Reducing Driver Fatigue During Forage Harvesting

Final Design

BREE 495
December 7th 2015

Brett Bennett, Meaghan Dustin, Yasmeen Hitti, Stephen McGuire

Executive Summary

Within recent years farm sizes have increased while the number of agricultural producers has decreased placing a greater workload on a fewer number of workers. Technological development in the agricultural field continually bridges the gap increasing productivity per hour of these systems. Forage harvesting takes place multiple times per year and most commonly includes hay or corn forage. Like any crop, harvest is a very important and time sensitive period for agricultural producers.

Forage harvesting differs from most other harvesting processes as many tasks are performed simultaneously during operation. Large self-propelled harvesters have implemented various systems to reduce operator task complexity during harvest. These options do not exist for pull behind forage harvesters as the gross market is much smaller. Often the harvesting capacity of pull type forage harvesters do not justify the extra cost of purchasing a self-guided offloading system.

Dion is an agricultural machinery company based out of Boisbriand Quebec. They currently lead the market in pull type forage harvesting technology. Dion is always looking to push the limits of what is available on the market and were more than willing to participate in this project. Dion presented their constraints as well as their hopes for a product when the project was completed.

Together it was determined that the design goal would be to develop a system which will provide assistance to the forage harvester operators during harvest to reduce task complexity and cognitive strain. After extensive research and development, it was determined that machine vision provided the optimal solution to this design problem.

The proposed system implements targets which are mounted on the side of the forage wagon. A camera mounted upon the chute of the harvester sends images to a processing unit which can then recognize the targets. Based on placement of the targets within the image, the position of the forage wagon is determined in reference to the chute and chute position is adjusted accordingly.

This design was made functional through prototyping with MATLAB® and an Arduino microprocessor. These systems provided easy modification as well as vast previous development within the machine vision field. It would not be recommended for Dion to continue implementing MATLAB® as it was only used within this design to prove initial design concepts.

The final design concept was proven through testing on site at Dion's testing facilities. Future development of user control interfaces will be required however this was out of the scope of the proposed project.

Table of Contents

Executive Summary 2

Introduction 4

Forage Harvesters and Task Complexity 4

The Client 5

Problem Statement 6

Constraints 6

Analysis and Specifications 7

Prototyping and Optimization 11

Testing 19

Failure Mode Effects Analysis (FMEA) 19

Uncertainty Calculation 20

Cost Analysis 20

Recommendations 21

Design Competition 23

Conclusion 23

Acknowledgements 24

Mentor Information 24

Client Information 24

References 26

Appendices 27

Appendix A: Flowcharts 27

Appendix B: MATLAB® Code 31

Appendix C: FMEA Ranking 31

Introduction

Forage Harvesters and Task Complexity

A forage harvester is a machine which harvests, conditions, and offloads a forage crop during operation. These machines range in size and configuration to best suit the customer's needs. For large operations, self-propelled harvesters are available which can provide up to 1100 horsepower when needed. Pull behind harvesters are available for smaller scale operations and allow the farmer to use their tractor to power the machine. Different configurations are also available which allow operators to harvest both hay as well as corn forage. With the variety of crop, operation size, and field capacity, forage harvesters must come in all shapes and sizes to provide versatility for the customer.

A traditional pull type forage harvester is connected to a wagon from the rear in a train like setup. During harvest the forage harvester picks up the forage, conditions, and offloads into the rear wagon until the wagon is completely full; harvest must stop, the wagon unhitched, and an empty wagon hooked up to the harvester before harvest can continue. In recent years Dion-Ag Inc. has developed a laterally offloading pull type harvester that resembles the offloading methods of full sized self-propelled harvesters. This method of offloading, as seen in Figure 1, provides continuous harvest throughout the day and increased productivity.

Figure 1 Laterally Offloading Pull Type Forage Harvester

While downtime has been decreased, task complexity has increased dramatically by the implementation of lateral offloading in pull type forage harvesting. The harvester operator must monitor the functioning harvester, the tractor speed, and the offloading forage placement within the forage wagon. These tasks add to operator workload and contribute to operator fatigue

(Nicholls, Bren et al. 2004). Cognitive computation is the decision process which the brain must go through when faced with multiple options. The more options that are present result in more decisions constantly having to be made. This combination of increased options within a smaller time frame results in increased cognitive strain (Cooper-Martin 1994). Long periods of increased cognitive strain then result in operator fatigue and ultimately reduce operating efficiency. If one of the tasks presented to the operator can be simplified or eliminated from the operator's focus, daily productivity will be increased.

The Client

Dion-Ag Incorporated is an internationally recognized agricultural machinery company focusing on the production of forage harvesting equipment. Established in 1920 and based out of Boisbriand Quebec, Dion holds 20% of the pull-behind forage harvesting market. With increased research and development this figure has the potential to rise considerably within the coming years.

In recent years Dion has taken immense strides in the development of their pull behind forage harvesting technology. This market is neglected by larger agricultural machinery companies due to its smaller market size and the overshadowing of the larger self-propelled machine market. Dion has developed the largest capacity pull behind harvester on the market and with such expertise in their field customers expect the very best.

Multiple competitors, including New Holland, John Deere, and CLAAS, have developed systems to automate the forage harvester chute placement on their self-propelled harvesters. None of these companies have yet implemented their design upon their pull type harvesters. By adding an automated system to the new Dion lateral offloading forage harvester their machine will be leading the way for pull type harvesting technology.

Dion was contacted and showed interested in developing a machine vision system to reduce the task complexity endured by their operators during harvest. Due to the complexity of the development of such a system Dion requested that the primary goal of the project be limited to the movement of the chute within the constraints of the forage wagon, and recognition of the location of the limits of the wagon. Through development of a system that achieves this, further optimization and modification can be taken on by Dion once the theoretical solution has been proven.

If this design is capable of reducing a single portion of the operator's responsibility while driving, then the cognitive strain of operation will be reduced. When operator fatigue is decreased, productivity increases and operation hours can be increased.

Problem Statement

To develop a system which will provide assistance to the forage harvester operators during harvest to reduce task complexity and cognitive strain.

Constraints

The vast majority of the constraints within this design are to ensure that the final product is compatible and usable by Dion for future projects. This system must be implemented on a Dion laterally offloading forage harvester and allow the harvester to work with or without this system installed. Therefore, there should be limited machine modification. This design should also be compatible with the current electronics applied on the machine. Currently the wiring system in place consists of hydraulic actuators that control the x, y, and z motion of the harvesting arm. Since this design uses machine vision with computer computations there should be minimal computational analysis to provide a quick response.

To reduce costs non-complex system components should be implemented. Dion has provided a preliminary budget of $1500.00 (CAD) to develop a basic prototype. Depending on prototype effectiveness this amount may be increased to allow for further testing. The final goal of this design project is for Dion to be able to market this machine vision control component as a $10,000 add-on option to their forage harvest machines, or be able to sell it as a standalone system. The cost target is to make 20% profit off of this system, therefore it has to cost less than $8000 to produce.

Analysis and Specifications

After extensive research and development it was determined that a machine vision system using targets mounted on the forage wagon would provide best results given the design constraints. This overall design is shown in Figure 2. Many variations of this design were tested including variable camera placement and angle, control signal input method, and target colour. After many trial runs and design modifications the final hardware arrangement and software script was developed.

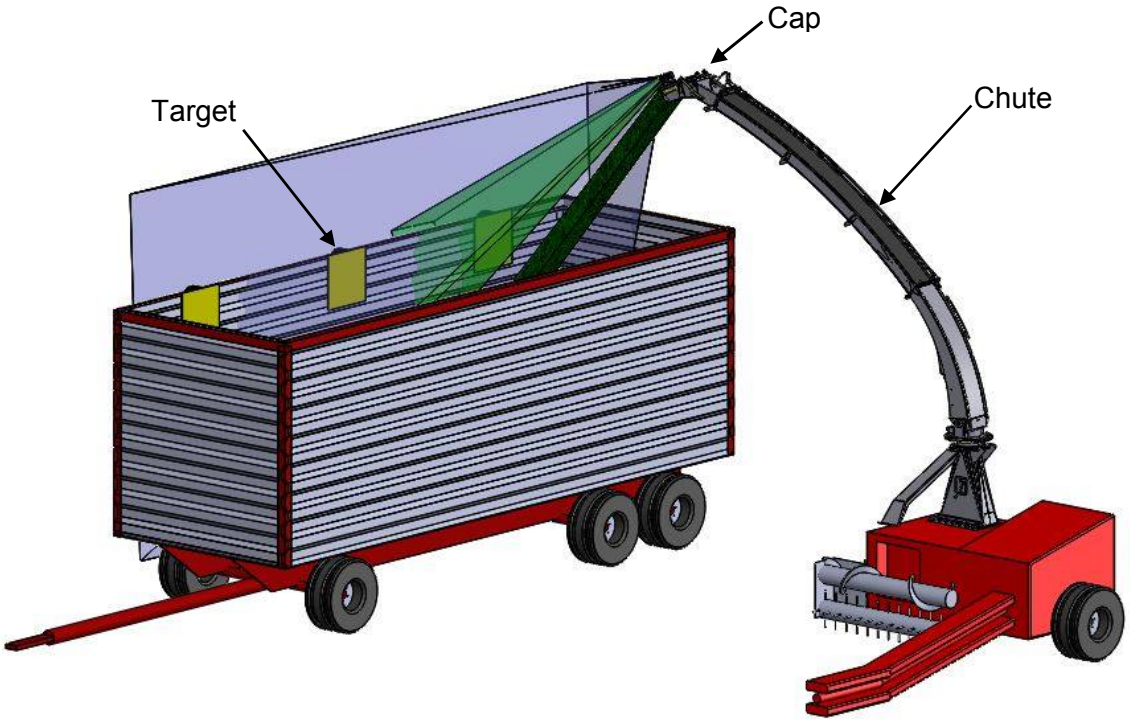


Figure 2 Forage Harvester, Wagon, & Design System

The camera used for image processing was placed upon the cap of the forage chute. The trajectory of the forage from the tip of the chute into the forage wagon can be assumed linear as per the recommendation of the client. A camera mount was implemented to provide an interior angle between the plane of the camera and the upper plane of the forage chute cap. This angle ensures that the forage is blown into the centre of the wagon while the camera is directed toward the targets placed upon the upper edge of the wagon. Depending upon operator preference this camera mount provides infinite variability between an interior angle of 0 to 90 degrees. This provides excessive variance to the operator ensuring that needs are met regardless of operator preference. Forage trajectory and camera field of view can be seen below in Figure 3.

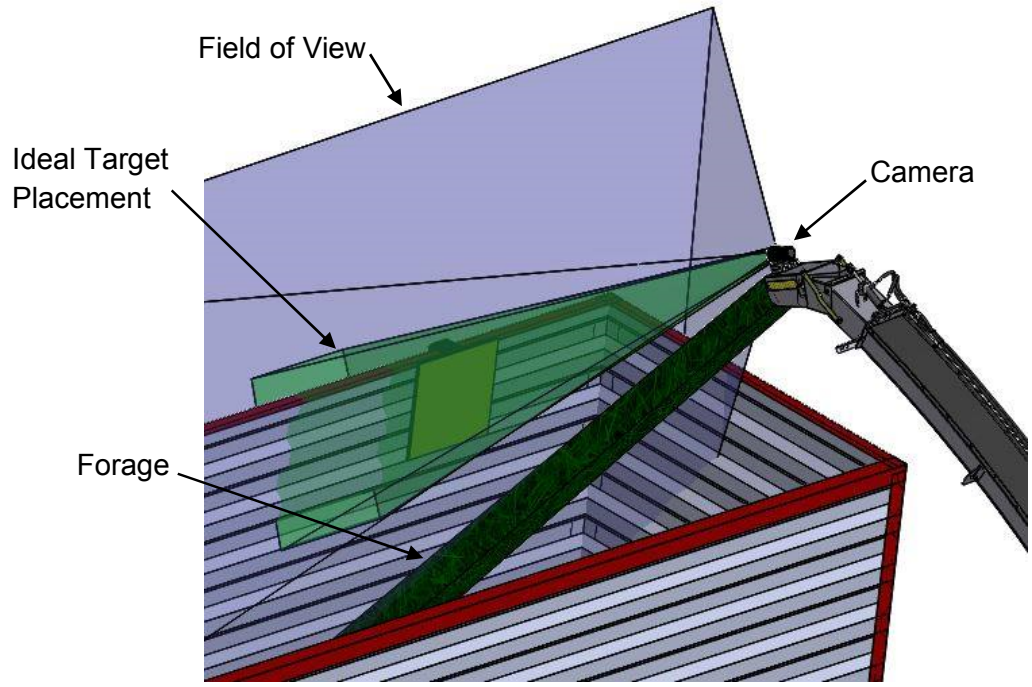


Figure 3 Camera View

During preliminary testing the 12 volt output from an Adafruit motor shield was connected directly into the electronically actuated hydraulic solenoids on the rear of the harvester. This provided on site testing functionality but was impractical to implement during in-field testing. By mounting the processing unit inside the tractor cab, durability as well as functionality was increased. Through proper installation it is possible for the proposed system to be tested with only one operator while harvesting. Integration of a 12 volt toggle switch provides the option to disconnect from the processing system control and reconnect to the manual operator control of the chute. The wiring diagram can be viewed in Figure 5 on the following page.

Using machine vision to control the forage chute of a Dion pull behind forage harvester required multiple components. These included physical components as well as a code to control the entire system.

Camera & Camera Mount

A camera was required to relay images of where the forage chute was aiming back to the automated code. The camera needed to be able to capture a large enough image that a target could always be seen, while having high enough resolution to pick out accurate details. The camera also needed to be sturdy enough that it could last in various outdoor conditions, and withstand the dust that is created while harvesting.

The current Dion forage harvester has a camera located ⅔ of the way up the chute. This camera is connected to the screen system located inside the tractor. The angle of the camera proved to be too

narrow, as there were certain locations where no target would show even though the chute was located within the forage wagon. Therefore the design required an additional camera with a higher resolution to ensure a target was always in the line of sight.

A magnetic camera mount will ensure the camera can easily be adjusted while maintaining one position while in use. If anything were to happen to the camera it is also accessible to quickly be replaced therefore reducing operational downtime. The camera mount was designed and built based on field testing; it was designed as a hinge and 3D printed. A magnet was bolted to camera mount to fasten the mount to the chute. The camera and camera mount design can be seen in Figure 4.

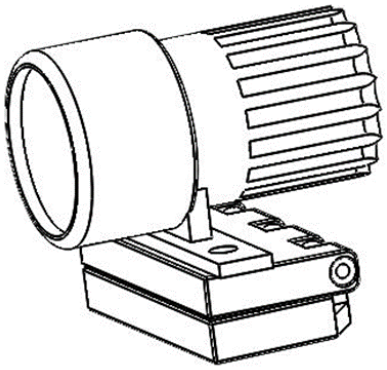


Figure 4 Camera & Camera Mount

Chute Control

The chute control required a physical component to relay the code. This can be done by using the solenoid connections at the base of the chute arm. Wiring in the circuit containing the Arduino motor shield and the external battery allows the code to take control of the chute solenoids.

Machine hookup

The automated system will be connected using a circuit wired directly to the solenoids on the harvester arm. This is done using wires connecting from the pins of the Arduino board to the corresponding wires in the Dion electrical system (Figure 5).

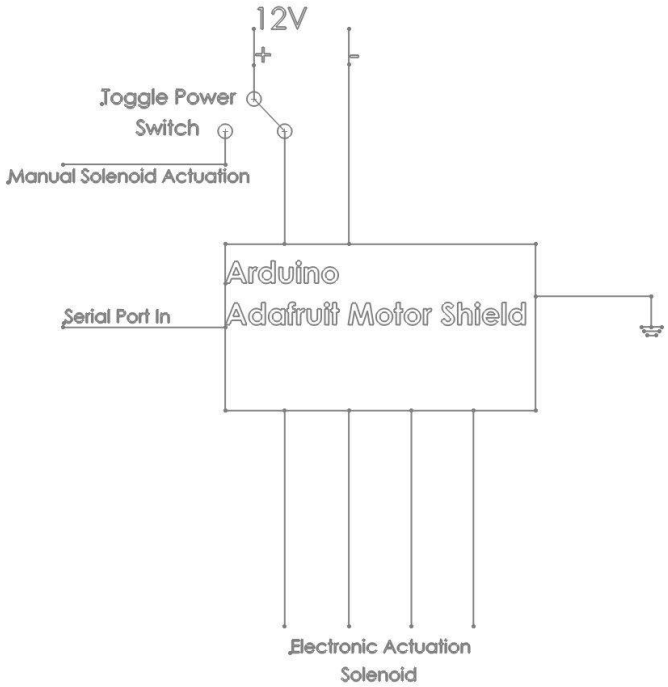


Figure 5 Wiring Diagram

Code

MATLAB® was used to create the computer code that controls the system. The MATLAB® code is then passed to the Arduino Uno using crossover code.

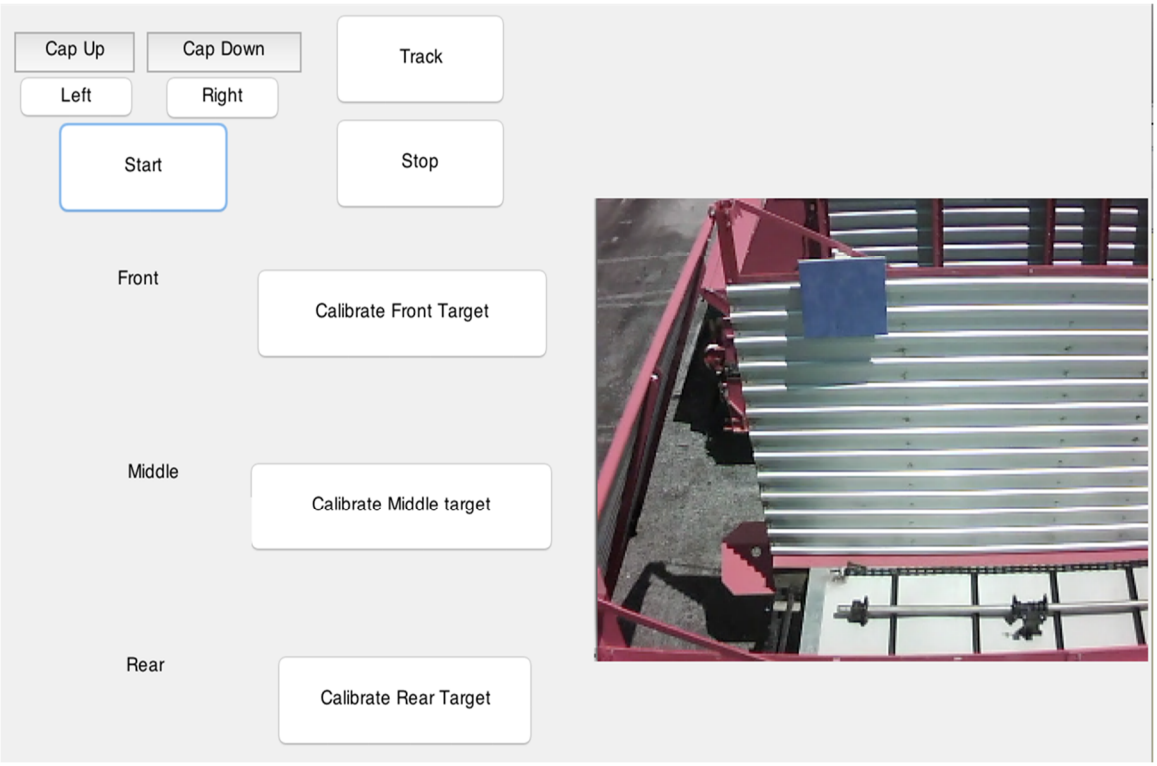


Figure 6 Operator GUI

The MATLAB® code consists of one main graphical user interface (GUI), as seen in Figure 6. Once the user starts the program by pressing the Start button they are prompted to connect to an Arduino. This option is given because the Arduino only needs to be connected once; if the user has decided to stop and then restart the program the Arduino does not need to be reconnected. The next popup menu gives the user the option to analyze live video from the camera, saved video, or still images. The final step in setting up the program is to calibrate the targets. This is done by the operator tracing the outline of a visible target on the screen. All three targets need to be calibrated so that the program can recognize each one. Once each target has been calibrated the Track button is selected and the chute is controlled through the program.

Flowcharts were created to outline the program process, these are located in Appendix A. The actual published code is in Appendix B.

Arduino Uno & Motor Shield

The Arduino Uno was used as the motherboard of the control circuit. This system was chosen for its affordable price and efficiency as a control board. To attain the required 12V to operate the solenoids within the forage harvester an Adafruit motor shield was used. This allows computer code to be uploaded to the shield, which can then run the forage harvester based on the uploaded program.

Targets

Targets were used to help the program determine the location of the chute with respect to the forage wagon. Multiple types of targets were considered including different shapes, different colours, and infrared. Wooden targets were initially used to test the system as they were the easiest to build, and the material was available.

Prototyping and Optimization

In the development of a machine vision guidance system different hardware and software options and configurations were explored. Following is a description of the development of the prototype, why the components and methods were used and how the shortcomings of the system were addressed, re-designed, and optimized before arriving at the current prototype.

To be able to control the harvester, commands generated from the software must be converted into 12 volt control signals to open and close the solenoid valves. The first solution to this problem was the use of a data acquisition card or DAQ such as the National Instruments MYDAQ. A MYDAQ was considered due to familiarity with the card as well as availability. The MYDAQ is capable of receiving digital commands over a serial connection and outputting up to 5 volts on 8 different terminals. Since the solenoids require 12 volt power, 12 volt relays would be required to raise the voltage of the 5 volt terminals. The 5 volt signal would provide the switching voltage, and the 12 volt side would be wired to the 12 volt power supply at the input and the solenoid at the output. This set up would have required 4 relays to control 2 bi-directional hydraulic solenoids. There were multiple concerns with this set up: first the wiring was relatively complex, it was difficult to troubleshoot problems in the system, and it would have been more difficult to transport and set up during testing.

Another concern with using a MYDAQ system was the life expectancy of the relays, which are a potential failure point of the system. The mechanical life expectancy of relays is generally millions of cycles however the electrical life is 100,000 operations (Tyco). This life expectancy is adequate for many applications, however the control system for the harvester will be making corrections constantly for many hours of operation and may reach the life expectancy of the relay in a relatively short amount of time. To solve the durability issue and to simplify the wiring, the use of a microcontroller with a 12 volt motor shield was proposed.

A microcontroller, such as the Arduino Uno, is capable of receiving commands from a personal computer over a serial connection and turning on or off various 5 volt pins. The microcontroller is used in conjunction with a 12 volt motor shield. When actuated the motor shield provides 12 volts at up to 3.2 amps (Toshiba, 2012). The microcontroller uses a metal oxide semi-conductor field effect transistor (MOSFET) instead of an electromechanical switch. A MOSFET has three main components: a source, a drain, and a gate. The 5 volt pin of the microcontroller connects to the gate where the source and drain connect to the positive terminal of the 12 volt power supply and the load respectively. When there is no electric potential at the gate, the material is unable to conduct between the source and drain. When the 5 volt pin is turned on, the electric potential at the gate allows the flow of electrons between the source and the drain, as seen in Figure 7.

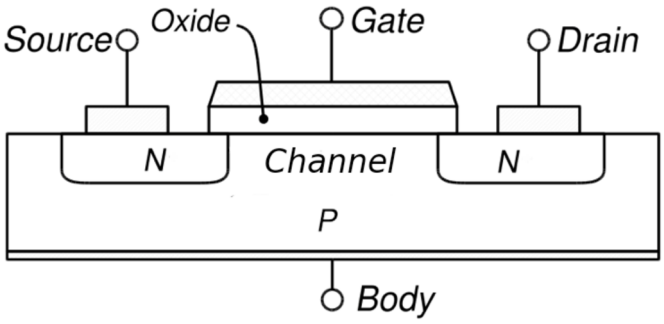


Figure 7 Diagram of a MOSFET

Since a MOSFET has no moving parts this allows for extremely fast switching (100 kHz for the Arduino motor shield) and an extremely long lifespan (Toshiba, 2012). Lifespan information is not readily available for many MOSFETs, however nearly all failures are due to manufacturers defects or working outside of the operating temperature range (Tyco). Therefore, this design should be extremely reliable given that the components will be used within their operating limits.

The camera used for testing was a USB 2.0 surveillance camera. The camera provided adequate resolution and durability as it is in a shock and weather resistant case. The camera used has a narrow field of view and poor colour representation. To improve on the operation of the system a camera with a wider field of view and better colour representation should be used to improve tracking and image segmentation.

For developing the guidance software, several programming languages were explored. MATLAB®, Python, C/C++ and LabVIEW all have toolboxes and libraries for machine vision and control. MATLAB® was chosen for the majority of software development due to its powerful and complete machine vision toolbox, familiarity with the software, libraries for communication with Arduino, as well as free access to an otherwise very expensive programming environment.

Testing and Optimization:

Colour thresholding was explored as the first step at extracting features of interest. Colour information from the camera comes as an M X N X 3 matrix. M and N are the resolution of the camera in pixels in the horizontal and vertical directions respectively. MATLAB® stores the three dimensional matrix in RGB format, meaning that the first matrix has values corresponding to how red each pixel in the image is, and the green and blue pixel value information is contained in matrix 2 and 3 respectively. The digital pixel information is 8 bit meaning that every pixel can have a value between 0 and 255. For example in the red matrix, a value of 0 indicates the pixel contains no red component whereas 255 is pure red. Thresholding is specifying what range of colours are to be displayed in the image. This can be done simply by finding the locations of pixels that are within the specified threshold and setting values outside of this range to 0. For a simplified example Figure 8 shows the thresholding of an image of different colour squares. If the objects of interest do not contain any red, the red channel may be removed by thresholding, setting the red channel to zero.

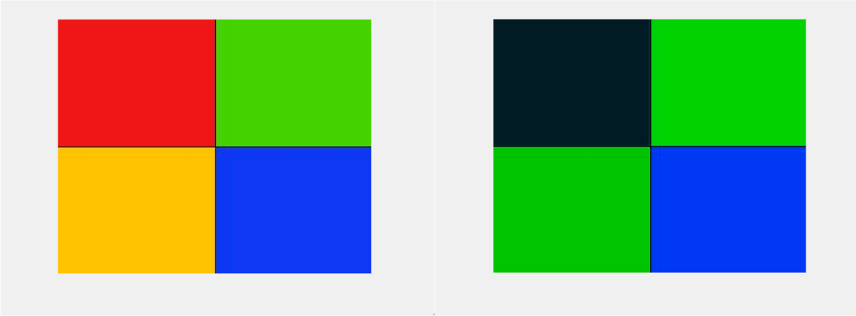


Figure 8 Before Thresholding Of Red Colour Channel And After.

The image that underwent thresholding now contains only objects that do not contain any red value, as can be seen in Figure 8. In developing a machine vision algorithm it is important to separate objects of interest from objects of noninterest. Thresholding allows removal of objects that do not contain pixel values within the specified range. This leaves a number of other objects that will need to be removed until only those of interest remain. A threshold image is usually converted to a logical binary matrix, consisting of ones where the pixel falls within the specified threshold and zero when outside of the specified range. The remaining regions of ones surrounded by zeroes are called objects.

Originally it was attempted to threshold an image of the forage wagon such as the outside edge of the wagon. The colour threshold application in MATLAB® was initially used to get an idea of the right range of pixel data. This application also allowed easy switching between different colour spaces. RGB, HSV, YCbCr, and L*a*b* colour spaces were explored. HSV yielded much better segmentation of the image than RGB. HSV also gives much more useful information about the image. RGB colour space uses three different integers to describe the value of red, green and blue in each pixel. HSV colour space combines all of the colour information into a

single float value between zero and 1, with red at a value of 0 and blue at 2/3. Saturation provides information about how close the pixel is to a pure colour, or the white component of the colour. If a colour is fully saturated (no white component) it will appear as the pure colour. Less and less saturated colours appear more faded until they are un-saturated, having a value of zero, and are pure white. Value indicates the black component or brightness of the pixel. A value of one is bright, where a value of zero is black (Mathworks, 2015). Figure 9 provides an illustration of the Hue Saturation Value (HSV) colour space. For these reasons the HSV colour space provides more useful information in the same amount of data as RGB uses. This allows for much better thresholding and therefore better tracking of objects in the image.

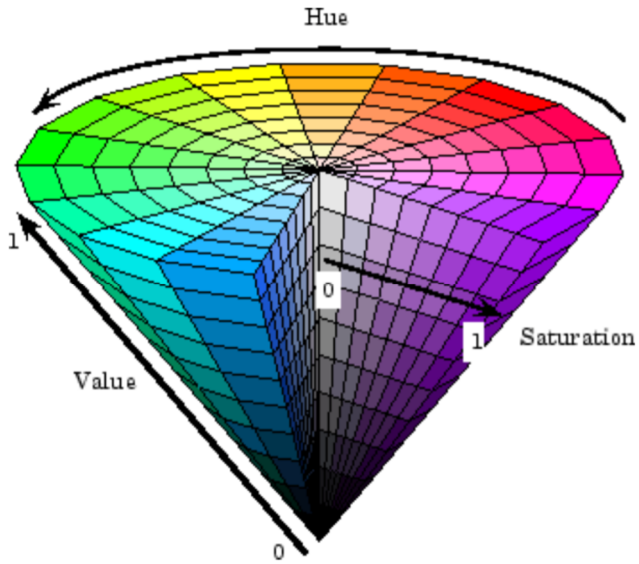


Figure 9 Illustration Of The Hue, Saturation, Value Colour Space.

A method needed to be developed for calibration of the threshold values for the H, S, and V channels. Initially the MATLAB® impixel function was used which allowed a user to click points on the image inside the target area (Toshiba, 2012). From this, the maximum and minimum values of the selected pixels were found and used to create the logical matrix used for object identification and tracking. This method however proved to have several serious issues as it was time consuming to click on enough points to have a good representation of changes in values within the region; this led to bad thresholding and therefore poor tracking. As a solution to this, the MATLAB® imfreehand function was used, which allows user input to select a region of interest and all points within that boundary are used to calculate the high and low threshold values (Mathworks, 2015). This was a major improvement as it gave a complete representation of the pixel values of the region of interest and allowed more points to be used than user selected points.

Using the maximum and minimum values for each channel proved not to be the best method for calculating the threshold, as a single outlying value could throw the calibration off. To solve this problem, a histogram for each channel was generated and a variable for the minimum frequency of repeated pixel values created. The histogram by default does a small boxcar average of all of the points and then indicates the number of values inside the boxcar. Using this histogram the smallest and largest values inside the range were selected as long as the frequency of the pixels was above the specified value; this removed outliers and provided a much more accurate threshold. Using a variable for pixel frequency allowed the calibration to be more dynamic. The calibration was performed in a loop only ending by user prompt when the binary mask is satisfactory. If the calibration yields a mask with more than one object in it, the frequency variable is increased to tighten the threshold. This method may also be used to provide better tracking when the binary image is too noisy for detection.

After attempting to track features of the wagon, targets of a specific shape and colour were created to allow much easier tracking of the wagon and the ability to work on multiple wagons without changing the software. Three square 24inch by 24 inch targets were created from plywood with rudimentary brackets to hang them over the side of the wagon. The three targets were painted different colours and mounted at the rear, middle, and front of the wagon to allow the program to distinguish where each target position was.

A graphical user interface was created using GUIDE in MATLAB® to allow easier use of the program and make rapid changes to parts of the code that did not work well. The calibration method described above was transferred to a function file and was called in three separate GUI buttons corresponding to the calibration of the three targets, to set up the thresholds. A track button was added in which the three sets of threshold values were passed to create the binary masks. Since the targets were square several properties could be evaluated to ensure that the target is being tracked and not another object. Even with good thresholding there is noise in the image that needs to be handled: small white spots on the image that are small areas of pixels inside the threshold, as well as the object of interest not appearing as square as it should. To handle these types of noise, morphological operations were used.

Morphological opening was used to eliminate objects in the mask smaller than a specified number of pixels. After this the image was dilated using a square structuring element and the area was filled to leave shapes more distinct. To determine which object to track the largest object was selected and its major and minor axes were evaluated. The ratio of these axes was calculated to determine how square the object was. If the object was both square enough and large enough it would be selected as an object to track and its centroid and bounding box would be calculated, and output to the GUI. Testing was required to determine how much tolerance needed to be allowed for the major/minor axis ratios as the object appeared less and less square at greater incidence angles. A ratio of 1.5 yielded correct object tracking without selecting incorrect objects. A weakness of this selection method is perfectly round objects, although it is unlikely that a perfectly round object with the same hue, saturation, and value as the target would be in the image, it is possible. To address this, the perimeter/area ratio of the object should be

evaluated to determine if it is square or round. Figure 10 shows the segmentation and proper identification of the target on the harvester.

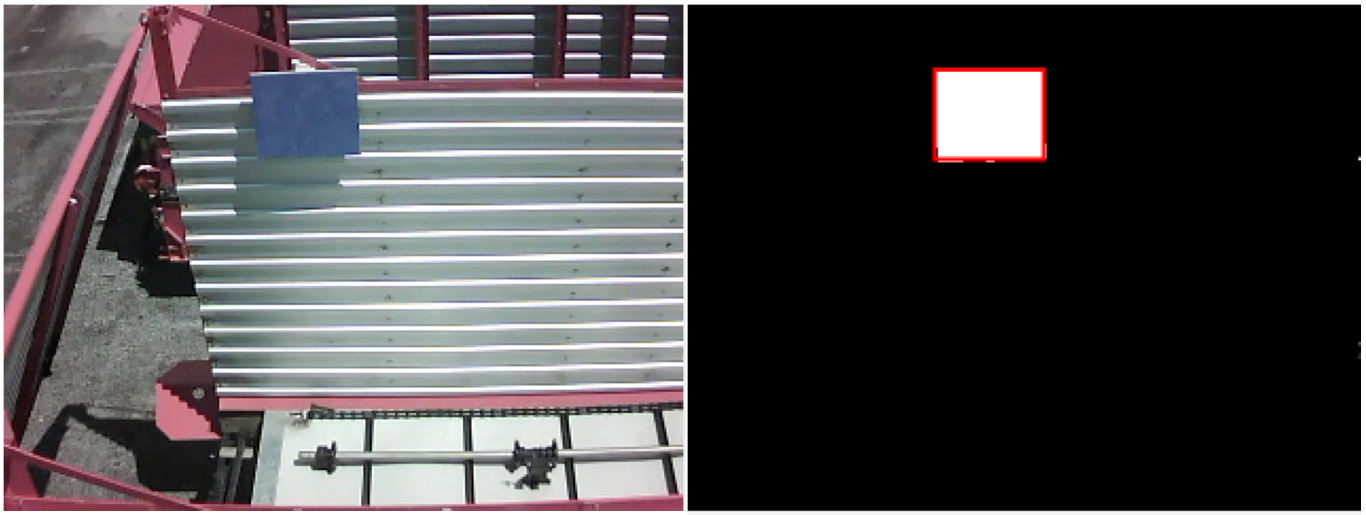


Figure 10 Image Thresholding And Proper Selection of Target

The mounting location of the camera presented several challenges during testing. When testing the vision system, the camera was mounted and calibrated for the targets. The chute of the harvester was moved using the manual controls on the tractor to see how the system would respond. Small rotation of the chute or movement of the cap resulted in the system losing track of the target. This was the result of the thresholds set up during calibration, as they were perfectly set up for the specific light condition and orientation at the start of testing, as soon as the orientation changed the object was no longer within the limits. Testing was done with the camera and target to see how much bigger the threshold range needed to be to allow for the change in orientation. A ten percent increase in the threshold values allowed for the change of camera orientation required without too much of an increase in image noise.

Painted targets with a different coloured border were examined. If the background of the main target area is inside the threshold range it will cause poor or no tracking. With the target with a border, the background can be the exact hue, saturation and, value as the target but it is still possible to track it, as the border segments the two objects. This was tested using a camera and the targets but has not yet been tested on the harvester.

To get the target location information to the solenoids an Arduino with a motor shield was used. To communicate with the Arduino, serial communication was used. The Arduino add-on for MATLAB® was used, greatly simplifying programming as the Arduino script is generated and uploaded from MATLAB®. Before automating the harvester, GUI buttons were added to see if the harvester could be digitally controlled. Connectors were donated by Dion that were crimped on to the Arduino cables so the system could be plugged into the solenoid leads. After testing for proper operation of the solenoids, it was found that the harvester moved very rapidly for a very

short on time of the microcontroller. The '*on*' time of the controller was adjusted to find the minimum '*on*' period possible, which is limited by the time required to move the solenoid from on/off states. This time was found to be 0.1 seconds, moving the harvester in small enough steps for smooth control to avoid under-shooting or over-shooting once the control was automated.

To allow automated control of the harvester, the camera needed to center the target in the image. The center of the screen was calculated based on the camera resolution. The centroid of the target could deviate from this point by 50 pixels before sending control signals to move the chute. Different ranges of deviation from the center of the screen were tested, however 50 pixels allowed the chute to remain focused on the target without constantly sending control signals due to under-shooting or over-shooting the allowable area. Once outside the desired location, the controller was turned on for 0.1 seconds in the direction towards the center and then the position is re-evaluated. This worked very well for keeping the harvester focused on the target, even when the harvester was at large distances from the wagon. However, when driving at higher speeds, the control and re-evaluation was too slow to keep up with the change in position. To solve this, once the harvester is a certain amount outside the desired location (it is falling behind), a transfer function should be used to increase the '*on*' time of the controller to catch up with the change in position. Control with a time domain transfer function was not tested but an example transfer function is provided below.

Transfer function:

$$Motor\ on\ time = 0.1\ seconds \cdot \frac{Distance\ away\ from\ center\ of\ screen}{allowable\ distance\ from\ centre\ of\ screen}$$

Once the target is just outside the allowable deviation of 50 pixels, the motor will turn on for 0.1 seconds to get back in the desired area. If the target keeps falling behind, the '*on*' time will increase linearly. For example, the resolution of the camera in X is 640 pixels, so the furthest the object can be away from the center is 320 pixels, this would result in a motor on time of .64 seconds, allowing the harvester to catch back up.

The tracking system worked well during the day, however in the late afternoon with the sun low in the sky, shadowing from the harvester chute as well as reflection of the targets became an issue during testing. Several solutions to this issue are possible to optimize the operation of the harvester under changing light conditions. First, the paint used for the targets had a gloss coating adding greatly to the bright spots reflected by the targets. Using a matte finish would provide a quick and easy improvement on performance of the system. Adding a light source to the targets and/or the chute to provide consistent illumination would also improve image segmentation as the target would remain visible even if the harvester is blocking sunlight. Software improvements may also provide a huge improvement during poor lighting conditions. Algorithms developed to handle reflection and shadow may be used to improve image segmentation significantly. The algorithm developed by Bradley et. al presents a method for image segmentation that is incredibly robust to changing illumination conditions (Bradley,

2007). In this method, the value of each pixel in the image is compared to the average of surrounding pixels, creating a local moving average. If the pixel of interest is a specified percentage above the average, the pixel is set to zero, otherwise it is set to white (Bradley, 2007). This adaptive thresholding method has the potential to greatly improve image segmentation for the forage harvester targets under varying light conditions. Figure 11 shows the adaptive thresholding methods effectiveness at handling shadow.

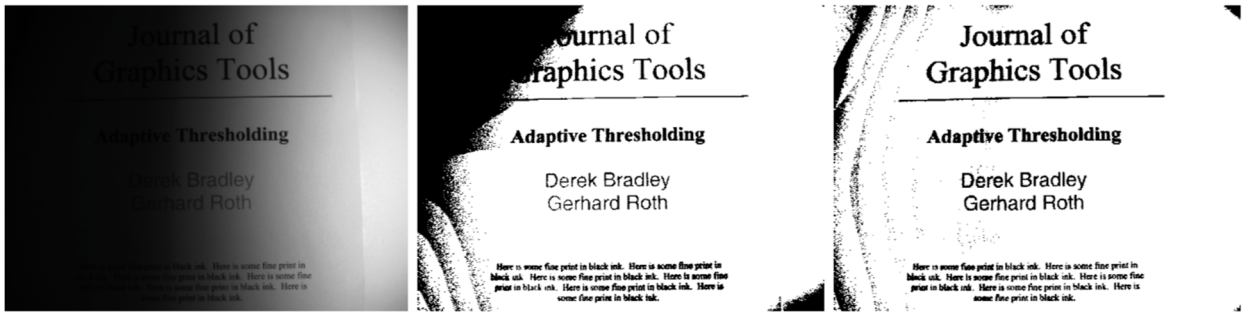


Figure 11 Original Image (Left), Previous Adaptive Threshold Method (Centre), Method Developed By Bradley Et. Al (Right) (Bradley, 2007)

Testing

The prototype was tested numerous times at the Dion-Ag Inc. shop in Boisbriand, QC. A hydraulic forage harvester connected to a tractor was pulled up beside a forage wagon. The forage harvester solenoid connections on the arm of the harvester were disconnected and connected into the Arduino circuit. The USB surveillance camera was attached to the chute arm, and connected to the Arduino and laptop. Once the set up was complete, the power to the forage chute was turned on, giving power to the machine vision system.

Most testing was done to see how the camera and software would identify the target from its surroundings. Through this testing the code was changed to optimally track the targets without tracking background interference.

Failure Mode Effects Analysis (FMEA)

In order to ensure that our proposed solution was a safe and responsible proposition. A failure modes and effects analysis was done in order to point out any flaws within the design that had gone unnoticed. The first step within this process was to determine what failure modes are most probable. When a complete list of possible failure modes has been assembled they are analyzed on a scale from 1 to 10 in categories of severity, occurrence, and detectability; this criteria can be found in Appendix C. These values are then multiplied to give the risk priority number (RPN). If the RPN is greater than 100 then it is determined that the failure mode is of significant importance and a resolution must be determined. Upon outlining recommended actions and resolution another analysis is done. If the RPN is now below 100 then the failure mode is seen as being resolved, if not further design modification is necessary. The analysis undertaken for the control of a forage harvester chute is shown in on the following page in Table 1.

FMEA

Item											Action Results				
Function	Potential Failure Mode	Potential Effect(s) of Failure	Severity	Potential Cause(s)/Mechanism(s) of Failure	Occurrence	Current Design Controls	Detectability	RPN	Recommended Action(s)	Responsibility and Target Completion Date	Actions Taken	Sev	Occ	Det	RPN
Camera	Loses connection	Input stops, code loses track of traget	2	USB disconnect or wire loom breakage	6	Hunting for last target, Alert for user	1	12							0
12V power to Adafruit board	Low voltage	Intermittent solenoid activation	8	System overload	3	Controlled by tractor system	6	144	Check available voltage, if too low need to look at tractor powersystem	Owner	If resolved reboot control system	8	1	6	48
Target Sighting	Stops finding target	The chute would stop moving	8	Software glitch, tractor speed	5	The chute should retrace it's movements back to last target sighting	2	80							0
Chute Control	Lose control	Uncontrollable movement of chute	9	Arduino malfunction	2	Operator system overide	2	36							0
Machine Solenoid	Control to solenoid fails	One axis of movement stops	7	Valve breakage due to shortened actuation sequence	6	Use design minimum actuation sequence frequency	2	84							0
Camera Mount	Breakage	The camera would no longer view the correct area	8	Loss of magnet, loss of contact with chute arm, physical disturbance	2	Bolted and held with 70lb force magnet	1	16							0
Target	Loss of target	Sighting system would stop working	7	Support breaks	4	The chute should retrace it's movements back to last target sighting	5	140	Replace/reinforce target	Owner	Target replaced/reinforced	7	2	5	70
Target Code	Locates and tracks area that is not a target	Chute would be directed in wrong direction	8	Bad lighting, similar target colour and surroundings	5	Code looks for most sqaure obejct of correct colour	6	240	Recalibrate targets with a smaller range colour frequency	Owner	Target recalibrated to more accurate range	8	2	6	96

Uncertainty Calculation

To determine the uncertainty of the camera testing was done. The camera was pointed at a blank area and the centroid of the image was continuously captured and recorded for 60 seconds. The equation used is as follows:

$$u = \frac{\sigma}{\sqrt{n}} = \sqrt{\frac{1}{n(n-1)}\sum_{i=1}^{n}(x_i - \mu)^2}$$

Where: σ = standard deviation; n = number of values; x_i = sample value; μ = mean

Using Microsoft excel to calculate the uncertainty of all 465 points taken, the results are as follows:

Table 1 Uncertainty Calculation Results

Horizontal Uncertainty	Vertical Uncertainty
0.055441717	0.04459032
Combined uncertainty	
0.042560549	

It was determined that the camera is accurate enough for this application.

Cost Analysis

The current design is a prototype and a proof of concept. The objective of the completed design is to demonstrate a convenient possibility to incorporate machine vision while operating a pull-type forage harvester. The prices listed are prototyping costs, and will not represent the final product which the client, Dion, will later pursue. The goal is to estimate the cost feasibility of integrating such technology on pull-type forage harvesters made by Dion-Ag Inc.

To examine whether or not this design system could possibly be produced for $8000, the cost of the prototype will be examined. The client originally provided a budget of $1500 (CAD) for the development of this prototype, and the total cost of the prototype ended up costing approximately $200.00 (CAD) as seen in Table 2. This did not include several components which were considered a free cost for the construction of this design. In the estimate, no software license was included due to the fact that as students the license fees of the used software are mostly offered by the academic institution, McGill University. The cost of MATLAB® and SOLIDWORKS® were not included in the table below. Conclusively, the cost of the prototype is very small when acknowledging the numerous resources McGill had to offer.

Table 2 Prototype Cost Estimate

Component	Cost per unit	Quantity	Cost
Adruino Uno	$24.95	1	$24.95
Adafruit Shield for Arduino	$19.95	1	$19.95
Camera (2.0 USB surveillance)	$56.43	1	$56.43
Camera Mount - 3D printer plastic	$30.66	1	$30.66
Camera Mount - Magnet	$7.00	3	$21.00
Target	$12.17	3	$36.50
Paint	$6.67	3	$20.00
Total			**$152.99**

Therefore, the cost of the prototype is well within the $8000 (CAD) budget limit of production as well as the $1500(CAD) prototype budget provided by Dion.

Recommendations

The final product will be produced by the client, Dion. Since this is a prototype there are components that can be improved for a final production system. Recommendations that would be beneficial to the full scale development of the current design are described below.

Microcontroller

The microcontroller used in the prototype was an Arduino Uno with a motor shield. If the product was to go through production, the recommended component would be an Arduino or a custom made circuit board. This custom circuit board suggestion depends on the economies of scale, this depends on the quantity of forage harvesters produce yearly.

Table 3 Microcontroller Estimate

Component	Cost per unit	Quantity	Cost
Single USB port	$1.26	1	$1.26
MOSFET 60V 27A	$0.95	6	$5.70
5V step-down voltage regulator	$5.99	1	$5.99
OSH park PCB base	$10.00	1	$10.00
Total			**$22.95**

Taking all figures into account, the $2 price difference between a custom board and an Arduino is not that immense for 14 harvesters. Equally, the cost of the PCB is an estimate and could be higher or lower in cost. Thus, this option should not be discarded immediately and further looked into by Dion.

Product cost

This section will provide a rough estimate for the optional product the client is seeking. The goal of the estimate is to verify if the budget limit will be respected. The cost of production will be similar to the prototype since several similar components have been recommended. Certain components such as the programming fees, the microcontroller and microprocessor will have to be reviewed. Dion will have to take into consideration some modifications with regards to the prototype.

For the microcontroller component, an estimate was done for the cost of one Printed Circuit Board. The cost is similar to the Arduino Uno therefore, not that advantageous. However, the cost may vary according to the economies of scale. According to Dion, 60 forage harvesters are sold per year and 10 out of them are they newest version with the hydraulics. The company thinks that the production of the hydraulic chute model will augment by 40% in the upcoming year.

Microprocessor

A personal laptop is used as the prototype microprocessor. For the implementation of this system into a product each pull-type forage harvester will have to be equipped with a microprocessor. Rather than bringing a laptop into the cab of a tractor, an Intel Atom is recommended. This provides faster processing space and an easy manipulative interface.

Programming

To create an optimal control program a computer programmer would be required. This extra cost would be worth the increased efficiency and reliability that the optimized program would

produce. Should the programmer choose to use python, the developed program could be made open source providing the client the ability to personally troubleshoot.

Targets

The current prototype targets are painted squares of plywood. The final material of the target can be decided by the client. The recommended material would be more durable to extended outdoor use than plywood. In this design the material of the target is not as critical as the colour and shape. Also, in case of bad lighting, the targets could be infrared or lit with LED lights so the targets provide their own light source. This would enable harvesting at all times of day.

Design Competition

This final design project will be entered in the *AGCO National Student Design Competition*. This competition is hosted by the American Society of Agricultural and Biological Engineers (ASABE). The registration for this competition is until April 27th 2016. In order to partake in this competition, one member of the design team must be registered as a student member of ASABE and have completed the design within their undergraduate studies (ASABE, 2016). The submitted projects for the *AGCO National Student Design Competition* require an engineering design that will entail a plan for a machine or system that will be useful to agriculture and related areas. A written report, drawings with clear details of the design, and proof of testing performed on the system with data acquisition needs to be included in the submission. The finalized submission will be completed by the beginning of 2016. The detailed design competition website can be found at http://www.asabe.org/awards-landmarks/student-awards,-competitions-scholarships/agco.aspx.

Conclusion

After more than a year of research and design, this machine vision system has been subject to the entire design cycle cumulating in a final design that reduces task complexity and cognitive strain upon forage harvester operators. Throughout the development, optimization, and testing of the design, small changes have been made to create a design the client can use as a foundation to the final development of a reliable product. In regards to the goal to develop a system to prove that machine vision can be used in a cost efficient manner to reduce the difficulty of forage offloading, this project has been successful. Final recommendations have been provided for the client to optimize the design base for production.

Acknowledgements

We would like to thank the following for their all support in our project:

- Dion team
    - Philippe Nieuwenhof & Frederic Renee-Laforest
- Dr. Adamchuck
- Dr. Clark
- Bharath Sudarsan
- Trevor Stanhope

Mentor Information

Dr. Adamchuk,
Associate Professor in Bioresource Engineering at McGill University,
viacheslav.adamchuk@mcgill.ca,
514-398-7657,
MS1-094 Macdonald-Stewart Building,
21111 Lakeshore Road
Ste. Anne de Bellevue, Quebec
H9X 3V9

Dr. Adamchuk provided guidance on the best system to use for the final design. Through his personal assistance along with the assistance of many members of his lab this project was able to be completed. Dr. Adamchuk provided much of the hardware used including the camera with USB wires for testing purposes and access to a 3D printer.

Client Information

Philippe Nieuwenhof,
Research and Development Engineer at Dion-Ag Inc.
phil@dion-ag.com
450-437-3449 ext. 237
429, Côte Sud
Boisbriand, Québec,
J7E 4H5.

Frederic Rene-Laforest
Engineer in Training at Dion-Ag Inc.,
fred@dion-ag.com
450-437-3449 ext. 236
429, Côte Sud
Boisbriand, Québec,
J7E 4H5.

Philippe and Fred were always very welcoming and open to the many ideas that were presented to them. Throughout the process the Dion team was more than willing to help with any questions and provide everything that was needed. Without access to their equipment and their knowledge this project would not have been realized.

References

Adafruit. Adafruit Motor/Stepper/Servo Shield for Arduino. Available at: https://www.adafruit.com/products/1438. Accessed 14 November 2015.

ARDUINO. ARDUINO UNO REV3. Available at: http://store-usa.arduino.cc/products/a000066. Accessed 15 November 2015.

Bradley, D., & Roth, Gerhard. 2007. Adaptive Thresholding Using the Integral Image

Clemente, S., and K. Teasdale. 1987. Understanding and Using Power MOSFET Reliability Data. International Rectifier.

Cooper-Martin, E. 1994. Measures of cognitive effort. *Marketing Letters* 5(1): 43-56.

Digi-Key. Electronic Components. Available at: http://www.digikey.ca/product-search/en?vendor=0&keywords=screw+connectors. Accessed 16 November 2015.

Electronics, S. P-Channel MOSFET. Available at: https://www.sparkfun.com/products/10349. Accessed 15 November 2015.

Intel. Intel Atom Processor. Available at: http://www.intel.com/content/www/us/en/processors/atom/atom-processor.html. Accessed 16 November 2015.

Mathworks. 2015. Image Processing Toolbox User's Guide. Natick, Massachusetts. The Mathworks, inc.

Nicholls, A., L. Bren, et al. (2004). Harvester productivity and operator fatigue: working extended hours. *International journal of forest engineering* 15(2): 57-65.

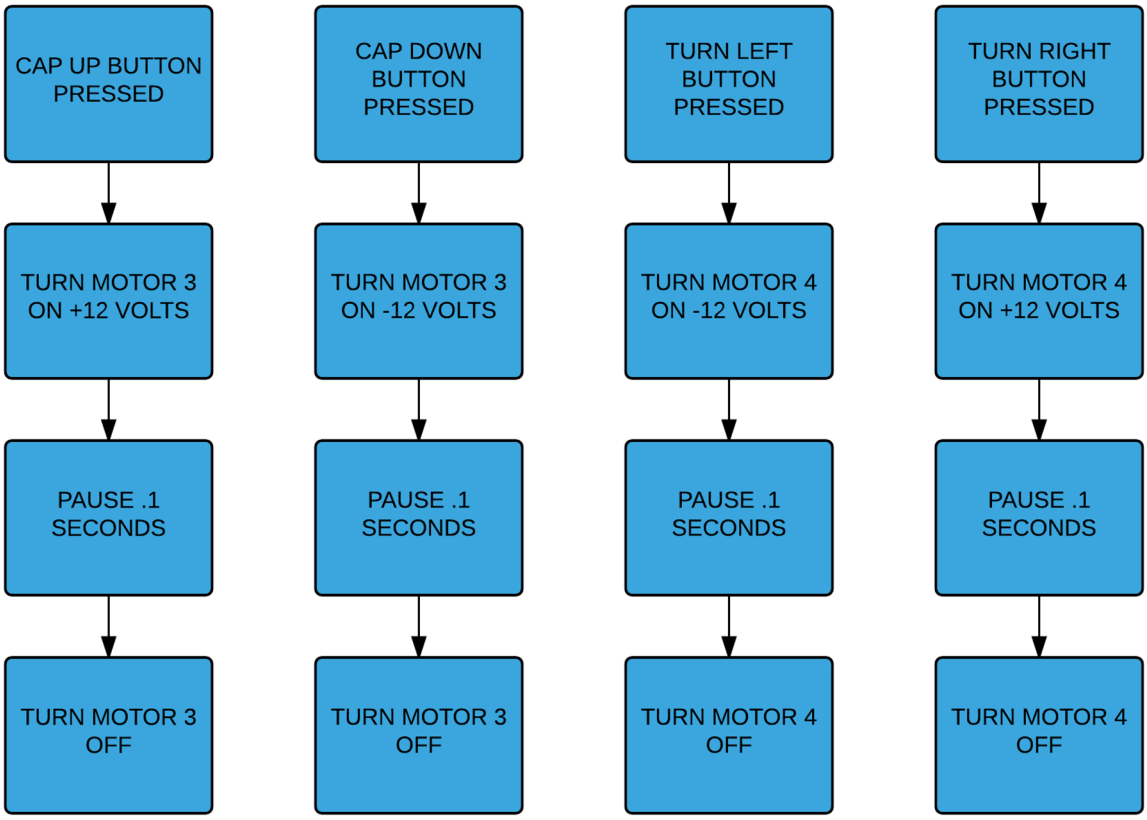
Staples. MakerBot 1.75 mm PLA Filament. Available at: http://www.staples.ca/en/MakerBot-175-mm-PLA-Filament-Small-Spool-05-lb-True-Blue/product_1490362_2-CA_1_20001?kpid=1490362&cid=PS:SBD:GS:n:n:SBD:58:21800&gclid=Cj0KEQiAg7ayBRD8qqSGt-fj6uYBEiQAucjOwfKABV4IHd3Pp4lZgCJQ1f8CGOOeIwGaVTbLmiZUaHYaAkG88P8HAQ. Accessed 13 November 2015.

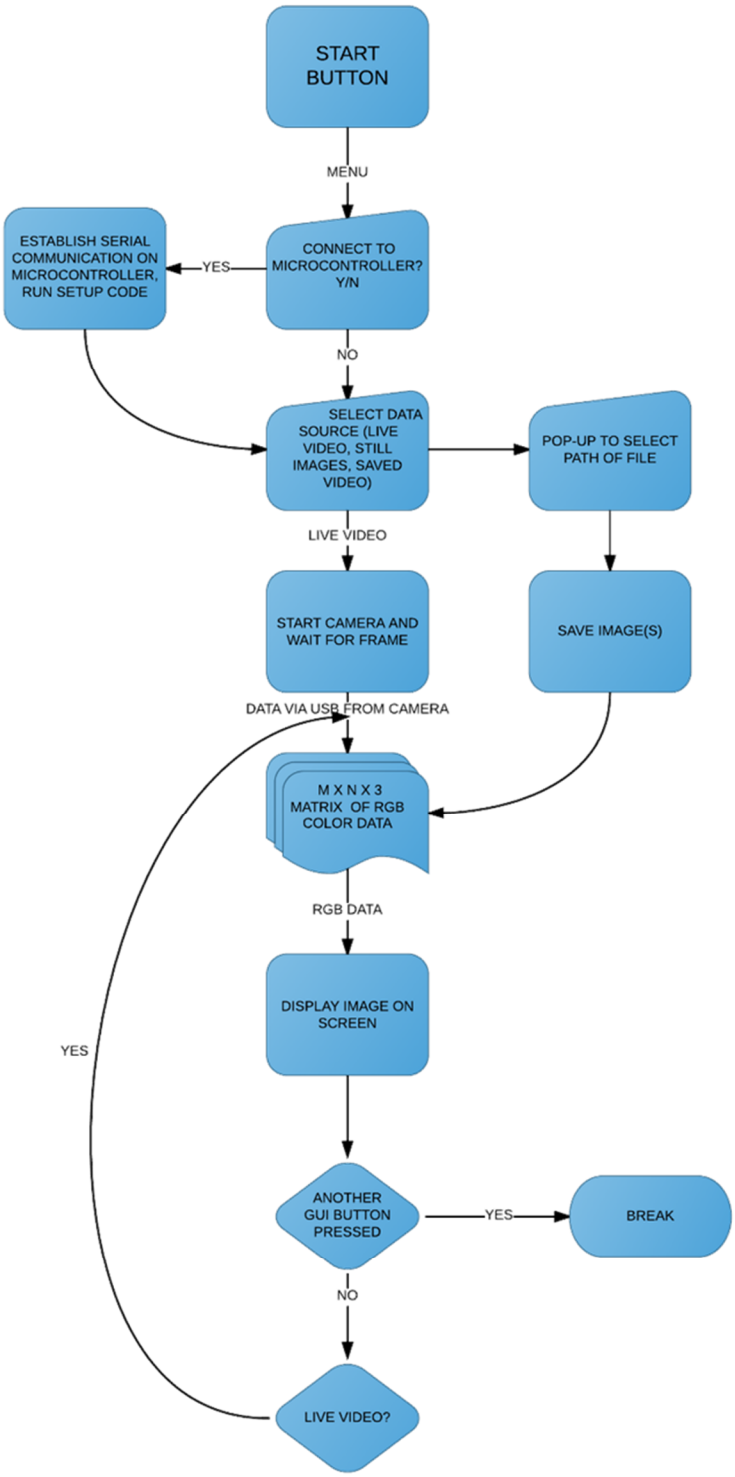
Toshiba. 2012. Toshiba Bi-CD Integrated Circuit. Toshiba Corporation.

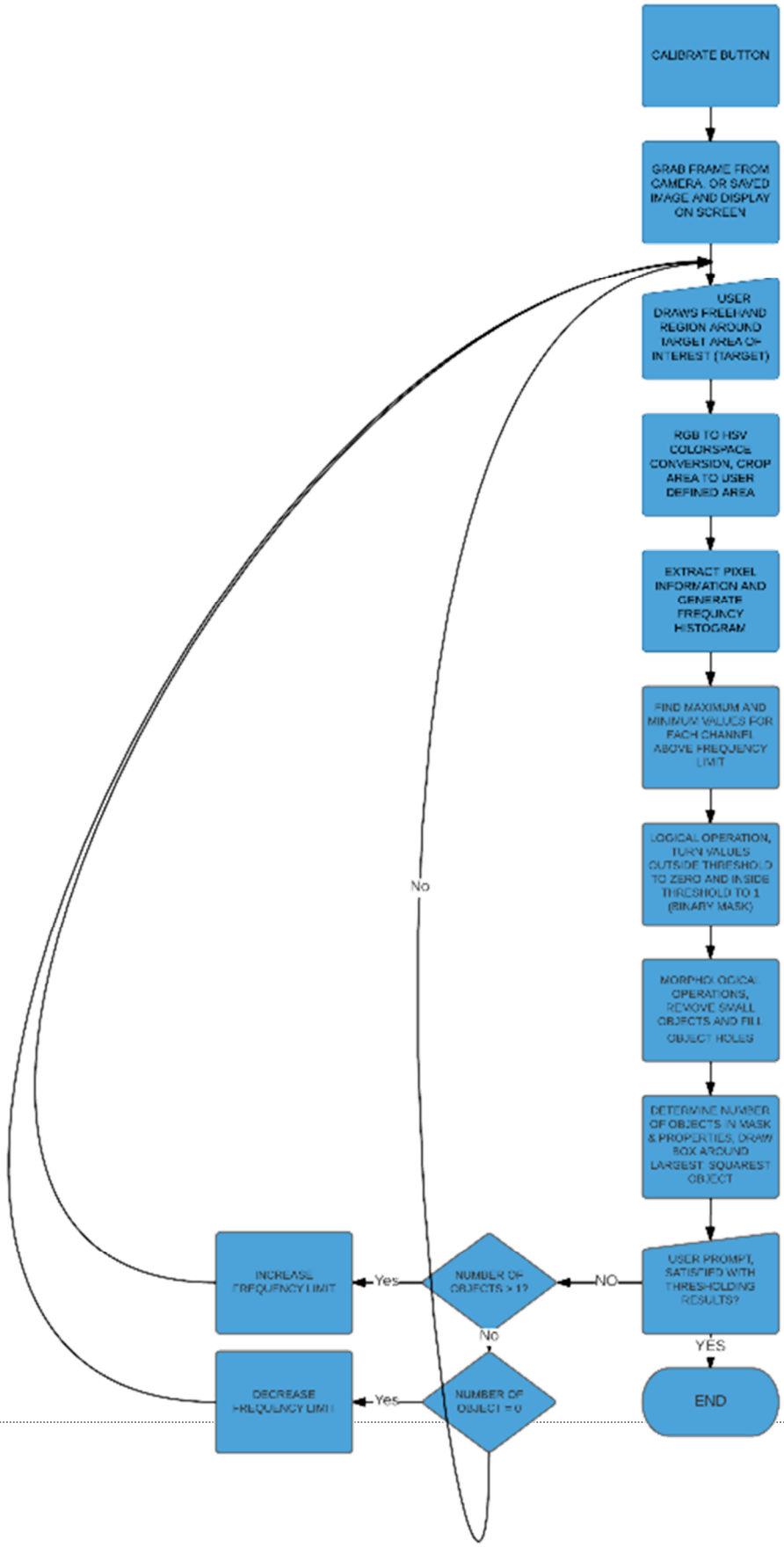
Tyco Electronics Corporation. Relay Contact Life. Winston-Salem, N.C.

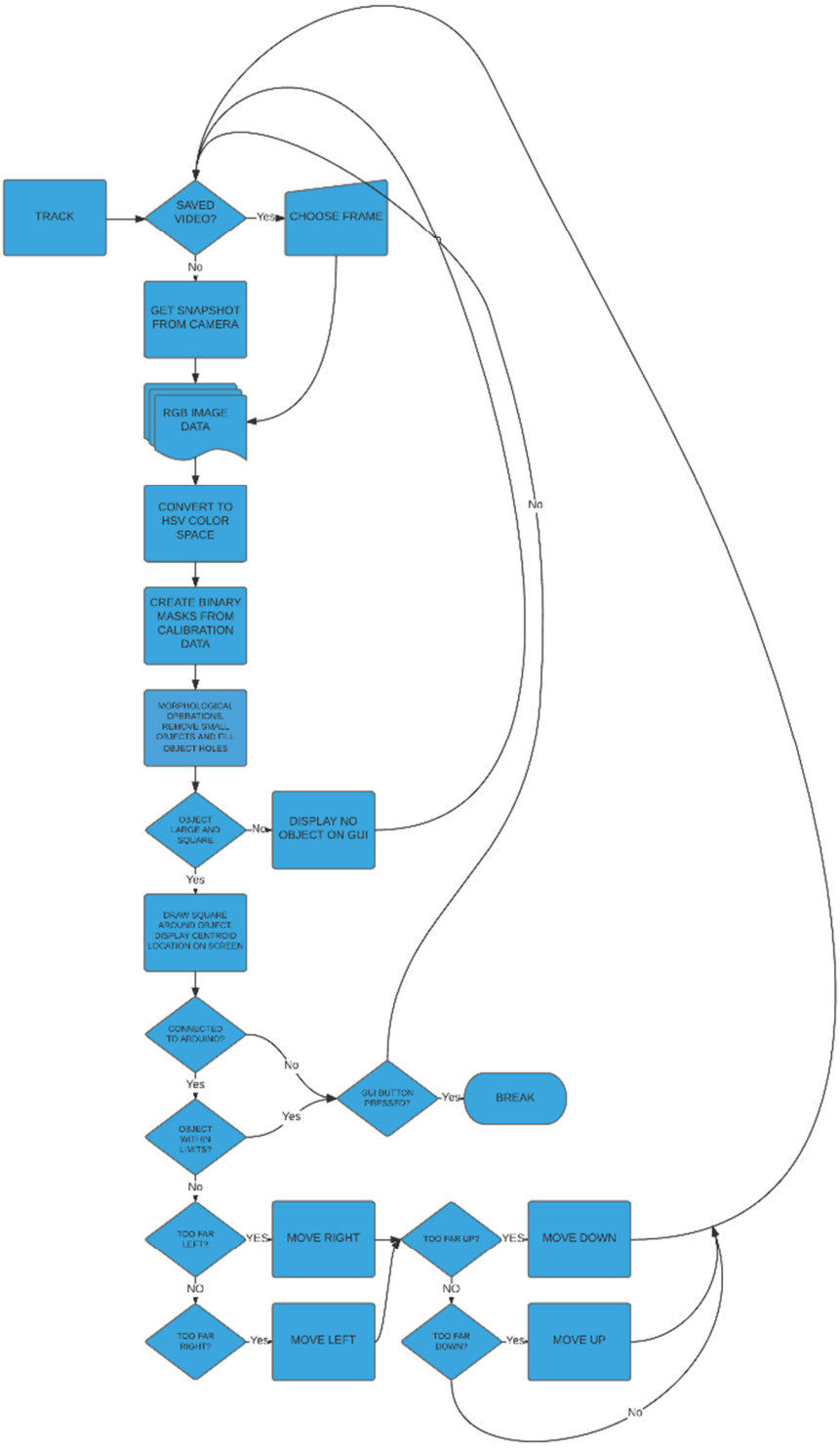
Appendices

Appendix A: Flowcharts




START BUTTON
MENU
CONNECT TO MICROCONTROLLER? Y/N
YES
ESTABLISH SERIAL COMMUNICATION ON MICROCONTROLLER, RUN SETUP CODE
NO
SELECT DATA SOURCE (LIVE VIDEO, STILL IMAGES, SAVED VIDEO)
POP-UP TO SELECT PATH OF FILE
LIVE VIDEO
START CAMERA AND WAIT FOR FRAME
SAVE IMAGE(S)
DATA VIA USB FROM CAMERA
M X N X 3 MATRIX OF RGB COLOR DATA
RGB DATA
DISPLAY IMAGE ON SCREEN
YES
ANOTHER GUI BUTTON PRESSED
YES
BREAK
NO
LIVE VIDEO?


CALIBRATE BUTTON
GRAB FRAME FROM CAMERA, OR SAVED IMAGE AND DISPLAY ON SCREEN
USER DRAWS FREEHAND REGION AROUND TARGET AREA OF INTEREST (TARGET)
RGB TO HSV COLORSPACE CONVERSION, CROP AREA TO USER DEFINED AREA
EXTRACT PIXEL INFORMATION AND GENERATE FREQUNCY HISTOGRAM
FIND MAXIMUM AND MINIMUM VALUES FOR EACH CHANNEL ABOVE FREQUENCY LIMIT
LOGICAL OPERATION, TURN VALUES OUTSIDE THRESHOLD TO ZERO AND INSIDE THRESHOLD TO 1 (BINARY MASK)
MORPHOLOGICAL OPERATIONS, REMOVE SMALL OBJECTS AND FILL OBJECT HOLES
DETERMINE NUMBER OF OBJECTS IN MASK & PROPERTIES, DRAW BOX AROUND LARGEST SQUAREST OBJECT
USER PROMPT, SATISFIED WITH THRESHOLDING RESULTS?
NO
YES
END
NUMBER OF OBJECTS > 1?
Yes
No
INCREASE FREQUENCY LIMIT
NUMBER OF OBJECT = 0
Yes
DECREASE FREQUENCY LIMIT
No


TRACK
SAVED VIDEO?
Yes
CHOOSE FRAME
No
GET SNAPSHOT FROM CAMERA
RGB IMAGE DATA
CONVERT TO HSV COLOR SPACE
CREATE BINARY MASKS FROM CALIBRATION DATA
MORPHOLOGICAL OPERATIONS, REMOVE SMALL OBJECTS AND FILL OBJECT HOLES
OBJECT LARGE AND SQUARE
No
DISPLAY NO OBJECT ON GUI
Yes
DRAW SQUARE AROUND OBJECT, DISPLAY CENTROID LOCATION ON SCREEN
CONNECTED TO ARDUINO?
No
Yes
GUI BUTTON PRESSED?
Yes
BREAK
OBJECT WITHIN LIMITS?
Yes
No
TOO FAR LEFT?
YES
MOVE RIGHT
TOO FAR UP?
YES
MOVE DOWN
NO
NO
TOO FAR RIGHT?
Yes
MOVE LEFT
TOO FAR DOWN?
Yes
MOVE UP
No
No

Contents

- dion_2.m: This program allows automated tracking of a Dion-ag. Stinger forage harvester. The program can interactively threshold user selected target area, have user input digital control of the harvester on 2 axes and allows various file types to be loaded for further development. This version works with an arduino uno and adafruit motor-shield with 12 volt output. Any 640 X 480 USB camera may be used. Arduino for Matlab and webcam suport packages required as well as matlab version with image processing toolbox installed
- --- Executes on button press in Start_button.
- --- Executes on button press in Front_target.
- --- Executes on button press in Calibrate_middle.
- --- Executes on button press in Calibrate_rear.
- --- Executes on button press in Stop_Button.
- --- Executes on button press in Track.
- --- Executes on button press in capup.
- --- Executes on button press in capdown.
- --- Executes on button press in turnleft.
- --- Executes on button press in turnright.

dion_2.m: This program allows automated tracking of a Dion-ag. Stinger forage harvester. The program can interactively threshold user selected target area, have user input digital control of the harvester on 2 axes and allows various file types to be loaded for further development. This version works with an arduino uno and adafruit motor-shield with 12 volt output. Any 640 X 480 USB camera may be used. Arduino for Matlab and webcam suport packages required as well as matlab version with image processing toolbox installed

```
% last modified December 6 - 2015
% for BREE 495, written, tested and developed by: Brett Bennett, Meaghan
% Dustin, Stephen McGuire and Yasmeen Hitti


function varargout = dion_2(varargin)
% DION_2 MATLAB code for dion_2.fig
%      DION_2, by itself, creates a new DION_2 or raises the existing
%      singleton*.
%
%      H = DION_2 returns the handle to a new DION_2 or the handle to
%      the existing singleton*.
%
%      DION_2('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in DION_2.M with the given input arguments.
%
%      DION_2('Property','Value',...) creates a new DION_2 or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before dion_2_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to dion_2_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help dion_2

% Last Modified by GUIDE v2.5 15-Oct-2015 12:09:47

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
```

```
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @dion_2_OpeningFcn, ...
                   'gui_OutputFcn',  @dion_2_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before dion_2 is made visible.
function dion_2_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.axes1.Visible = 'off';

% varargin   command line arguments to dion_2 (see VARARGIN)

% Choose default command line output for dion_2
handles.output = hObject;
% Update handles structure
guidata(hObject, handles);

% UIWAIT makes dion_2 wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = dion_2_OutputFcn(hObject, eventdata, handles)
```

```
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;
```

--- Executes on button press in Start_button.

alows user to set up data source and begin displaying images on the GUI

```
function Start_button_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to Start_button (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Create variables to be shared
```

```
global RGB
global Cam
global Vid_choice
global pic
global shield
global dcm3
global dcm4


set(handles.Start_button,'UserData','') % condition to check if another
%button has been pressed

% prompt to connect to arduino
arduino_start = menu('Connect to Arduino?', 'Yes','No');
% if user wants to connect to arduino, execute
if arduino_start ==1
    % arduino set up function with port id and hardware arguments
    Arduino = arduino('/dev/cu.usbmodem1411', 'Uno', 'Libraries', 'Adafruit\MotorShieldV2')
    % specify hardware
 shield = addon(Arduino, 'Adafruit\MotorShieldV2')
 % specify that motors 3 and 4 will be used and there associated variables
 dcm3 = dcmotor(shield, 3)
 dcm4 = dcmotor(shield, 4)
end

% menu to select live video, still images or saved video to be analyzed
Vid_choice = menu('Select Video options','Live Video From Camera',...
    'Saved Video','Still Images');

% executes for live video
if Vid_choice == 1
    Cam = webcam('USB2.0 Camera'); % start camera

% loop executes as long as another button hasnt been pressed
% and displays video from camera on screen
    while strcmp(get(handles.Start_button,'UserData'),'Pressed') == 0
   % attempt to get frame from camera
   try

    RGB = snapshot(Cam); % get image from running camera
    imshow(RGB) % show the aquired image

   catch
       disp('error') % display error if webcam is not ready
   end

  % if another GUI button is pressed, stop the loop
  if strcmp(get(handles.Start_button,'UserData'),'Pressed') == 1
      break
  end % end condition

    end % end loop
  % executes  for saved video choice
elseif Vid_choice ==2
    path = uigetfile(); % allow user to select desired file
    reader = VideoReader(path); % create object to read video
    k = 1; % initialize counter
for count = 1:40 % number of frames to read
    pic(k).cdata = readFrame(reader); % structure variable, saves all frames
    k = k+1; % increase counter
    disp(k)
end % end condition
disp('done') % display on screen when done loading

elseif Vid_choice ==3 % choice to load saved images
```

```
    path = uigetfile(); % user selects file
    RGB = imread(path); % read selected image
    imshow(RGB) % show image
end % end condition
```

--- Executes on button press in Front_target.

interactive thresholding set up for front target

```
function Front_target_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to Front_target (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% clear the axes
cla
% load required global variables
global RGB
global pic
global Cam
global Vid_choice
global channel1Min
global channel1Max
global channel2Min
global channel2Max
global channel3Min
global channel3Max

count_front = 10; % initialize minimum frequency for repeated pixels, to
% be used later in thresholding.

set(handles.Start_button,'UserData','Pressed') % update handles to indicate
% button has been pressed

if Vid_choice == 2 % executes for saved video
    frame = input('input desired frame'); % user input for desired frame in
    % saved video structure
    RGB = pic(frame).cdata; % set picture as frame of interest
    imshow(RGB) % show image
        pause(1) % give user a chance to see image


I = rgb2hsv(RGB); % convert to HSV color space
elseif Vid_choice ==1 % executes for live video
    RGB = snapshot(Cam); % get still image from camera
    imshow(RGB) % display image
    pause(1) % allow user cahnce to see image
    I = rgb2hsv(RGB); % convert to HSV color space
    imshow(I) % show HSV image
    pause(1) % allow user to see HSV image


elseif Vid_choice ==3 % executes for saved still images
    I = rgb2hsv(RGB); % convert to HSV colorspace
    imshow(I); % show HSV image
    pause(1); % allow user to see image
end % end conditions

k = 1; % counter allows execution until break command
while k>0 % while true
% call dionalibrate function to set up thresholds
```

```
[channel1Min,channel1Max,channel2Min,channel2Max,channel3Min,channel3Max]=...
DionCalibrate(RGB,count_front);

% Create mask based on chosen histogram thresholds, logical operations to,
% create binary mask

BW_Front = (I(:,:,1) >= channel1Min ) & (I(:,:,1) <= channel1Max) & ...
    (I(:,:,2) >= channel2Min ) & (I(:,:,2) <= channel2Max) & ...
    (I(:,:,3) >= channel3Min ) & (I(:,:,3) <= channel3Max);
% Initialize output masked image based on input image.
maskedRGBImage = RGB;
% Set background pixels where BW is false to zero. colour mask, used in
% debugging
maskedRGBImage(repmat(~BW_Front,[1 1 3])) = 0;
% create square 10 pixel structuring element for morphological operations
se = strel('square',10);
% show the binary image
imshow(BW_Front)
% allow user time to see image
pause(1)
% dilate image with structuring element
BW_Front = imdilate(BW_Front,se);
% show image again
imshow(BW_Front)
% wait for 1 second
pause(1)
% remove objects smaller than 1000 pixels
BW_Front = bwareaopen(BW_Front, 1000);
% show resulting binary image
imshow(BW_Front)
% wait for 1 second
pause(1)
% fill interior object holes smaller than 100 pixels
BW_Front = bwmorph(BW_Front,'Fill',100);
% extract all properties of all objects
props = regionprops(BW_Front,'all');
% variable corresponding to object area
area = [props.Area];
% find the index of the object that is the largest
[large,Index] = max(area);
% variable for the major axis of the largest object
majoraxis = props(Index).MajorAxisLength;
% variable for the minor axis of the largest object
minoraxis = props(Index).MinorAxisLength;
% compute the ratio of the major to minor axis to determine how square the
% object is
axis_ratio = majoraxis ./ minoraxis
% allowable squarness range condition
if axis_ratio >= 1 & axis_ratio < 1.5
    % get the bounding box of the object
Bound = [props(Index).BoundingBox];
% determine how many objects are in the binary image
num_comps = bwconncomp(BW_Front);
num_comps = num_comps.NumObjects
% condition for muliple mask objects
if num_comps >= 2;
    % increase histogram minimum frequency requirement
    count_front = count_front + 100
elseif num_comps ==0
    % no obects detected, decrease histogram frequency requirement
    count_front = 20;
    % single object in mask
elseif num_comps ==1
    % dilate again with previous structure element
    BW_Front = imdilate(BW_Front,se);
```

```
% end number of object conditions
end
% show image
 imshow(BW_Front)
 % wait for 1 second
pause(1)
end % end squareness condition
try % execute if possible (there is at least one square object)
rectangle('position', Bound,'EdgeColor','r','LineWidth',2) % draw rectangle
catch % do nothing if error is encountered
end % end try/catch
Calib_choice = menu('Done Calibrating Front?', 'Yes', 'No'); % ask user if
% they are satisfied with thresholding results
if Calib_choice == 1 % execute if yes
    set(handles.Start_button,'UserData','') % allow start button to begin
    % showing video again
    break % break from while loop
end % end prompt condition
end % end of while loop
```

--- Executes on button press in Calibrate_middle.

interactive thresholding set up for middle target

```
function Calibrate_middle_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to Calibrate_middle (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% clear axes
cla
% load required variables
global RGB
global Cam
global Vid_choice
global midchannel1Min
global midchannel1Max
global midchannel2Min
global midchannel2Max
global midchannel3Min
global midchannel3Max
global pic

% initialize minimum frequency for repeated pixels, to
% be used later in thresholding.
count_mid = 10;
% update handles to indicate button has been pressed
set(handles.Start_button,'UserData','Pressed')
% executes for saved video
if Vid_choice == 2
    % user selects frame of video to use
    RGB = pic((input('select frame'))).cdata;
    % convert to HSV colorspace
I = rgb2hsv(RGB);
% display image
imshow(RGB)
% allow user to see image for 1 second
pause(1);

% executes for live video
elseif Vid_choice ==1
```

```
    % get still image from camera
    RGB = snapshot(Cam);
    % display image
    imshow(RGB)
    % wait for 1 second
    pause(1)
    % convert to HSV color space
    I = rgb2hsv(RGB);

    % executes for still images
elseif Vid_choice ==3
    % convert to HSV color space
    I = rgb2hsv(RGB);
end % end conditions

k = 1; % counter for while loop, execute forever until commanded to break
while k>0
    % call dion calibrate function to set up middle target thresholds
    [midchannel1Min,midchannel1Max,midchannel2Min,midchannel2Max,midchannel3Min,midchannel3Max]=...
DionCalibrate(RGB,count_mid);

% Create binary mask based on chosen histogram thresholds, logical
% operations

BW_mid = (I(:,:,1) >= midchannel1Min ) & (I(:,:,1) <= midchannel1Max) & ...
    (I(:,:,2) >= midchannel2Min ) & (I(:,:,2) <= midchannel2Max) & ...
    (I(:,:,3) >= midchannel3Min ) & (I(:,:,3) <= midchannel3Max);

% Initialize output masked image based on input image.
maskedRGBImage = RGB;
% Set background pixels where BW is false to zero. Creates coloured mask,
% with colours outside mask black, used for debugging and optimization only
maskedRGBImage(repmat(~BW_mid,[1 1 3])) = 0;
% create square structuring element 10 pixels by 10 pixels
se = strel('square',10);
% show binary image
imshow(BW_mid)
% wait for 1 second
pause(1)
% dilate image using structuring element
BW_mid = imdilate(BW_mid,se);
% remove objects smaller than 1000 pixels in area
BW_mid = bwareaopen(BW_mid,1000);
% display image again
imshow(BW_mid)
% wait for 1 second
pause(1)
% fill interior areas of objects with holes 100 pixels or smaller
BW_mid = bwmorph(BW_mid,'Fill',100);
% get all properties of all objects
props = regionprops(BW_mid,'all');
% variable for area of objects
area = [props.Area];
% get index of largest object
[large,Index] = max(area);
majoraxis = props(Index).MajorAxisLength;
% variable for the minor axis of the largest object
minoraxis = props(Index).MinorAxisLength;
% compute the ratio of the major to minor axis to determine how square the
% object is
axis_ratio = majoraxis ./ minoraxis
% allowable squarness range condition
if axis_ratio >= 1 & axis_ratio < 1.5
    % get the bounding box of the object
Bound = [props(Index).BoundingBox];
```

```
% determine how many objects are in the binary image
num_comps = bwconncomp(BW_mid);
num_comps = num_comps.NumObjects
% condition for muliple mask objects
if num_comps >= 2;
    % increase histogram minimum frequency requirement
    count_mid = count_mid + 100
elseif num_comps ==0
    % no obects detected, decrease histogram frequency requirement
    count_mid = 20;
    % single object in mask
elseif num_comps ==1
    % dilate again with previous structure element
    BW_mid = imdilate(BW_mid,se);
% end number of object conditions
end
% show image
 imshow(BW_mid)
 % wait for 1 second
pause(1)
end % end squareness condition
try % execute if possible (there is at least one square object)
rectangle('position', Bound,'EdgeColor','r','LineWidth',2) % draw rectangle
catch % do nothing if error is encountered
end % end try/catch
Calib_choice = menu('Done Calibrating Middle?', 'Yes', 'No'); % ask user if
% they are satisfied with thresholding results
if Calib_choice == 1 % execute if yes
    set(handles.Start_button,'UserData','') % allow start button to begin
    % showing video again
    break % break from while loop
end % end prompt condition
end % end of while loop
% clear axes
cla
```

--- Executes on button press in Calibrate_rear.

interactive thresholding set up for rear target

```
function Calibrate_rear_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to Calibrate_rear (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% clear axes
cla
% load required variables
global RGB
global Cam
global Vid_choice
global rearchannel1Min
global rearchannel1Max
global rearchannel2Min
global rearchannel2Max
global rearchannel3Min
global rearchannel3Max
global pic
% minimum required number of repeated pixels for threshold
count_rear = 10;
% update handles to indicate a button has been pressed
```

```
set(handles.Start_button,'UserData','Pressed')
% execute for saved video
if Vid_choice == 2
    % user input frame of interest
    RGB = pic((input('select frame'))).cdata;
    % display image
    imshow(RGB)
    % wait for 1 second
    pause(1)
    % convert to HSV color space
I = rgb2hsv(RGB);
% execute for live video
elseif Vid_choice ==1
    % get still image from camera
    RGB = snapshot(Cam);
    % display still image
    imshow(RGB)
    % wait for 1 second
    pause(1)
    % convert to HSV color space
    I = rgb2hsv(RGB);
% execute for still images
elseif Vid_choice ==3
    % convert to HSV color space
    I = rgb2hsv(RGB);
end
% counter for while loop, loop executes forever until comanded to break
k = 1;
while k>0
    % call dioncalibrate function to set up threhold values
    [rearchannel1Min,rearchannel1Max,rearchannel2Min,rearchannel2Max,rearchannel3Min,rearchannel3Max]=...
DionCalibrate(RGB,count_rear);

% Create binary mask based on chosen histogram thresholds, logical operations
BW_Rear = (I(:,:,1) >= rearchannel1Min ) & (I(:,:,1) <= rearchannel1Max) & ...
    (I(:,:,2) >= rearchannel2Min ) & (I(:,:,2) <= rearchannel2Max) & ...
    (I(:,:,3) >= rearchannel3Min ) & (I(:,:,3) <= rearchannel3Max);

% Initialize output masked image based on input image.
maskedRGBImage = RGB; % colorized mask for debugging only
% Set background pixels where BW is false to zero.
maskedRGBImage(repmat(~BW_Rear,[1 1 3])) = 0;
% create structuring element, 10 by 10 pixels square
se = strel('square',10);
% show binary image
imshow(BW_Rear)
% wait for 1 second
pause(1)
% dilate objects with structuring element
BW_Rear = imdilate(BW_Rear,se);
% display new binary image
imshow(BW_Rear)
% wait for 1 second
pause(1)
% fill holes in objects up to 100 pixels
BW_Rear = bwmorph(BW_Rear,'Fill',100);
% remove objects smaller than 1000 pixels
BW_Rear = bwareaopen(BW_Rear,1000);
% get properties of all objects
props = regionprops(BW_Rear,'all');
% area of objects
area = [props.Area];
% indexed location of largest object
[large,Index] = max(area);
majoraxis = props(Index).MajorAxisLength;
```

```
% variable for the minor axis of the largest object
minoraxis = props(Index).MinorAxisLength;
% compute the ratio of the major to minor axis to determine how square the
% object is
axis_ratio = majoraxis ./ minoraxis
% allowable squarness range condition
if axis_ratio >= 1 & axis_ratio < 1.5
    % get the bounding box of the object
Bound = [props(Index).BoundingBox];
% determine how many objects are in the binary image
num_comps = bwconncomp(BW_Rear);
num_comps = num_comps.NumObjects
% condition for muliple mask objects
if num_comps >= 2;
    % increase histogram minimum frequency requirement
    count_rear = count_rear + 100
elseif num_comps ==0
    % no obects detected, decrease histogram frequency requirement
    count_rear = 20;
    % single object in mask
elseif num_comps ==1
    % dilate again with previous structure element
    BW_Rear = imdilate(BW_Rear,se);
% end number of object conditions
end
% show image
 imshow(BW_Rear)
 % wait for 1 second
pause(1)
end % end squareness condition
try % execute if possible (there is at least one square object)
rectangle('position', Bound,'EdgeColor','r','LineWidth',2) % draw rectangle
catch % do nothing if error is encountered
end % end try/catch
Calib_choice = menu('Done Calibrating Rear?', 'Yes', 'No'); % ask user if
% they are satisfied with thresholding results
if Calib_choice == 1 % execute if yes
    set(handles.Start_button,'UserData','') % allow start button to begin
    % showing video again
    break % break from while loop
end % end prompt condition
end % end of while loop
% clear axes
cla
```

--- Executes on button press in Stop_Button.

stop tracking, clear axes and close camera

```
function Stop_Button_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to Stop_Button (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global Cam
set(handles.Start_button,'UserData','Pressed')
set(handles.Track,'UserData','')

Cam = [];
cla
```

--- Executes on button press in Track.

when this button is pressed, 3 masks are created for the three targets to be tracked and the centroid of respective targets displayed on the GUI, for the current version, only the rear target is used to control the motion of the harvester without switching control to other targets, to be developed in a later version.

```
function Track_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to Track (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

cla % clear current axes
% load required variables
global Cam
global pic
global Vid_choice
global RGB
global channel1Min
global channel1Max
global channel2Min
global channel2Max
global channel3Min
global channel3Max
global midchannel1Min
global midchannel1Max
global midchannel2Min
global midchannel2Max
global midchannel3Min
global midchannel3Max
global rearchannel1Min
global rearchannel1Max
global rearchannel2Min
global rearchannel2Max
global rearchannel3Min
global rearchannel3Max
global dcm3
global dcm4
% allows the start button to know that another button has been pressed to
% break the image display loop
set(handles.Start_button,'UserData','Pressed')
% initiate track loop condition
set(handles.Track,'UserData','Pressed')
% while loop continues until another button is pressed
while strcmp(get(handles.Track,'UserData'),'Pressed') == 1
  % if live video get a snapshot
if Vid_choice == 1
    RGB = snapshot(Cam);
    % if it is saved video get the desired frame
elseif Vid_choice ==2
    RGB = pic(input('selectframe')).cdata;

end
% hold current axes
 hold on
 % convert to hsv colorspace
    I = rgb2hsv(RGB);


% create a logical matrix from the threshold values from calibration of the
% front target & the current camera image
BW_Front = (I(:,:,1) >= channel1Min ) & (I(:,:,1) <= channel1Max) & ...
    (I(:,:,2) >= channel2Min ) & (I(:,:,2) <= channel2Max) & ...
```

```
    (I(:,:,3) >= channel3Min ) & (I(:,:,3) <= channel3Max);
% create a logical matrix from the threshold values from calibration of the
% middle target & the current camera image
BW_Middle = (I(:,:,1) >= midchannel1Min ) & (I(:,:,1) <= midchannel1Max) & ...
    (I(:,:,3) >= midchannel3Min ) & (I(:,:,3) <= midchannel3Max);
% create a logical matrix from the threshold values from calibration of the
% rear target & the current camera image
BW_Rear = (I(:,:,1) >= rearchannel1Min ) & (I(:,:,1) <= rearchannel1Max) & ...
     (I(:,:,2) >= rearchannel2Min ) & (I(:,:,2) <= rearchannel2Max) & ...
    (I(:,:,3) >= rearchannel3Min ) & (I(:,:,3) <= rearchannel3Max);
% Initialize output masked image based on input image. for debugging use
% only
maskedRGBImage = RGB;
% specify the center of the image horizontally based on resolution of the
% camera
center_screen_horiz = 320;
% specify the center of the image vertically based on resolution of the
% camera
center_screen_vert = 240;
% specify furthest left allowable deviation from center of screen
left_limit = center_screen_horiz - 50;
% specify furthest right allowable deviation from center of screen
right_limit = center_screen_horiz + 50;
% specify furthest up allowable deviation from center of screen
upper_limit = center_screen_vert + 40;
% specify furthest down allowable deviation from center of screen
lower_limit = center_screen_vert - 40;
% ensure bounding rectangles created from previous iteration are not
% visible
rect_rear.Visible = 'off';
rect_mid.Visible = 'off';
rear_marker.Visible = 'off';
% delete previous object centroid locations
Front_cent = [];
Middle_cent = [];
Rear_cent = [];
% create second hsv image, used if histogram equalization is performed.
I1 =I;
% try statement, if object is not detected, will not throw an error that
% crashes the program
try
  % square structuring element for morphlogical operations on image
se = strel('square',10);
% remove objects smaller than 1000 pixels
BW_Front = bwareaopen(BW_Front, 1000);
% dilate objects using structuring element
BW_Front = imdilate(BW_Front,se);
% fill object holes
BW_Front = bwmorph(BW_Front,'Fill',100);
% extract object properties of interest
props_front = regionprops(BW_Front,'MajorAxisLength','MinorAxisLength','Centroid','Area','BoundingBox','ConvexHull');
% area of objects in pixels
area = [props_front.Area];
% locate object with largest area
[large,Index] = max(area);
% calculate major axis of largest object
majoraxis = props_front(Index).MajorAxisLength;
% calculate minor axis of largest object
minoraxis = props_front(Index).MinorAxisLength;
% variable for centroid of largest object
Front_cent = [props_front(Index).Centroid];
% calculate ratio of major axis to minor axis, to tell how square the
% object is
axis_ratio = majoraxis / minoraxis;
% x position of centroid
```

```
center_x = Front_cent(1);
% y position of centroid
center_y = Front_cent(2);

% execute if the object is square, ie it is the target
if axis_ratio >= 1 && axis_ratio < 2
% get the bounding box of the image
Bound_front = [props_front(Index).BoundingBox];
% display the location of the centroid on the gui
set(handles.Front,'String',Front_cent)

else % not square object, display that it is not tracking the target
    set(handles.Front,'String','no object')
end
% catch errors
catch ME
error_text = ME.identifier; % error identification
set(handles.error,'String',error_text) % show in error text box
end

try
    % 10 by ten structuring element
se = strel('square',10);
% dilate mask with structuring element
BW_Middle = imdilate(BW_Middle,se);
% remove objects smaller than 1000 pixels
BW_Middle = bwareaopen(BW_Middle, 1000);
% fill holes
BW_Middle = bwmorph(BW_Middle,'Fill',100);
% get desired object properties
props_middle = regionprops(BW_Middle,'MajorAxisLength','MinorAxisLength','Centroid','Area','BoundingBox');
% area of objects
area = [props_middle.Area];
% indexed location of largest object
[large,Index] = max(area);
% major axis of largest object
majoraxis = props_middle(Index).MajorAxisLength;
% minor axis of largest object
minoraxis = props_middle(Index).MinorAxisLength;
% centroid of largest object
Middle_cent = props_middle(Index).Centroid;
% x component of centroid
center_x = Middle_cent(1);
center_y = Middle_cent(2);
% squareness of object
axis_ratio = majoraxis / minoraxis;
% execute if object is square
if axis_ratio >= 1 && axis_ratio < 1.5
    % get bounding box
 Bound_mid = [props_middle(Index).BoundingBox];
 % draw rectangle around object
rect_mid = rectangle('position', Bound_mid,'EdgeColor','r','LineWidth',20);
% display object centroid
set(handles.Middle,'String',Middle_cent)
 % if not square show no object
else
    set(handles.Middle,'String','no object')
end % end square condition

catch % catch errors
set(handles.Middle,'String','no object') % if error show no object text
end

try % if object is not detected, program will continue without crashing
% square structuring element
```

```
se = strel('square',10);
% dilate binary mask
BW_Rear = imdilate(BW_Rear,se);
% remove objects smaller than 1000 pixels
BW_Rear = bwareaopen(BW_Rear, 1000);
% fill holes up to 100 pixels in size
BW_Rear = bwmorph(BW_Rear,'Fill',100);
% get desired object properties
props_rear = regionprops(BW_Rear,'MajorAxisLength','MinorAxisLength','Centroid','Area','BoundingBox');
% area of object
area = [props_rear.Area];
% indexed location of largest object in masl
[large,Index] = max(area);
% major axis of largest object
majoraxis = props_rear(Index).MajorAxisLength;
% minor axis of largest object
minoraxis = props_rear(Index).MinorAxisLength;
% centroid of largest object
Rear_cent = props_rear(Index).Centroid;
% squarness of largest object
axis_ratio = majoraxis / minoraxis;

catch % if no object detected display no object
set(handles.Rear,'String','no object')
end
try % if object is square
if axis_ratio >= 1 && axis_ratio < 2
    % get bounding box and draw rectangle around object
Bound_rear = [props_middle(Index).BoundingBox];
rect_rear = rectangle('position', Bound_rear,'EdgeColor','r','LineWidth',10);
% mark centroid of object
rear_marker = line(center_x, center_y, 'Marker', '*', 'MarkerEdgeColor', 'r');
% display centroid on GUI
set(handles.Rear,'String',Rear_cent)
% if the chute is moved too far left, move the chute to the right
   if Rear_cent(1) < left_limit
       start(dcm4) % turns motor  on
  dcm4.Speed = -1 % sets voltage to -12 volts
  pause(0.2) % leave on for 0.2 seconds
  stop(dcm4) % shut off motor

% if the chute is moved too far left, move the chute to the right
elseif Rear_cent(1) > right_limit
  start(dcm4) % turn motor 4 on
  dcm4.Speed = 1 % set voltage to 12 volts
  pause(0.2) % leave on for 0.2 seconds
  stop(dcm4)  % shut motor off
    end
  % chute cap to high, move downward
if Rear_cent(2) > upper_limit

start(dcm3) % turn on motor 3
  dcm3.Speed = -1 % set voltage to -12 volts
  pause(0.1) % leave motor on for .1 seconds
  stop(dcm3) % shut motor off

% if cap to low, move upward
elseif Rear_cent(2) < lower_limit
start(dcm3) % turn motor on
  dcm3.Speed = 1 % set voltage to 12 volts
  pause(0.1) % leave on for .1 seconds
  stop(dcm3) % shut motor off
end

end % end squareness conditions
```

```
catch ME
     % display encountered error
    error_text = ME.identifier;

set(handles.error,'String',error_text)
end
% check if another button is pressed
 if   strcmp(get(handles.Track,'UserData'),'') == 1
     break % stop tracking if another button has been pressed
 end
 % combine masks to be displayed for debugging
  BW = BW_Front + BW_Rear + BW_Middle;
  %show image
 imshow(RGB)
 % turn axis hold off
 hold off

end
```

--- Executes on button press in capup.

This button when pressed will turn on motor 3 on the arduino, leave it on for the specified time and then turn it off. This motor is wired to the solenoid for cap up and down movement and supplies 12 volts to the solenoid moving the cap upwards

```
function capup_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to capup (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global dcm3 % load required variables
start(dcm3) % start motor 3
  dcm3.Speed = 1 % set motor speed to 100 percent in the positive direction
  % + 12 volts
  pause(0.001) % leave motor on for specified time
  stop(dcm3) % turn off motor
```

--- Executes on button press in capdown.

This button when pressed will turn on motor 3 on the arduino,

```
%leave it on for the specified time and then turn it off. This motor is
% wired to the solenoid for cap up and down movement and supplies -12 volts
%to the solenoid moving the cap downwards
```

```
function capdown_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to capdown (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global dcm3 % load required variables
start(dcm3) % start motor 3
 dcm3.Speed = -1 % set motor power to 100 percent in the negative direction
 % -12 volts
 pause(0.001) % pause for specified time
  stop(dcm3) % stop motor
```

--- Executes on button press in turnleft.

This button when pressed will turn on motor 4 on the arduino,

```
%leave it on for the specified time and then turn it off. This motor is
% wired to the solenoid for left and right turns and supplies -12 volts to
%the solenoid  turning the chute left
```

```
function pushbutton10_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to turnleft (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global dcm4 % load required variables
start(dcm4) % start motor 4
dcm4.Speed = -1 % set motor power to 100 percent in the negative direction
% -12 volts
 pause(0.1) % leave motor on for specified time
 stop(dcm4) % shut motor off
```

--- Executes on button press in turnright.

This button when pressed will turn on motor 4 on the arduino,

```
%leave it on for the specified time and then turn it off. This motor is
% wired to the solenoid for left and right turns and supplies +12 volts to
%the solenoid turning the chute right
```

```
function pushbutton11_Callback(hObject, eventdata, handles)
% hObject    handle to turnright (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global dcm4 % load required variables
start(dcm4) % start motor 4
 dcm4.Speed = 1 % motor power is 100 percent in the positive direction (12 volts)
 pause(0.1) % pause to allow motor to stay on for specified time
 stop(dcm4) % stop motor 4
```

DionCalibrate - for use with dion_2.m

This function allows the user to select a freehand region of the screen to select the target area. Once this area has been selected, the image is cropped to this area and a histogram of the hue, saturation and value channels are created. The minimum frequency is passed to the function as an argument. The highest and lowest pixel values for each channel that have a frequency greater than the specified argument are selected and output as the limits for the threshold, for hue, saturation and value channels.

```
function [channel1Min, channel1Max, channel2Min,channel2Max,channel3Min,channel3Max] = DionCalibrate(RGB,count)
% display image
imshow(RGB)
% convert to HSV
I = rgb2hsv(RGB);
% make copy of HSV image
I1 = I;
% get graphics handle of freehand region
h_region =imfreehand;
%create mask from region
region_mask = createMask(h_region);
% set  pixels outside mask to zero on 3 matrices
I1(repmat(~region_mask,[1,1,3])) = 0;
% wait 1 second
pause(1)
% show HSV image
imshow(I)
% wait 1 second
pause(1)
% break matrix into 3 one dimensional matrices
hue = I1(:,:,1);
saturation = I1(:,:,2);
value = I1(:,:,3);

% Define thresholds for channel 1 based on histogram settings
% hue histogram
[counth, binh] = imhist(hue);
% saturation histogram
[counts, bins] = imhist(saturation);
% value histogram
[countv, binv] = imhist(value);
% locate pixels above specified frequency and do not use pixels equal to
% zero (they are outside mask).
refh = find(counth > count & binh ~= 0);
% use reference to get values above required frequency
hue_thresh = binh(refh);
% find lowest value and make .1 smaller
channel1Min = min(hue_thresh) - .1;
% find highest value and make .1 bigger
channel1Max = max(hue_thresh)+ .1;

% Define thresholds for channel 2 based on histogram settings
% locate pixels above specified frequency and do not use pixels equal to
% zero (they are outside mask).
refs = find(counts > count & bins ~= 0);
% use reference to get values above required frequency
sat_thresh = bins(refs);
% find lowest value and make .1 smaller
channel2Min = min(sat_thresh) - .1;
% find highest value and make .1 bigger
channel2Max = max(sat_thresh) + .1;

% Define thresholds for channel 3 based on histogram settings
```

```
% locate pixels above specified frequency and do not use pixels equal to
% zero (they are outside mask).
refv = find(countv > count & binv ~= 0);
% use reference to get values above required frequency
value_thresh = binv(refv);
% find lowest value and make .1 smaller
channel3Min = min(value_thresh) - .1;
% find highest value and make .1 bigger
channel3Max = max(value_thresh) +.1;
% delete the freehand drawn region
delete(h_region)



end
```

Appendix C: FMEA Ranking

FMEA Ranking Sheet

SEVERITY

Description	Design	Ranking
VERY HIGH Severity Ranking when Variation or Potential Failure Mode affects Safe Machine Operation and/or Involves Non-Compliance with Government Regulations - **WITHOUT WARNING**	***Hazardous w/o Warning***	**10**
VERY HIGH Severity Ranking when Variation or Potential Failure Mode affects Safe Machine Operation and/or Involves Non-Compliance with Government Regulations - **WITH WARNING**	***Hazardous with Warning***	**9**
Vehicle / Item **INOPERABLE, LOSS OF PRIMARY FUNCTION**	***Very High***	**8**
Vehicle / Item OPERABLE, But at a **REDUCED LEVEL OF PERFORMANCE.** *Customer Very Dissatisfied.*	***High***	**7**
Vehicle / Item OPERABLE, But **COMFORT / CONVENIENCE ITEM(S) INOPERABLE.** *Customer IS Dissatisfied.*	***Moderate***	**6**
Vehicle / Item OPERABLE, But **COMFORT / CONVENIENCE ITEM(S) OPERABLE at a REDUCED LEVEL** of Performance. *Customer Somewhat Dissatisfied.*	***Low***	**5**
Fit & Finish / Squeak & Rattle Item Varies or Does not conform. *Defect* ***NOTICED BY MOST CUSTOMERS*** *(>75%)*	***Very Low***	**4**
Fit & Finish / Squeak & Rattle Item Varies or Does not conform. *Defect* ***NOTICED BY 50% CUSTOMERS***	***Minor***	**3**
Fit & Finish / Squeak & Rattle Item Varies or Does not conform. *Defect* ***NOTICED BY DISCRIMINATING CUSTOMERS (<25%)***	***Very Minor***	**2**
No Discernable Effect	***None***	**1**

Notes: Severity is a RELATIVE RANKING, within the scope of the individual FMEA

OCCURRENCE

	CNH Proposal for Low Volume Applications		SAE J1739 (Rev AUG2002)	
Probability of Failure	***Possible Failure Rates***	***Frequency***	***Frequency***	***Ranking***
Very High: Persistent Failures	≥ 20 per 100 vehicles, OR 200 per thousand vehicles/items	1/5	≥ 1/10	10
Very High: Frequent Failures	10 per 100 vehicles, OR 100 per thousand vehicles/items	1/10	1/20	9
	5 per 100 vehicles, OR 50 per thousand vehicles/items	1/20	1/50	8
Moderate: Occasional Failures	3.3 per 100 vehicles, OR 33 per thousand vehicles/items	1/30	1/100	7
	2 per 100 vehicles, OR 20 per thousand vehicles/items	1/50	1/200	6
Low: Relatively Few Failures	1 per 100 vehicles, OR 10 per thousand vehicles/items	1/100	1/500	5
	0.5 per 100 vehicles, OR 5 per thousand vehicles/items	1/200	1/1000	4
	0.3 per 100 vehicles, OR 3 per thousand vehicles/items	1/300	1/2000	3
Remote: Failure is Unlikely	0.2 per 100 vehicles, OR 2 per thousand vehicles/items	1/500	1/10,000	2
	0.1 per 100 vehicles, OR 1 per thousand vehicles/items	1/1000	< 1/100,000	1

Notes: Occurrence = Likelihood that specific cause mechanism or Variability will occur during design life. Ranking Number has a relative meaning rather than an absolute value.
Must be applied consistently during the FMEA

DETECTION

	Criteria: Likelihood of detection by Design Control	***Ranking***
Almost Impossible	Design Control will not and/or cannot detect a potential cause/mechanism and subsequent failure mode; or there is no design control.	10
Very Remote	Very remote chance the Design Control will detect a potential cause/mechanism and subsequent failure mode.	9
Remote	Remote chance the Design Control will detect a potential cause/mechanism and subsequent failure mode.	8
Very Low	Very Low chance the Design Control will detect a potential cause/mechanism and subsequent failure mode.	7
Low	Low chance the Design Control will detect a potential cause/mechanism and subsequent failure mode.	6
Moderate	Moderate chance the Design Control will detect a potential cause/mechanism and subsequent failure mode.	5
Moderately High	Moderately High chance the Design Control will detect a potential cause/mechanism and subsequent failure mode.	4
High	High chance the Design Control will detect a potential cause/mechanism and subsequent failure mode.	3
Very High	Very High chance the Design Control will detect a potential cause/mechanism and subsequent failure mode.	2
Almost Certain	Design Control will almost certainly detect a potential cause/mechanism and subsequent failure mode	1