
ftr(i).num_points = points;


    for j=1:points
       temploc=location(j,:);
         oriDat=imread('C:\Documents and
Settings\Aravind\Desktop\Grain_o_bot\EndY\1.tif');
        IsPoint=eltest(oriDat,B,temploc);
        if IsPoint==1
       tempFinal=imread('C:\Documents and
Settings\Aravind\Desktop\Grain_o_bot\EndY\1.tif');
                        tempFinal(location(j,1),location(j,2))=255;
            x=location(j,1);
            y=location(j,2);
            for j=j+1:points
                H(location(j,1),location(j,2))=0;
            end
            break;
        else
            H(location(j,1),location(j,2))=0;
            x=0;
            y=0;
        end
    end

end
t=toc;
```

2.b. eltest.m (function for Cor2.m)

```
function RetValue=eltest(original,final,loc)
RetValue=0;
A1=original;A=final;
img_size = size(A);
l=loc(1,1); m=loc(1,2);
%%l=82; m=85;
% %   For l=1:img_size(1)
% %        For m=1:img_size(2)
% % % %              if (A(l,m)==255)
if l>65 && l<700 && m>65 && m<1100
%B= A1(l-50:l+50,m-50:m+50);
B= A1(l-60:l+60,m-60:m+60);
%Imshow(B);
C = imadjust(B,[0.3 0.7],[]);%Imshow(C);
%S = rangefilt(C);
T = edge(C,'canny');%imshow(T);
U = strel('disk', 4, 4);
V = imdilate(T,U); %imshow(V);
%%W = imfill(V,'holes');
%%imshow(W);
%%X = strel('square',3);
%%Y = imerode(V,X); imshow(Y);
Z = bwperim(V,8);
%imshow(Z);

% %
% % C= edge(B, 'canny');
% % Imshow(C)
C = bwareaopen(Z,100);
%imshow(C),title('mask');
[C,num] = bwlabel(C,4);
[sx sy]=size(C);
sq1=zeros(sx,sy);
sq2=zeros(sx,sy);
eli1=zeros(sx,sy);
eli2=zeros(sx,sy);
 sq1=(C==3);
 %Imshow(sq1);
 sq2=(C==4);
 %Imshow(sq2);
 eli1=(C==1);
 %imshow(eli1);
 eli2=(C==2);
%imshow(eli2);
try
eli1 = bwlabel(eli1,4);
eliprop1=regionprops(eli1,'Centroid');
eliprop3=regionprops(eli1,'MajorAxisLength');
eliprop4=regionprops(eli1,'MinorAxisLength');
eli2 = bwlabel(eli2,4);
eliprop2=regionprops(eli2,'Centroid');
eliprop5=regionprops(eli1,'MajorAxisLength');
eliprop6=regionprops(eli1,'MinorAxisLength');
```

```
sq1 = bwlabel(sq1,4);
sqprop1=regionprops(sq1,'Centroid');
sqprop3=regionprops(sq1,'MajorAxisLength');
sqprop4=regionprops(sq1,'MinorAxisLength');
sq2 = bwlabel(sq2,4);
sqprop2=regionprops(sq2,'Centroid');
sqprop5=regionprops(sq2,'MajorAxisLength');
sqprop6=regionprops(sq2,'MinorAxisLength');
if ((sqprop3.MajorAxisLength - sqprop4.MinorAxisLength)<5)
    if ((eliprop3.MajorAxisLength - eliprop4.MinorAxisLength)<40)
        if ((sqprop1.Centroid - eliprop1.Centroid)<15)
            oriCent=(sqprop1.Centroid+eliprop1.Centroid)/2;
            if((oriCent-[65 65])<15)%#ok<BDSCA,BDSCA>
                RetValue=1;
            end
        end
    end
end
if ((sqprop3.MajorAxisLength - sqprop4.MinorAxisLength)<5)
    if ((eliprop5.MajorAxisLength - eliprop6.MinorAxisLength)<40)
        if ((sqprop1.Centroid - eliprop2.Centroid)<15)
            oriCent=(sqprop1.Centroid+eliprop2.Centroid)/2;
            if((oriCent-[65 65])<15)%#ok<BDSCA,BDSCA>
               RetValue=1;
            end
        end
    end
end
if ((sqprop5.MajorAxisLength - sqprop6.MinorAxisLength)<5)
    if ((eliprop3.MajorAxisLength - eliprop4.MinorAxisLength)<40)
        if ((sqprop2.Centroid - eliprop1.Centroid)<15)
            oriCent=(sqprop2.Centroid+eliprop1.Centroid)/2;
            if((oriCent-[65 65])<15)%#ok<BDSCA,BDSCA>
                RetValue=1;
            end
        end
    end
end

if ((sqprop5.MajorAxisLength - sqprop6.MinorAxisLength)<5)
   if ((eliprop5.MajorAxisLength - eliprop6.MinorAxisLength)<40)
        if ((sqprop2.Centroid - eliprop2.Centroid)<15)
            oriCent=(sqprop2.Centroid+eliprop2.Centroid)/2;
            if((oriCent-[65 65])<15)%#ok<BDSCA,BDSCA>
                RetValue=1;
            end
        end
   end
end

catch
    Retvalue=0; %#ok<NASGU>
end
end
return;
```

2.c. Grain_O_Bot.exe

```
Public Class Grain_O_Bot
    Inherits System.Windows.Forms.Form
    Private DaqBoard As MccDaq.MccBoard = New MccDaq.MccBoard(0)

#Region " Windows Form Designer generated code "

    Public Sub New()
        MyBase.New()
        'This call is required by the Windows Form Designer.
        InitializeComponent()
        'Add any initialization after the InitializeComponent() call
        InitUL()
    End Sub

    'Form overrides dispose to clean up the component list.
    Protected Overloads Overrides Sub Dispose(ByVal disposing As
Boolean)
        If disposing Then
            If Not (components Is Nothing) Then
                components.Dispose()
            End If
        End If
        MyBase.Dispose(disposing)
    End Sub

    'Required by the Windows Form Designer
    Private components As System.ComponentModel.IContainer
    'NOTE: The following procedure is required by the Windows Form
Designer
    'It can be modified using the Windows Form Designer.
    'Do not modify it using the code editor.
    Friend WithEvents Button1 As System.Windows.Forms.Button
    Friend WithEvents AxMDisplay1 As AxMatrox.ActiveMIL.AxMDisplay
    Friend WithEvents AxMGraphicContext1 As
AxMatrox.ActiveMIL.AxMGraphicContext
    Friend WithEvents AxMDigitizer1 As AxMatrox.ActiveMIL.AxMDigitizer
    Friend WithEvents AxMImage1 As AxMatrox.ActiveMIL.AxMImage
    Friend WithEvents AxMSystem1 As AxMatrox.ActiveMIL.AxMSystem
    Friend WithEvents AxMApplication1 As
AxMatrox.ActiveMIL.AxMApplication
    Friend WithEvents AxMImage3 As AxMatrox.ActiveMIL.AxMImage
    Friend WithEvents AxMImage7 As AxMatrox.ActiveMIL.AxMImage
    Friend WithEvents AxMImage6 As AxMatrox.ActiveMIL.AxMImage
    Friend WithEvents AxMGraphicContext4 As
AxMatrox.ActiveMIL.AxMGraphicContext
    Friend WithEvents AxMGraphicContext3 As
AxMatrox.ActiveMIL.AxMGraphicContext
    Friend WithEvents AxMGraphicContext2 As
AxMatrox.ActiveMIL.AxMGraphicContext
    Friend WithEvents AxMDigitizer2 As AxMatrox.ActiveMIL.AxMDigitizer
    Friend WithEvents AxMDisplay2 As AxMatrox.ActiveMIL.AxMDisplay
    Friend WithEvents AxMImage2 As AxMatrox.ActiveMIL.AxMImage
```

```
    Friend WithEvents AxMSystem2 As AxMatrox.ActiveMIL.AxMSystem
    Friend WithEvents AxMApplication2 As
AxMatrox.ActiveMIL.AxMApplication
    Friend WithEvents TextBox1 As System.Windows.Forms.TextBox
    Friend WithEvents Label1 As System.Windows.Forms.Label
    Friend WithEvents Button2 As System.Windows.Forms.Button
    Friend WithEvents Button3 As System.Windows.Forms.Button
    <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()
        Dim resources As System.Resources.ResourceManager = New
System.Resources.ResourceManager(GetType(Grain_O_Bot))
        Me.Button1 = New System.Windows.Forms.Button
        Me.AxMDisplay1 = New AxMatrox.ActiveMIL.AxMDisplay
        Me.AxMGraphicContext1 = New
AxMatrox.ActiveMIL.AxMGraphicContext
        Me.AxMDigitizer1 = New AxMatrox.ActiveMIL.AxMDigitizer
        Me.AxMImage1 = New AxMatrox.ActiveMIL.AxMImage
        Me.AxMSystem1 = New AxMatrox.ActiveMIL.AxMSystem
        Me.AxMApplication1 = New AxMatrox.ActiveMIL.AxMApplication
        Me.AxMImage3 = New AxMatrox.ActiveMIL.AxMImage
        Me.AxMImage7 = New AxMatrox.ActiveMIL.AxMImage
        Me.AxMImage6 = New AxMatrox.ActiveMIL.AxMImage
        Me.AxMGraphicContext4 = New
AxMatrox.ActiveMIL.AxMGraphicContext
        Me.AxMGraphicContext3 = New
AxMatrox.ActiveMIL.AxMGraphicContext
        Me.AxMGraphicContext2 = New
AxMatrox.ActiveMIL.AxMGraphicContext
        Me.AxMDigitizer2 = New AxMatrox.ActiveMIL.AxMDigitizer
        Me.AxMDisplay2 = New AxMatrox.ActiveMIL.AxMDisplay
        Me.AxMImage2 = New AxMatrox.ActiveMIL.AxMImage
        Me.AxMSystem2 = New AxMatrox.ActiveMIL.AxMSystem
        Me.AxMApplication2 = New AxMatrox.ActiveMIL.AxMApplication
        Me.TextBox1 = New System.Windows.Forms.TextBox
        Me.Label1 = New System.Windows.Forms.Label
        Me.Button2 = New System.Windows.Forms.Button
        Me.Button3 = New System.Windows.Forms.Button
        CType(Me.AxMDisplay1,
System.ComponentModel.ISupportInitialize).BeginInit()
        CType(Me.AxMGraphicContext1,
System.ComponentModel.ISupportInitialize).BeginInit()
        CType(Me.AxMDigitizer1,
System.ComponentModel.ISupportInitialize).BeginInit()
        CType(Me.AxMImage1,
System.ComponentModel.ISupportInitialize).BeginInit()
        CType(Me.AxMSystem1,
System.ComponentModel.ISupportInitialize).BeginInit()
        CType(Me.AxMApplication1,
System.ComponentModel.ISupportInitialize).BeginInit()
        CType(Me.AxMImage3,
System.ComponentModel.ISupportInitialize).BeginInit()
        CType(Me.AxMImage7,
System.ComponentModel.ISupportInitialize).BeginInit()
        CType(Me.AxMImage6,
System.ComponentModel.ISupportInitialize).BeginInit()
        CType(Me.AxMGraphicContext4,
System.ComponentModel.ISupportInitialize).BeginInit()
```

```
        CType(Me.AxMGraphicContext3,
System.ComponentModel.ISupportInitialize).BeginInit()
        CType(Me.AxMGraphicContext2,
System.ComponentModel.ISupportInitialize).BeginInit()
        CType(Me.AxMDigitizer2,
System.ComponentModel.ISupportInitialize).BeginInit()
        CType(Me.AxMDisplay2,
System.ComponentModel.ISupportInitialize).BeginInit()
        CType(Me.AxMImage2,
System.ComponentModel.ISupportInitialize).BeginInit()
        CType(Me.AxMSystem2,
System.ComponentModel.ISupportInitialize).BeginInit()
        CType(Me.AxMApplication2,
System.ComponentModel.ISupportInitialize).BeginInit()
        Me.SuspendLayout()
        '
        'Button1
        '
        Me.Button1.Location = New System.Drawing.Point(40, 136)
        Me.Button1.Name = "Button1"
        Me.Button1.TabIndex = 7
        Me.Button1.Text = "Start"
        '
        'AxMDisplay1
        '
        Me.AxMDisplay1.Enabled = True
        Me.AxMDisplay1.Location = New System.Drawing.Point(168, 32)
        Me.AxMDisplay1.Name = "AxMDisplay1"
        Me.AxMDisplay1.OcxState =
CType(resources.GetObject("AxMDisplay1.OcxState"),
System.Windows.Forms.AxHost.State)
        Me.AxMDisplay1.Size = New System.Drawing.Size(768, 400)
        Me.AxMDisplay1.TabIndex = 13
        Me.AxMDisplay1.Visible = False
        '
        'AxMGraphicContext1
        '
        Me.AxMGraphicContext1.Enabled = True
        Me.AxMGraphicContext1.Location = New System.Drawing.Point(256,
0)
        Me.AxMGraphicContext1.Name = "AxMGraphicContext1"
        Me.AxMGraphicContext1.OcxState =
CType(resources.GetObject("AxMGraphicContext1.OcxState"),
System.Windows.Forms.AxHost.State)
        Me.AxMGraphicContext1.Size = New System.Drawing.Size(32, 32)
        Me.AxMGraphicContext1.TabIndex = 12
        Me.AxMGraphicContext1.Visible = False
        '
        'AxMDigitizer1
        '
        Me.AxMDigitizer1.Enabled = True
        Me.AxMDigitizer1.Location = New System.Drawing.Point(288, 0)
        Me.AxMDigitizer1.Name = "AxMDigitizer1"
        Me.AxMDigitizer1.OcxState =
CType(resources.GetObject("AxMDigitizer1.OcxState"),
System.Windows.Forms.AxHost.State)
        Me.AxMDigitizer1.Size = New System.Drawing.Size(32, 32)
```

```
        Me.AxMDigitizer1.TabIndex = 11
        Me.AxMDigitizer1.Visible = False
        '
        'AxMImage1
        '
        Me.AxMImage1.Enabled = True
        Me.AxMImage1.Location = New System.Drawing.Point(224, 0)
        Me.AxMImage1.Name = "AxMImage1"
        Me.AxMImage1.OcxState =
CType(resources.GetObject("AxMImage1.OcxState"),
System.Windows.Forms.AxHost.State)
        Me.AxMImage1.Size = New System.Drawing.Size(32, 32)
        Me.AxMImage1.TabIndex = 10
        Me.AxMImage1.Visible = False
        '
        'AxMSystem1
        '
        Me.AxMSystem1.Enabled = True
        Me.AxMSystem1.Location = New System.Drawing.Point(192, 0)
        Me.AxMSystem1.Name = "AxMSystem1"
        Me.AxMSystem1.OcxState =
CType(resources.GetObject("AxMSystem1.OcxState"),
System.Windows.Forms.AxHost.State)
        Me.AxMSystem1.Size = New System.Drawing.Size(32, 32)
        Me.AxMSystem1.TabIndex = 9
        '
        'AxMApplication1
        '
        Me.AxMApplication1.Enabled = True
        Me.AxMApplication1.Location = New System.Drawing.Point(160, 0)
        Me.AxMApplication1.Name = "AxMApplication1"
        Me.AxMApplication1.OcxState =
CType(resources.GetObject("AxMApplication1.OcxState"),
System.Windows.Forms.AxHost.State)
        Me.AxMApplication1.Size = New System.Drawing.Size(32, 32)
        Me.AxMApplication1.TabIndex = 8
        Me.AxMApplication1.Visible = False
        '
        'AxMImage3
        '
        Me.AxMImage3.Enabled = True
        Me.AxMImage3.Location = New System.Drawing.Point(688, 16)
        Me.AxMImage3.Name = "AxMImage3"
        Me.AxMImage3.OcxState =
CType(resources.GetObject("AxMImage3.OcxState"),
System.Windows.Forms.AxHost.State)
        Me.AxMImage3.Size = New System.Drawing.Size(32, 32)
        Me.AxMImage3.TabIndex = 31
        Me.AxMImage3.Visible = False
        '
        'AxMImage7
        '
        Me.AxMImage7.Enabled = True
        Me.AxMImage7.Location = New System.Drawing.Point(752, 16)
        Me.AxMImage7.Name = "AxMImage7"
```

```
        Me.AxMImage7.OcxState =
CType(resources.GetObject("AxMImage7.OcxState"),
System.Windows.Forms.AxHost.State)
        Me.AxMImage7.Size = New System.Drawing.Size(32, 32)
        Me.AxMImage7.TabIndex = 30
        Me.AxMImage7.Visible = False
        '
        'AxMImage6
        '
        Me.AxMImage6.Enabled = True
        Me.AxMImage6.Location = New System.Drawing.Point(720, 16)
        Me.AxMImage6.Name = "AxMImage6"
        Me.AxMImage6.OcxState =
CType(resources.GetObject("AxMImage6.OcxState"),
System.Windows.Forms.AxHost.State)
        Me.AxMImage6.Size = New System.Drawing.Size(32, 32)
        Me.AxMImage6.TabIndex = 29
        Me.AxMImage6.Visible = False
        '
        'AxMGraphicContext4
        '
        Me.AxMGraphicContext4.Enabled = True
        Me.AxMGraphicContext4.Location = New System.Drawing.Point(848,
16)
        Me.AxMGraphicContext4.Name = "AxMGraphicContext4"
        Me.AxMGraphicContext4.OcxState =
CType(resources.GetObject("AxMGraphicContext4.OcxState"),
System.Windows.Forms.AxHost.State)
        Me.AxMGraphicContext4.Size = New System.Drawing.Size(32, 32)
        Me.AxMGraphicContext4.TabIndex = 28
        Me.AxMGraphicContext4.Visible = False
        '
        'AxMGraphicContext3
        '
        Me.AxMGraphicContext3.Enabled = True
        Me.AxMGraphicContext3.Location = New System.Drawing.Point(816,
16)
        Me.AxMGraphicContext3.Name = "AxMGraphicContext3"
        Me.AxMGraphicContext3.OcxState =
CType(resources.GetObject("AxMGraphicContext3.OcxState"),
System.Windows.Forms.AxHost.State)
        Me.AxMGraphicContext3.Size = New System.Drawing.Size(32, 32)
        Me.AxMGraphicContext3.TabIndex = 27
        Me.AxMGraphicContext3.Visible = False
        '
        'AxMGraphicContext2
        '
        Me.AxMGraphicContext2.Enabled = True
        Me.AxMGraphicContext2.Location = New System.Drawing.Point(784,
16)
        Me.AxMGraphicContext2.Name = "AxMGraphicContext2"
        Me.AxMGraphicContext2.OcxState =
CType(resources.GetObject("AxMGraphicContext2.OcxState"),
System.Windows.Forms.AxHost.State)
        Me.AxMGraphicContext2.Size = New System.Drawing.Size(32, 32)
        Me.AxMGraphicContext2.TabIndex = 26
        Me.AxMGraphicContext2.Visible = False
```

```
        '
        'AxMDigitizer2
        '
        Me.AxMDigitizer2.Enabled = True
        Me.AxMDigitizer2.Location = New System.Drawing.Point(952, 16)
        Me.AxMDigitizer2.Name = "AxMDigitizer2"
        Me.AxMDigitizer2.OcxState =
CType(resources.GetObject("AxMDigitizer2.OcxState"),
System.Windows.Forms.AxHost.State)
        Me.AxMDigitizer2.Size = New System.Drawing.Size(32, 32)
        Me.AxMDigitizer2.TabIndex = 25
        Me.AxMDigitizer2.Visible = False
        '
        'AxMDisplay2
        '
        Me.AxMDisplay2.Enabled = True
        Me.AxMDisplay2.Location = New System.Drawing.Point(304, 56)
        Me.AxMDisplay2.Name = "AxMDisplay2"
        Me.AxMDisplay2.OcxState =
CType(resources.GetObject("AxMDisplay2.OcxState"),
System.Windows.Forms.AxHost.State)
        Me.AxMDisplay2.Size = New System.Drawing.Size(672, 328)
        Me.AxMDisplay2.TabIndex = 24
        Me.AxMDisplay2.Visible = False
        '
        'AxMImage2
        '
        Me.AxMImage2.Enabled = True
        Me.AxMImage2.Location = New System.Drawing.Point(656, 16)
        Me.AxMImage2.Name = "AxMImage2"
        Me.AxMImage2.OcxState =
CType(resources.GetObject("AxMImage2.OcxState"),
System.Windows.Forms.AxHost.State)
        Me.AxMImage2.Size = New System.Drawing.Size(32, 32)
        Me.AxMImage2.TabIndex = 23
        Me.AxMImage2.Visible = False
        '
        'AxMSystem2
        '
        Me.AxMSystem2.Enabled = True
        Me.AxMSystem2.Location = New System.Drawing.Point(632, 16)
        Me.AxMSystem2.Name = "AxMSystem2"
        Me.AxMSystem2.OcxState =
CType(resources.GetObject("AxMSystem2.OcxState"),
System.Windows.Forms.AxHost.State)
        Me.AxMSystem2.Size = New System.Drawing.Size(32, 32)
        Me.AxMSystem2.TabIndex = 22
        '
        'AxMApplication2
        '
        Me.AxMApplication2.Enabled = True
        Me.AxMApplication2.Location = New System.Drawing.Point(600, 16)
        Me.AxMApplication2.Name = "AxMApplication2"
        Me.AxMApplication2.OcxState =
CType(resources.GetObject("AxMApplication2.OcxState"),
System.Windows.Forms.AxHost.State)
        Me.AxMApplication2.Size = New System.Drawing.Size(32, 32)
```

```
Me.AxMApplication2.TabIndex = 21
Me.AxMApplication2.Visible = False
'
'TextBox1
'
Me.TextBox1.Location = New System.Drawing.Point(232, 160)
Me.TextBox1.Name = "TextBox1"
Me.TextBox1.Size = New System.Drawing.Size(264, 20)
Me.TextBox1.TabIndex = 32
Me.TextBox1.Text = "TextBox1"
'
'Label1
'
Me.Label1.Location = New System.Drawing.Point(232, 120)
Me.Label1.Name = "Label1"
Me.Label1.Size = New System.Drawing.Size(176, 24)
Me.Label1.TabIndex = 33
Me.Label1.Text = "The Grain Identified is"
'
'Button2
'
Me.Button2.Location = New System.Drawing.Point(224, 208)
Me.Button2.Name = "Button2"
Me.Button2.Size = New System.Drawing.Size(96, 32)
Me.Button2.TabIndex = 34
Me.Button2.Text = "Open"
'
'Button3
'
Me.Button3.Location = New System.Drawing.Point(400, 208)
Me.Button3.Name = "Button3"
Me.Button3.Size = New System.Drawing.Size(88, 32)
Me.Button3.TabIndex = 35
Me.Button3.Text = "Do Not Open"
'
'Grain_O_Bot
'
Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
Me.ClientSize = New System.Drawing.Size(1008, 454)
Me.Controls.Add(Me.Button3)
Me.Controls.Add(Me.Button2)
Me.Controls.Add(Me.Label1)
Me.Controls.Add(Me.TextBox1)
Me.Controls.Add(Me.AxMImage3)
Me.Controls.Add(Me.AxMImage7)
Me.Controls.Add(Me.AxMImage6)
Me.Controls.Add(Me.AxMGraphicContext4)
Me.Controls.Add(Me.AxMGraphicContext3)
Me.Controls.Add(Me.AxMGraphicContext2)
Me.Controls.Add(Me.AxMDigitizer2)
Me.Controls.Add(Me.AxMDisplay2)
Me.Controls.Add(Me.AxMImage2)
Me.Controls.Add(Me.AxMSystem2)
Me.Controls.Add(Me.AxMApplication2)
Me.Controls.Add(Me.AxMDisplay1)
Me.Controls.Add(Me.AxMGraphicContext1)
Me.Controls.Add(Me.AxMDigitizer1)
```

```
        Me.Controls.Add(Me.AxMImage1)
        Me.Controls.Add(Me.AxMSystem1)
        Me.Controls.Add(Me.AxMApplication1)
        Me.Controls.Add(Me.Button1)
        Me.Name = "Grain_O_Bot"
        Me.Text = "Grain_O_Bot"
        CType(Me.AxMDisplay1,
System.ComponentModel.ISupportInitialize).EndInit()
        CType(Me.AxMGraphicContext1,
System.ComponentModel.ISupportInitialize).EndInit()
        CType(Me.AxMDigitizer1,
System.ComponentModel.ISupportInitialize).EndInit()
        CType(Me.AxMImage1,
System.ComponentModel.ISupportInitialize).EndInit()
        CType(Me.AxMSystem1,
System.ComponentModel.ISupportInitialize).EndInit()
        CType(Me.AxMApplication1,
System.ComponentModel.ISupportInitialize).EndInit()
        CType(Me.AxMImage3,
System.ComponentModel.ISupportInitialize).EndInit()
        CType(Me.AxMImage7,
System.ComponentModel.ISupportInitialize).EndInit()
        CType(Me.AxMImage6,
System.ComponentModel.ISupportInitialize).EndInit()
        CType(Me.AxMGraphicContext4,
System.ComponentModel.ISupportInitialize).EndInit()
        CType(Me.AxMGraphicContext3,
System.ComponentModel.ISupportInitialize).EndInit()
        CType(Me.AxMGraphicContext2,
System.ComponentModel.ISupportInitialize).EndInit()
        CType(Me.AxMDigitizer2,
System.ComponentModel.ISupportInitialize).EndInit()
        CType(Me.AxMDisplay2,
System.ComponentModel.ISupportInitialize).EndInit()
        CType(Me.AxMImage2,
System.ComponentModel.ISupportInitialize).EndInit()
        CType(Me.AxMSystem2,
System.ComponentModel.ISupportInitialize).EndInit()
        CType(Me.AxMApplication2,
System.ComponentModel.ISupportInitialize).EndInit()
        Me.ResumeLayout(False)

    End Sub

#End Region
#Region "Universal Library Initialization - Expand to change error
handling, etc."

    Private Sub InitUL()

        Dim ULStat As MccDaq.ErrorInfo
        'Initiate error handling
        '  activating error handling will trap errors like
        '  bad channel numbers and non-configured conditions.
        '  Parameters:
        '    MccDaq.ErrorReporting.PrintAll :all warnings and errors
encountered will be printed
```

```
        '    MccDaq.ErrorHandling.StopAll   :if any error is
encountered, the program will stop

        ULStat =
MccDaq.MccService.ErrHandling(MccDaq.ErrorReporting.PrintAll,
MccDaq.ErrorHandling.StopAll)

        If ULStat.Value <> MccDaq.ErrorInfo.ErrorCode.NoErrors Then
            Stop
        End If

    End Sub

#End Region

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        Dim MatLab As Object
        Dim Result, Result_1 As String
        Dim MReal(1, 3) As Double
        Dim MImag(1, 3) As Double
        Dim x As Object
        Dim y As Object
        Dim Gr_Id As Object
        Dim Image1 As Object

        'For Test commented on 17th Dec
        Dim ULStat As MccDaq.ErrorInfo
        Dim BoardNum As Integer
        BoardNum = 0
        DaqBoard = New MccDaq.MccBoard(BoardNum)
        Dim A, B, C, D, F, G, I, J, K, L, M, N As Integer

        Dim BitValue As MccDaq.DigitalLogicState
        Dim BitNum As Integer 'Changed from Short to Integer
        Dim PortType As MccDaq.DigitalPortType
        Dim PortNum As MccDaq.DigitalPortType =
MccDaq.DigitalPortType.FirstPortA
        Dim Direction As MccDaq.DigitalPortDirection =
MccDaq.DigitalPortDirection.DigitalOut
        Dim DataValue As UInt16
        PortType = PortNum
        ULStat =
DaqBoard.DConfigPort(MccDaq.DigitalPortType.FirstPortA,
MccDaq.DigitalPortDirection.DigitalOut)

        'Bring X to Base
        DataValue = Convert.ToUInt16(253)
        ULStat = DaqBoard.DOut(PortType, DataValue)
        If ULStat.Value <> MccDaq.ErrorInfo.ErrorCode.NoErrors Then
Stop
        System.Threading.Thread.Sleep(1000) 'Change to Length of X
        DataValue = Convert.ToUInt16(255)
        ULStat = DaqBoard.DOut(PortType, DataValue)
        If ULStat.Value <> MccDaq.ErrorInfo.ErrorCode.NoErrors Then
Stop
```

```
        'Bring Y to Base
        DataValue = Convert.ToUInt16(251)
        ULStat = DaqBoard.DOut(PortType, DataValue)
        If ULStat.Value <> MccDaq.ErrorInfo.ErrorCode.NoErrors Then
Stop
        System.Threading.Thread.Sleep(1000) 'Change to Length of Y
        DataValue = Convert.ToUInt16(255)
        ULStat = DaqBoard.DOut(PortType, DataValue)
        If ULStat.Value <> MccDaq.ErrorInfo.ErrorCode.NoErrors Then
Stop

        'Bring Z to Base
        DataValue = Convert.ToUInt16(223)
        ULStat = DaqBoard.DOut(PortType, DataValue)
        If ULStat.Value <> MccDaq.ErrorInfo.ErrorCode.NoErrors Then
Stop
        System.Threading.Thread.Sleep(1000) 'Change to Length of Z
        DataValue = Convert.ToUInt16(255)
        ULStat = DaqBoard.DOut(PortType, DataValue)
        If ULStat.Value <> MccDaq.ErrorInfo.ErrorCode.NoErrors Then
Stop

        'Bring Y to Base+View
        DataValue = Convert.ToUInt16(247)
        ULStat = DaqBoard.DOut(PortType, DataValue)
        If ULStat.Value <> MccDaq.ErrorInfo.ErrorCode.NoErrors Then
Stop
        System.Threading.Thread.Sleep(10000) 'Change to Length of Z up
        DataValue = Convert.ToUInt16(255)
        ULStat = DaqBoard.DOut(PortType, DataValue)
        If ULStat.Value <> MccDaq.ErrorInfo.ErrorCode.NoErrors Then
Stop

        'Grab(Image)
        'Till This Commented on 17th Dec
        Image1 = Me.AxMImage1
        AxMDigitizer1.Grab()
        Image1.Save("C:\Documents and
Settings\Aravind\Desktop\Grain_o_bot\EndY\1.tif")

        'Start This Commented on 17th Dec
        'Analyse with Matlab for the 1st time
        MatLab = CreateObject("Matlab.Application")
        Result = MatLab.Execute("C:\Documents and Settings\Aravind\My
Documents\MATLAB")
        Result = MatLab.Execute("cor2")
        x = MatLab.GetVariable("x", "base")
        y = MatLab.GetVariable("y", "base")
        'MsgBox(x & "," & y) 'Remove this later
        ''MatLab.Quit()

        'If Image is not found move grab and analyse
        For I = 1 To 2
            If x = 0 Then
                'Move
                DataValue = Convert.ToUInt16(254)
                ULStat = DaqBoard.DOut(PortType, DataValue)
```

```
                If ULStat.Value <> MccDaq.ErrorInfo.ErrorCode.NoErrors
Then Stop
                System.Threading.Thread.Sleep(60000) 'Change to Length
of half a frame
                DataValue = Convert.ToUInt16(255)
                ULStat = DaqBoard.DOut(PortType, DataValue)
                If ULStat.Value <> MccDaq.ErrorInfo.ErrorCode.NoErrors
Then Stop

                'Grab Image
                Image1 = Me.AxMImage1
                AxMDigitizer1.Grab()
                Image1.Save("C:\Documents and
Settings\Aravind\Desktop\Grain_o_bot\EndY\1.tif")

                'Analyze
                MatLab = CreateObject("Matlab.Application")
                Result = MatLab.Execute("C:\Documents and
Settings\Aravind\My Documents\MATLAB")
                Result = MatLab.Execute("cor2")
                x = MatLab.GetVariable("x", "base")
                y = MatLab.GetVariable("y", "base")
                'MsgBox(x & "," & y) ' Remove this later
                'MatLab.Quit()
            Else
                Exit For
            End If
        Next I

        'Readings for Y End
        A = 568
        B = 63

        'Aligning Idenified Image with base template
        F = A - (x + 60)
        D = B - (y - 60)
        'MsgBox(D & "," & F)
        If D > 0 Then
            D = ((Math.Abs(D) / 11) * 1000)
            DataValue = Convert.ToUInt16(254)
            ULStat = DaqBoard.DOut(PortType, DataValue)
            If ULStat.Value <> MccDaq.ErrorInfo.ErrorCode.NoErrors Then
Stop
            System.Threading.Thread.Sleep(D)
            DataValue = Convert.ToUInt16(255)
            ULStat = DaqBoard.DOut(PortType, DataValue)
            If ULStat.Value <> MccDaq.ErrorInfo.ErrorCode.NoErrors Then
Stop
        Else
            D = ((Math.Abs(D) / 11) * 1000)
            DataValue = Convert.ToUInt16(253)
            ULStat = DaqBoard.DOut(PortType, DataValue)
            If ULStat.Value <> MccDaq.ErrorInfo.ErrorCode.NoErrors Then
Stop
            System.Threading.Thread.Sleep(D)
            DataValue = Convert.ToUInt16(255)
            ULStat = DaqBoard.DOut(PortType, DataValue)
```

```
            If ULStat.Value <> MccDaq.ErrorInfo.ErrorCode.NoErrors Then
Stop
        End If

        If F > 0 Then
            F = ((Math.Abs(F) / 11.5) * 1000)
            DataValue = Convert.ToUInt16(247)
            ULStat = DaqBoard.DOut(PortType, DataValue)
            If ULStat.Value <> MccDaq.ErrorInfo.ErrorCode.NoErrors Then
Stop
            System.Threading.Thread.Sleep(F)
            DataValue = Convert.ToUInt16(255)
            ULStat = DaqBoard.DOut(PortType, DataValue)
            If ULStat.Value <> MccDaq.ErrorInfo.ErrorCode.NoErrors Then
Stop
        Else
            F = ((Math.Abs(F) / 11.5) * 1000)
            DataValue = Convert.ToUInt16(251)
            ULStat = DaqBoard.DOut(PortType, DataValue)
            If ULStat.Value <> MccDaq.ErrorInfo.ErrorCode.NoErrors Then
Stop
            System.Threading.Thread.Sleep(F)
            DataValue = Convert.ToUInt16(255)
            ULStat = DaqBoard.DOut(PortType, DataValue)
        End If

        'Move in Z till stop
        DataValue = Convert.ToUInt16(239) 'Change for Z (223)
        ULStat = DaqBoard.DOut(PortType, DataValue)
        If ULStat.Value <> MccDaq.ErrorInfo.ErrorCode.NoErrors Then
Stop
        System.Threading.Thread.Sleep(73000) 'Will engage (Check) '
Change Back to 73000
        DataValue = Convert.ToUInt16(255)
        ULStat = DaqBoard.DOut(PortType, DataValue)
        If ULStat.Value <> MccDaq.ErrorInfo.ErrorCode.NoErrors Then
Stop

        ''Move in Rot for a sec
        'DataValue = Convert.ToUInt16(191)
        'ULStat = DaqBoard.DOut(PortType, DataValue)
        'If ULStat.Value <> MccDaq.ErrorInfo.ErrorCode.NoErrors Then
Stop
        'System.Threading.Thread.Sleep(250) 'half a second change if
needed
        'DataValue = Convert.ToUInt16(255)
        'ULStat = DaqBoard.DOut(PortType, DataValue)
        'If ULStat.Value <> MccDaq.ErrorInfo.ErrorCode.NoErrors Then
Stop



        ''Move in Z for a sec and stop
        'DataValue = Convert.ToUInt16(239) 'Change for Z (223)
        'ULStat = DaqBoard.DOut(PortType, DataValue)
        'If ULStat.Value <> MccDaq.ErrorInfo.ErrorCode.NoErrors Then
Stop
```

```
        'System.Threading.Thread.Sleep(250) 'Will engage
        'DataValue = Convert.ToUInt16(255)
        'ULStat = DaqBoard.DOut(PortType, DataValue)
        'If ULStat.Value <> MccDaq.ErrorInfo.ErrorCode.NoErrors Then
Stop


        'Move in opp for a sec and move back *Assuming the sprocket is
closed*

        'Opening the sprocket

        DataValue = Convert.ToUInt16(191)
        ULStat = DaqBoard.DOut(PortType, DataValue)
        If ULStat.Value <> MccDaq.ErrorInfo.ErrorCode.NoErrors Then
Stop
        System.Threading.Thread.Sleep(1000) 'Will engage
        DataValue = Convert.ToUInt16(255)
        ULStat = DaqBoard.DOut(PortType, DataValue)
        If ULStat.Value <> MccDaq.ErrorInfo.ErrorCode.NoErrors Then
Stop

        'Closing the sprocket

        DataValue = Convert.ToUInt16(127)
        ULStat = DaqBoard.DOut(PortType, DataValue)
        If ULStat.Value <> MccDaq.ErrorInfo.ErrorCode.NoErrors Then
Stop
        System.Threading.Thread.Sleep(1000) 'Will engage
        DataValue = Convert.ToUInt16(255)
        ULStat = DaqBoard.DOut(PortType, DataValue)
        If ULStat.Value <> MccDaq.ErrorInfo.ErrorCode.NoErrors Then
Stop

        'Start 2nd camera and acquire image, and Identify the image and
provide message if 'Yes' then open If 'No'then move back
        Dim Image2 As Object
        'Dim Image3 As Object
        'Dim Image4 As Object
        'Dim Image5 As Object
        Image2 = Me.AxMImage2
        'Image3 = Me.AxMImage3
        'Image4 = Me.AxMImage6
        'Image5 = Me.AxMImage7
        AxMDigitizer2.Grab()
        Image2.Save("C:\Documents and
Settings\Aravind\Desktop\Grain_o_bot\Grainimg\1.tif")
        'Image3.Save("C:\Documents and
Settings\Aravind\Desktop\Grain_o_bot\Grainimg\2.tif")
        'Image4.Save("C:\Documents and
Settings\Aravind\Desktop\Grain_o_bot\Grainimg\3.tif")
        'Image5.Save("C:\Documents and
Settings\Aravind\Desktop\Grain_o_bot\Grainimg\4.tif")


        'AxMImage2.Clear()
        'AxMImage3.Clear()
```

```
        'AxMImage6.Clear()
        'AxMImage7.Clear()
        'AxMDisplay2.Free()
        'Run Matlab to find Grain
        MatLab = CreateObject("Matlab.Application")
        Result_1 = MatLab.Execute("C:\Documents and Settings\Aravind\My
Documents\MATLAB")
        Result_1 = MatLab.Execute("GrainId")
        Gr_Id = MatLab.GetVariable("Iam", "base")
        'MatLab.Quit()
        Dim Grain As String
        If Gr_Id = 1 Then
            Grain = "Barley"
        ElseIf Gr_Id = 2 Then
            Grain = "CWRS"
        ElseIf Gr_Id = 3 Then
            Grain = "Canola"
        ElseIf Gr_Id = 4 Then
            Grain = "Unknown"

        End If
        TextBox1.Text = Grain
        TextBox1.Visible = True
        Label1.Visible = True
        Button2.Visible = True
        Button3.Visible = True

        'Close all

    End Sub

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        Dim ULStat As MccDaq.ErrorInfo
        Dim DataValue As UInt16
        Dim PortNum As MccDaq.DigitalPortType =
MccDaq.DigitalPortType.FirstPortA
        Dim PortType As MccDaq.DigitalPortType
        PortType = PortNum
        ULStat =
DaqBoard.DConfigPort(MccDaq.DigitalPortType.FirstPortA,
MccDaq.DigitalPortDirection.DigitalOut)
        DataValue = Convert.ToUInt16(255)
        ULStat = DaqBoard.DOut(PortNum, DataValue)
        TextBox1.Visible = False
        Label1.Visible = False
        Button2.Visible = False
        Button3.Visible = False
        If ULStat.Value <> MccDaq.ErrorInfo.ErrorCode.NoErrors Then
Stop

    End Sub

    Private Sub Grain_O_Bot_Closing(ByVal sender As Object, ByVal e As
System.ComponentModel.CancelEventArgs) Handles MyBase.Closing
        Dim ULStat As MccDaq.ErrorInfo
        Dim DataValue As UInt16
```

```
        Dim PortNum As MccDaq.DigitalPortType =
MccDaq.DigitalPortType.FirstPortA

        AxMDigitizer1.Free()
        AxMSystem1.Free()
        AxMApplication1.Dispose()
        AxMDigitizer2.Free()
        AxMSystem2.Free()
        AxMApplication2.Dispose()

        ULStat =
DaqBoard.DConfigPort(MccDaq.DigitalPortType.FirstPortA,
MccDaq.DigitalPortDirection.DigitalOut)
        DataValue = Convert.ToUInt16(255)
        ULStat = DaqBoard.DOut(PortNum, DataValue)
        If ULStat.Value <> MccDaq.ErrorInfo.ErrorCode.NoErrors Then
Stop
        End
    End Sub

    Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button2.Click
        'To Open
        Dim ULStat As MccDaq.ErrorInfo
        Dim BoardNum As Integer
        BoardNum = 0
        DaqBoard = New MccDaq.MccBoard(BoardNum)
        Dim A, B, C, D, F, G, I, J, K, L, M, N As Integer

        Dim BitValue As MccDaq.DigitalLogicState
        Dim BitNum As Integer 'Changed from Short to Integer
        Dim PortType As MccDaq.DigitalPortType
        Dim PortNum As MccDaq.DigitalPortType =
MccDaq.DigitalPortType.FirstPortA
        Dim Direction As MccDaq.DigitalPortDirection =
MccDaq.DigitalPortDirection.DigitalOut
        Dim DataValue As UInt16
        PortType = PortNum
        ULStat =
DaqBoard.DConfigPort(MccDaq.DigitalPortType.FirstPortA,
MccDaq.DigitalPortDirection.DigitalOut)

        'Open Sprocket

        DataValue = Convert.ToUInt16(191)
        ULStat = DaqBoard.DOut(PortType, DataValue)
        If ULStat.Value <> MccDaq.ErrorInfo.ErrorCode.NoErrors Then
Stop
        System.Threading.Thread.Sleep(2000) 'Will engage
        DataValue = Convert.ToUInt16(255)
        ULStat = DaqBoard.DOut(PortType, DataValue)
        If ULStat.Value <> MccDaq.ErrorInfo.ErrorCode.NoErrors Then
Stop
        System.Threading.Thread.Sleep(10000)

        'Close Sprocket
        DataValue = Convert.ToUInt16(127)
```

```
        ULStat = DaqBoard.DOut(PortType, DataValue)
        If ULStat.Value <> MccDaq.ErrorInfo.ErrorCode.NoErrors Then
Stop
        System.Threading.Thread.Sleep(2000) 'Will engage
        DataValue = Convert.ToUInt16(255)
        ULStat = DaqBoard.DOut(PortType, DataValue)
        If ULStat.Value <> MccDaq.ErrorInfo.ErrorCode.NoErrors Then
Stop

        'Return Back to oroginal location
        'Bring z back to base
        DataValue = Convert.ToUInt16(223)
        ULStat = DaqBoard.DOut(PortType, DataValue)
        If ULStat.Value <> MccDaq.ErrorInfo.ErrorCode.NoErrors Then
Stop
        System.Threading.Thread.Sleep(1000) 'Change to Length of Z
        DataValue = Convert.ToUInt16(255)
        ULStat = DaqBoard.DOut(PortType, DataValue)
        If ULStat.Value <> MccDaq.ErrorInfo.ErrorCode.NoErrors Then
Stop

        'Bring Y to Base
        DataValue = Convert.ToUInt16(251)
        ULStat = DaqBoard.DOut(PortType, DataValue)
        If ULStat.Value <> MccDaq.ErrorInfo.ErrorCode.NoErrors Then
Stop
        System.Threading.Thread.Sleep(1000) 'Change to Length of Y
        DataValue = Convert.ToUInt16(255)
        ULStat = DaqBoard.DOut(PortType, DataValue)
        If ULStat.Value <> MccDaq.ErrorInfo.ErrorCode.NoErrors Then
Stop

        'Bring X to Base
        DataValue = Convert.ToUInt16(253)
        ULStat = DaqBoard.DOut(PortType, DataValue)
        If ULStat.Value <> MccDaq.ErrorInfo.ErrorCode.NoErrors Then
Stop
        System.Threading.Thread.Sleep(1000) 'Change to Length of X
        DataValue = Convert.ToUInt16(255)
        ULStat = DaqBoard.DOut(PortType, DataValue)
        If ULStat.Value <> MccDaq.ErrorInfo.ErrorCode.NoErrors Then
Stop

    End Sub

    Private Sub Button3_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button3.Click
        'To Close
        Dim ULStat As MccDaq.ErrorInfo
        Dim BoardNum As Integer
        BoardNum = 0
        DaqBoard = New MccDaq.MccBoard(BoardNum)
        Dim A, B, C, D, F, G, I, J, K, L, M, N As Integer

        Dim BitValue As MccDaq.DigitalLogicState
        Dim BitNum As Integer 'Changed from Short to Integer
        Dim PortType As MccDaq.DigitalPortType
```

```
        Dim PortNum As MccDaq.DigitalPortType =
MccDaq.DigitalPortType.FirstPortA
        Dim Direction As MccDaq.DigitalPortDirection =
MccDaq.DigitalPortDirection.DigitalOut
        Dim DataValue As UInt16
        PortType = PortNum
        ULStat =
DaqBoard.DConfigPort(MccDaq.DigitalPortType.FirstPortA,
MccDaq.DigitalPortDirection.DigitalOut)
        'Bring z back to base
        DataValue = Convert.ToUInt16(223)
        ULStat = DaqBoard.DOut(PortType, DataValue)
        If ULStat.Value <> MccDaq.ErrorInfo.ErrorCode.NoErrors Then
Stop
        System.Threading.Thread.Sleep(1000) 'Change to Length of Z
        DataValue = Convert.ToUInt16(255)
        ULStat = DaqBoard.DOut(PortType, DataValue)
        If ULStat.Value <> MccDaq.ErrorInfo.ErrorCode.NoErrors Then
Stop

        'Bring Y to Base
        DataValue = Convert.ToUInt16(251)
        ULStat = DaqBoard.DOut(PortType, DataValue)
        If ULStat.Value <> MccDaq.ErrorInfo.ErrorCode.NoErrors Then
Stop
        System.Threading.Thread.Sleep(1000) 'Change to Length of Y
        DataValue = Convert.ToUInt16(255)
        ULStat = DaqBoard.DOut(PortType, DataValue)
        If ULStat.Value <> MccDaq.ErrorInfo.ErrorCode.NoErrors Then
Stop

        'Bring X to Base
        DataValue = Convert.ToUInt16(253)
        ULStat = DaqBoard.DOut(PortType, DataValue)
        If ULStat.Value <> MccDaq.ErrorInfo.ErrorCode.NoErrors Then
Stop
        System.Threading.Thread.Sleep(1000) 'Change to Length of X
        DataValue = Convert.ToUInt16(255)
        ULStat = DaqBoard.DOut(PortType, DataValue)
        If ULStat.Value <> MccDaq.ErrorInfo.ErrorCode.NoErrors Then
Stop
    End Sub
End Class
```

2.d. Grainid.m

```
clc;
clear all;
close all;

dirname = input('Please enter the directory name to analyse: ','s');
out = input ('Please enter the results file name: ', 's');

%=========================================================================
====
%  Finding the file names from the given directory
%=========================================================================
====

n= length(dirname);
index=strfind(dirname,'\');
index_len=length(index);

%=========================================================================
====
% Getting Current path and directory information
%=========================================================================
====
cur_path = ''; %Current Path
cur_path = strcat(cur_path,dirname(1:index(index_len)-1));
cur_dir  = ''; %Current directory
cur_dir  = strcat(cur_dir,dirname(index(index_len)+1:n));

%=========================================================================
====
% Getting Files Information
%=========================================================================
====
files = dir(fullfile(cur_path,cur_dir));
temp_name = ' ';
temp_name = strcat(temp_name, cur_path,'\',out);
disp('Writing the data to the file');disp(temp_name);
fid = fopen(temp_name,'W');

%=========================================================================
====
% Getting Files Information
%=========================================================================
====

ftr.name = 'blank';
ftr.num_points = 0;
ftr.locationx = 0;
ftr.locationy = 0;


ftr(2).name = 'blank';
ftr(2).num_points = 0;
```

```
ftr(2).locationx = 0;
ftr(2).locationy = 0;


for i = 3:length(files)
    disp ('Processing file:'); disp(files(i).name);
   %[A, map] = imread(strcat(dirname,'\',files(i).name));
    %[B, map] = imread(strcat(dirname,'\',files(i).name));
[C, map] = imread(strcat(dirname,'\',files(i).name));
%A= imread('C:\Documents and
Settings\Aravind\Desktop\Grain_Identification\BARLEY\BARLEY1.tif');
%B= imread('C:\Documents and
Settings\Aravind\Desktop\Grain_Identification\CWRS\CWRS1.tif');
%C=imread('C:\Documents and
Settings\Aravind\Desktop\Grain_Identification\CANOLA\CANOLA1.tif');
%D=A(500:1100, 191:791);
%imshow(C);
%E=B(225:825, 300:925);
%imshow(D);
F=C(550:1150, 225:825);
%G=imadjust(F);%Change to appropriate subimage
G=F;
%H=imadjust(E);
%I=imadjust(F);
%GG = rangefilt(G);
%HH = rangefilt(H);
%II = rangefilt(I);
%imshow(D), figure,imhist(D),figure,imshow(E), figure, imhist(E),
figure,imshow(F),figure,imhist(F),figure,imshow(G),figure,
imhist(G),figure,imshow(H),figure, imhist(H),figure, imshow(I),figure,
imhist(I),figure, imshow(GG),figure,imhist(GG),figure,
imshow(HH),figure,imhist(HH),figure,imshow(II),figure, imhist(II);

CountA = 0;
for I = 1 : 600
    for J = 1 : 600
        if G(I,J)>160 && G(I,J)<165
            CountA = CountA+1;
        end
    end
end

CountB = 0;
for I = 1 : 600
    for J = 1 : 600
        if G(I,J)>75 && G(I,J)<85
            CountB = CountB+1;
        end
    end
end

CountC = 0;
for I = 1 : 600
    for J = 1 : 600
        if G(I,J)>100 && G(I,J)<105
            CountC = CountC+1;
```

```
        end
    end
end
CountD = 0;
for I = 1 : 600
    for J = 1 : 600
        if G(I,J)>105 && G(I,J)<115
            CountD = CountD+1;
        end
    end
end

CountE = 0;
for I = 1 : 600
    for J = 1 : 600
        if G(I,J)>195 && G(I,J)<205
            CountE = CountE+1;
        end
    end
end
CountF = 0;
for I = 1 : 600
    for J = 1 : 600
        if G(I,J)>175 && G(I,J)<185
            CountF = CountF+1;
        end
    end
end
CountG = 0;
for I = 1 : 600
    for J = 1 : 600
        if G(I,J)>205 && G(I,J)<215
            CountG = CountG+1;
        end
    end
end
CountH = 0;
for I = 1 : 600
    for J = 1 : 600
        if G(I,J)>145 && G(I,J)<155
            CountH = CountH+1;
        end
    end
end

CountI = 0;
for I = 1 : 600
    for J = 1 : 600
        if G(I,J)>25 && G(I,J)<35
            CountI = CountI+1;
        end
    end
end

CountJ = 0;
for I = 1 : 600
```

```
    for J = 1 : 600
        if G(I,J)>45 && G(I,J)<55
            CountJ = CountJ+1;
        end
    end
end

CountK = 0;
for I = 1 : 600
    for J = 1 : 600
        if G(I,J)==150
            CountK = CountK+1;
        end
    end
end
CountL = 0;
for I = 1 : 600
    for J = 1 : 600
        if G(I,J)==200
            CountL = CountL+1;
        end
    end
end

if CountL >0 && CountL < 250 && CountG >100 && CountG < 1000 && CountF
>1000 && CountF < 5000&& CountE >100 && CountE < 2000
    Iam = 1;

elseif CountL >250 && CountL < 700 && CountG >2500 && CountG < 4000 &&
CountF >6000 && CountF < 9000&& CountE >3500 && CountE < 6000
     Iam = 2;

elseif CountL == 0 && CountG == 0 && CountF == 0 && CountE ==0 &&
CountJ >100 && CountJ < 5000
      Iam = 3;
else
      Iam = 4;


end


ftr(i).name=files(i).name;
ftr(i).CountA = CountA;
ftr(i).CountB = CountB;
ftr(i).CountC = CountC;
ftr(i).CountD = CountD;
ftr(i).CountE = CountE;
ftr(i).CountF = CountF;
ftr(i).CountG = CountG;
ftr(i).CountH = CountH;
ftr(i).CountI = CountI;
ftr(i).CountJ = CountJ;
ftr(i).CountK = CountK;
ftr(i).CountL = CountL;
```

```
ftr(i).Iam = Iam;
fopen(out, 'wt');
fprintf(fid,'%s
\t%6d\t%6d\t%6d\t%6d\t%6d\t%6d\t%6d\t%6d\t%6d\t%6d\t%6d\t%6d \n',
ftr(i).name,ftr(i).CountI,ftr(i).CountJ,ftr(i).CountA, ftr(i).CountB,
ftr(i).CountC, ftr(i).CountD, ftr(i).CountE, ftr(i).CountF,
ftr(i).CountG, ftr(i).CountH, ftr(i).CountK, ftr(i).CountL,ftr(i).Iam);
%
% CountW = 0;
% for K = 1 : 600
%     for L = 1 : 600
%         if HH(K,L)>150
%             CountW = CountW+1;
%         end
%     end
% end

end
fclose(fid);
```