final report

Project code: A.TEC.0062

Prepared by: Richard Aplin

Strategic Engineering System Pty Ltd

Date published: April 2009

PUBLISHED BY
Meat & Livestock Australia Limited
Locked Bag 991
NORTH SYDNEY NSW 2059

Sheep hock reposition vision system

Meat & Livestock Australia acknowledges the matching funds provided by the Australian Government and contributions from the Australian Meat Processor Corporation to support the research and development detailed in this publication.

Contents

Scope

This report details the conclusion of milestone two and three outcomes, including methodology used to perform the vision systems required tasks, the results obtained thus far and a path forward for the further development of research completed during the project. The sole milestone 4 outcome is the completion of this report.

2 Software

2.1 Operating system

For ease of development Microsoft Windows Vista was chosen as the operating system. However, as the project may need to be ported to an embedded operating system in the future, cross-platform software libraries have been utilised. This allows the easy porting of the software from the current operating system to another with minimal changes, mostly related to the compiler utilised on the varying operating system. The Integrated Development Environment (IDE) used in developing the Sheep Hock Repositioning Vision System was Microsoft Visual Studio. Some of the possible future operating system choices for embedding the vision system include Windows CE, and numerous Linux variants with a real-time kernel.

2.2 Software architecture

The final software architecture for the Sheep Hock Reposition Vision System is detailed in Figure 1. A further explanation of each of these modules and their purpose is provided following the diagram. The architecture has changed from the initial design to better perform tracking while reducing processing requirements.

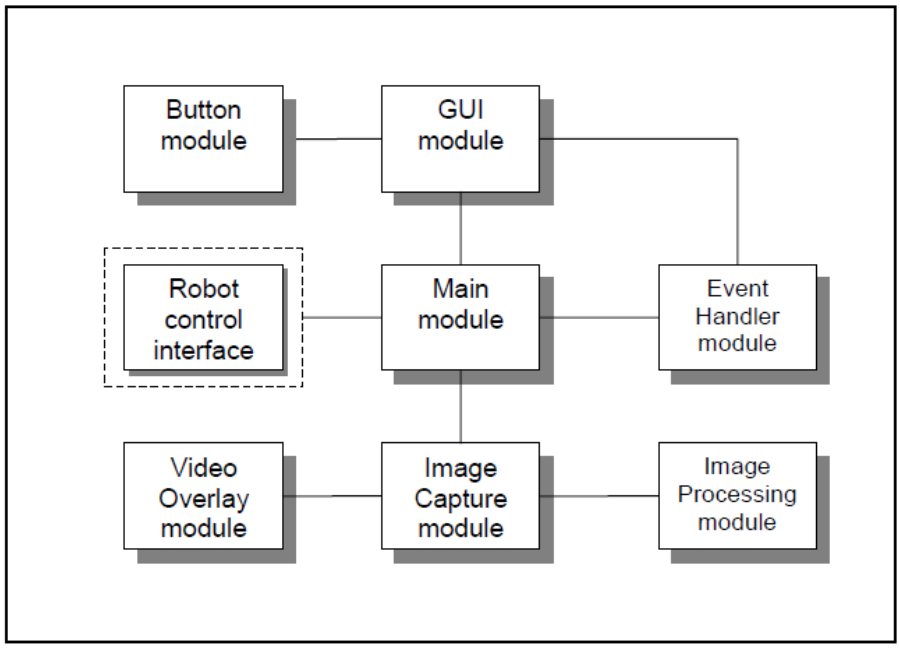

Figure 1: Software architecture. NB: dotted section refers to software components to be implemented in possible future projects

2.2.1 Main module

Contains the main loop and controls the process of initialising modules at program start-up and de-initialising all modules before program exiting.

2.2.2 GUI module

The GUI contains all user interfaces and is used to refresh the GUI plane on the right of the video overlay, including drawing of all buttons. The role of this module has changed from initial design, as the video overlay control is now removed from this module and handled separately.

2.2.3 Button module

All functions related to button control are implemented in this module. Buttons are used to update settings, add new detection areas and other essential related functions.

2.2.4 Event Handler module

This module handles all mouse and keyboard events including mouse button presses and works in conjunction with the button handling module to perform tasks such as creating new detection areas or setting up gambrel information (gambrel thickness, etc).

2.2.5 Video Overlay module

Drawing of video data and all processing information is handled by this module. Processing information overlaid onto video data includes detection area boxes, gambrel axis and detection points and leg data such as top position of leg, knee, and leg axis.

2.2.6 Image Capture module

The Image Capture module interfaces with the camera driver and starts the image capture stream. A call-back function is defined within the module and called each time the camera captures an image.

2.2.7 Image Processing module

Processing of image data includes lens distortion correction, background subtraction and edge detection and is performed within this module. Edge detection data is stored as contour vectors, processing and filtering of these contours into gambrel and leg data is the core task of the module. Default detection areas are spawned within the initialisation of the module from a settings file.

2.3 Detailed software overview

2.3.1 Main function (`main`)

- Initialise camera module (`initCamera`)
- Initialise processing module (`initProcessing`)
- Initialise GUI module (`initGUI`)
- Initialise event handler module (`initEventsHandler`)
- Start video capture, starts a new thread where the image capture call back function (`see FrameReadyCallback`)

- is called from (`startVideoCapture`)
- Start of loop
- Wait for an event (`handleEventsWait`)
- Refresh GUI (`refreshGUI`)
- Go to start of loop if a quit event has not occurred
- Stop video capture (`stopVideoCapture`)
- De-initialise camera (`deInitCamera`)
- De-initialise processing module (`deInitProcessing`)
- De-initialise GUI (`deInitGUI`)

2.3.2 Image capture call back function (`FrameReadyCallback`)

- Get frame (`UCC_GetBitmapImage`)
- Update/subtract background (`updateAndSubtractBackground`)
- Remap image to remove lens distortion (`cvRemap`)
- Process all corner detection areas - not used for this project (`overlayAllCorners`)
- Process all edge detection areas (`overlayAllEdges`)
- Track all objects (`see trackAllObjects`)
- Convert video format to an SDL texture for displaying on the screen (`overlayAllEdges`)
- Refresh video overlay (`refreshVideoOverlay`)
- Update detection area positions (`updateAllDetectionAreaPoisitions`)

2.3.3 Tracking all objects (`trackAllObjects`)

- Find connecting lines between detected blobs in gambrel top and bottom detection areas (`findAllConnectingLines`)
- If at least two gambrels were found then convert points for left gambrel into a gambrel axis line (`convertTwoPointsToLine`)
- Find end point (bottom) of gambrel (`findEdgePointAlongLineInEdgeArea`)
- Modify end point of gambrel to a more accurate known location (`modifyEdgePointToGambrelCenter`)
- Calculate masking points of gambrel (`findMaskingPointsOfGambrel`)
- Perform above processing tasks for right gambrel
- Find connecting lines between detected blobs in leg top and bottom detection areas (`findAllConnectingLines`)

- Filter out gambrel data from contour/edge data (see filterAllContours)
- Convert points for left leg into a leg axis line (convertTwoPointsToLine)
- Find end/top point of leg (findEndOfLegInEdgeArea)
- Find all points of interest for leg (findLegPointsOfInterest)
- Perform above processing tasks for right leg

2.3.4 Filter out gambrel data from contour/edge data for both legs (filterAllContours)

- Set up gambrel masking (setupGambrelMaskingData)
- Filter left leg (see filterLegContourInArea)
- Filter right leg (see filterLegContourInArea)
- Clear and clean up gambrel masking data (clearGambrelMaskingData)

2.3.5 Filter contour/edge data for a leg (filterLegContourInArea)

- Remove gambrel data from contour/edge data. This function causes the contour data to be cut into sections that have been masked. These sections are placed in an array with information on how their start and end points have been masked (see maskContoursIntoContourSections)
- Find contour sections that should be joined and join them. This can be such things as the leg hitting the gambrel (findJoiningContourSections)
- Join contour sections (joinContourSections)
- Remove empty contour sections from array. These contour sections have been joined to other sections in the array and their entries in the array have been memset to '0' (removeEmptyContourSections)
- Place contours from array into contour tree that OpenCV can understand placeContourArrayIntoContourTree)

2.3.6 Filtering contour data into sections (maskContoursIntoContourSections)

A member of the tProcessingData structure is the IplImage pointer to pTestingImage. This image is an 8-bit image used when testing for a contour intersecting with a known line. For example when testing for the bottom of the gambrel a white line (colour 0xFF) of a specified thickness is projected along the axis of the gambrel. Each point of the contour data (which contains x and y values) is then checked against the corresponding point on the testing image, if the image contains a white pixel at a contour point then we know that at this specific point the contour crosses the projected axis and it can be determined that this is the bottom of the gambrel. This methodology is also used when filtering and masking gambrel contour data to isolate only leg contour data only on a much lower lever.

The diagram below shows an approximation of how a gambrel is masked by drawing lines on the pTestingImage image, this is performed by the setupGambrelMaskingData function. The image on the left in **Figure 2** shows a representation of what this masking data looks like when drawn. Please note that pTestingImage is an 8-bit single channel image and hence is

greyscale, coloured illustrations have been used below to easily identify the difference in masking lines.

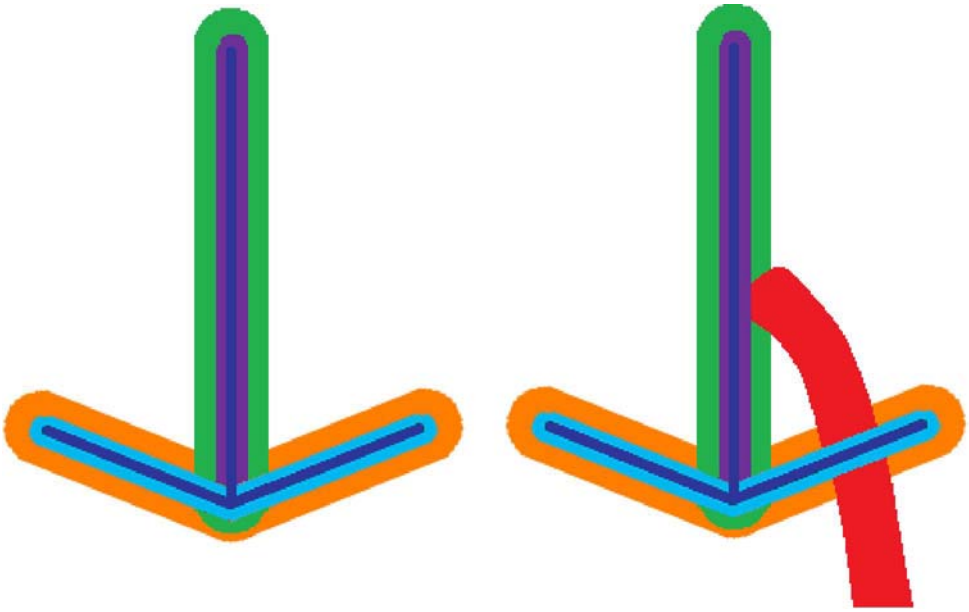

Figure 2: Masking of a gambrel

The masking lines are explained below, the red line on the right image in **Figure 2** is a representation of a leg and how it is masked:

- **Purple** – masks the gambrel upright and should be wide enough to completely mask the upright part of the gambrel. Contours that touch this part of the masking area are cut, and at the next allowed un-masked contour point a new contour section is started and continues until the contour finishes or hits another purple or light blue masking line.
- **Light blue** – masks the gambrel prongs and should be wide enough to mask them, care should be taken in choosing this value as the gambrel prong can often be bent in the real world environment. Contours that touch this part of the masking area are cut, and at the next allowed un-masked contour point a new contour section is started and continues until the contour finishes or hits another purple or light blue masking line.
- **Dark blue** – these lines go along the axis of the gambrel and along its prongs. This is used to allow us when joining contours to know if the join being calculated crosses through the gambrel
- **Green –** contours masked by this colour are not removed from the contour being filtered. These areas are used when finding which contours should be joined as a line sample between two end points which should be joined will contain both green and purple pixels
- **Light blue** – contours masked by this colour are not removed from the contour being filtered. These areas are used when finding which contours should be joined as a line sample between two end points which should be joined will contain.

2.3.7 Finding end bottom of gambrel and modifying to constant position

`(findEdgePointAlongLineInEdgeArea and modifyEdgePointToGambrelCenter)`

These two functions are responsible for finding the end of the gambrel and using this end point to then find a point on the gambrel that is always relatively in the same location. This second point, coloured blue in **Figure 3**, is used when referencing the gambrel.

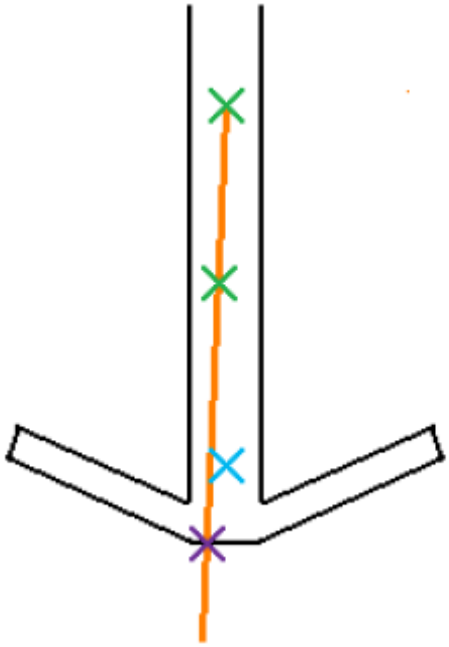

Figure 3: Finding the end point and adjusted masking reference of the gambrel

The two green crosses in **Figure 3** represent the midpoint of the two contours detected by the upper and lower gambrel detection areas. These midpoints are calculated through the centre moment of the contours. The function **`findEdgePointAlongLineInEdgeArea`** uses the axis created by these two midpoints to project a white line of a specified width on the testing image (`pTestingImage`). The contour data is then processed by taking each point of the contour and checking the corresponding pixel value in the testing image, if the pixel is white then that contour point crosses the axis of the gambrel. All points at which the contour passes through the line of axis are then averaged to find the end point of the gambrel as represented by the purple cross in **Figure 3**.

To adjust this endpoint to a point whose position is relatively the same and can be used as a masking reference the **`modifyEdgePointToGambrelCenter`** is called. The first step in determining the masking reference point is taking a point a specified distance up the axis of the gambrel from the end point. At this point the distance between the left and right sides of the gambrel shaft is determined (in a similar manner to finding the end point using the testing image and checking contour data against lines projected perpendicularly from the point). The midpoint between the two sides can calculated by finding the average of the left and right side points of the gambrel shaft.

The process used to find the bottom point of the gambrel in the function `findEdgePointAlongLineInEdgeArea` is also utilised within the function for finding the end point of a leg, `findEndOfLegInEdgeArea`. However, modifications have been made to increase the accuracy the process better suited for determining the end point of a leg.

3 Hardware

The hardware layout has been modified from the initial proposed layout. Through software design it has been possible to remove the need for a triggering device to start tracking of the gambrels. **Figure 4** details the revised layout.

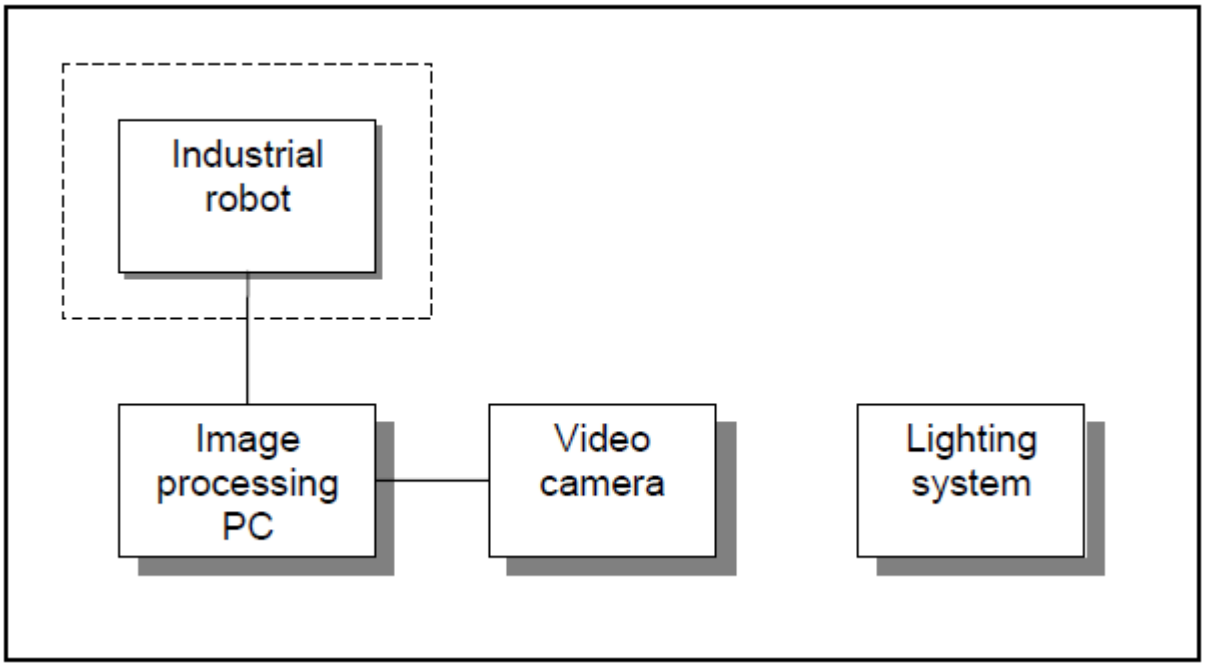

Figure 4: Hardware block diagram. NB: dotted section refers to hardware to be implemented in possible future projects

3.1 Camera and lens selection

An Allied Vision Technologies Marlin F-080B was selected for the vision system. The specifications for this camera are:

- 1024 x 768 pixels
- Black and white
- 20 fps (full frame mode)
- Firewire camera

Initially a colour camera was proposed, however, the use of back lighting essentially creates a silhouette hindering the use of colour for processing useless. Using a colour camera under these circumstances would then require a colour to black and white conversion step, wasting processing time.

The lens utilised was a 4.0mm – 12.0mm varifocal industrial lens. A varifocal lens was used to accommodate the different camera mounting positions allowing the system to be flexible enough to trial in most abattoirs.

3.2 Computer system

The computer used was a standard laptop with the following specifications

- Intel Core 2 Duo 2.2GHz
- 2GB RAM
- 220GB HDD
- Onboard 4-pin Firewire port (requires 4 to 6-pin adapter for camera)
- Windows Vista operating system

Processing time using this system was approximately 16ms per frame. A frame rate of 25fps allows for 40ms of processing time before the processing time causes lost frames and a decrease in overall frame rate. Processing times show the computer system used easily handles the processing requirements of the vision system.

3.3 Lighting box

During a preliminary site visit appropriate measurements for the light box dimensions were taken. The initially proposed lighting box dimensions were 1200 x 900, however it was found that the height of the box could be reduced. The finalised lighting box dimensions were 1250 x 700 x 2400. The lighting box contains five double battens evenly spaced with the iron core ballasts swapped for high frequency Omnitronix ballasts to stop the flicker effects caused by a 50Hz AC power supply.

3.4 Trigger device

A trigger device is no longer required as system processes data in real time and is able to determine when the target is present in the viewing area.

4 Tracking methodology

The overall tracking methodology for the Sheep Hock Repositioning System is based on edge detection and contour processing. Edge detection and contour processing combined with multiple detection areas allows the accurate processing of leg and gambrel positions. **Figure 5** shows the system successfully tracking a set of gambrels, sheep's legs and associated points of interest.

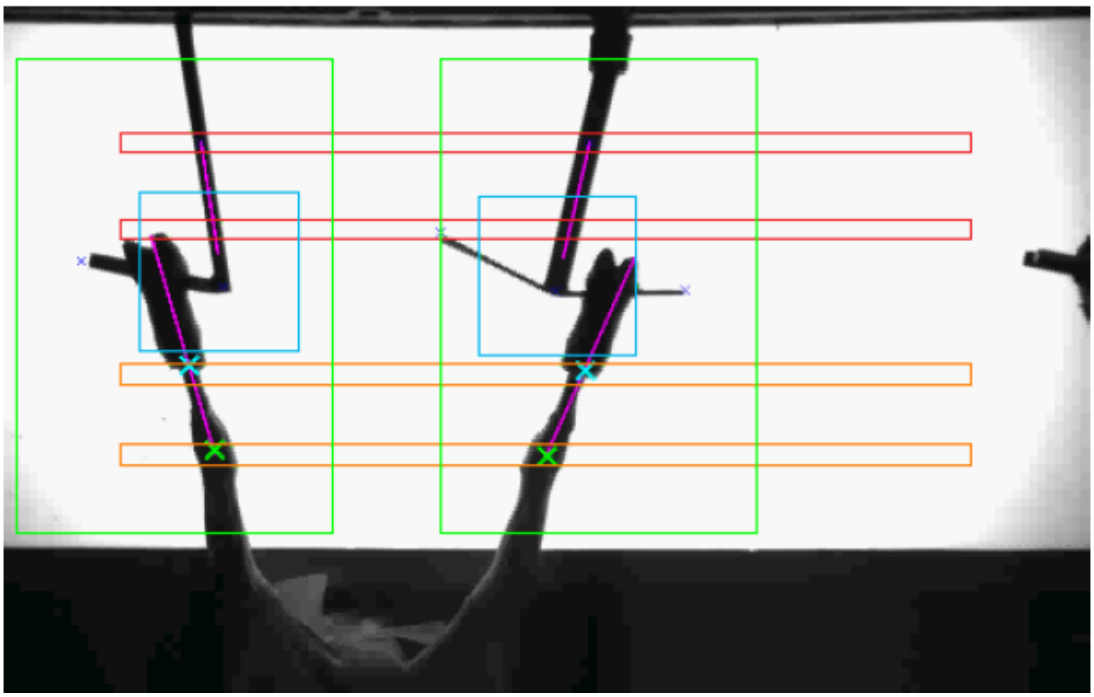

Figure 5: Screen capture of system successfully tracking

To find gambrel position -

1) The upper and lower gambrel detection areas are highlighted in red. The contour data obtained from the black spots produce by the gambrel shafts in these detection areas is used to find the axis of each gambrel. These axes are shown as purple lines.
2) These axes are then used within the blue detection areas. The bottom of the gambrels is found by projecting this axis through the detection area and finding where this intersects with the contour/edge detection data.
3) Once the bottom point of the gambrel is located the contour data from within the blue detection area is used to more accurately find a reference from which the gambrel masking data can be referenced to. This point is found by moving a specified distance up the gambrel axis from the bottom point and then finding the midpoint of the gambrel shaft.

To find leg position and point of interest

1) In a similar method to calculate the gambrel axes the orange detection areas are used to determine the leg axes.
2) Gambrel position and masking information is then used to masking gambrel edge/contour data from the green detection areas.

3) The resulting contours are then processed in a similar method to finding the end points of the gambrels.
4) Instead of further refining the top point of the leg into a masking reference point as it is on the gambrels, this point is used as a reference for calculating the leg points of interest such as the hair point and the knee.
5) The masked leg contour data is processed into a leg width array with the starting point being the top point of the leg.
6) Leg width data is further processed into gradient data that is used to determine where the hair point (light blue cross in **Figure 5**) and knee (light green cross in **Figure 5**) are located for each leg

Using this methodology the Sheep Hock Repositioning Vision System provided results suitable to be used within a larger robotic sheep leg repositioning system to provide appropriate pick up positions for the robot.

5 Results

Promising results were obtained almost immediately after setting up the system on site. With only a few modifications to the software the system was running extremely robustly. It was noted that under some circumstances when the wool is not correctly removed from the leg then tracking of the knee and wool line at the foot may not yield accurate results. This issue may need to be addressed before integration of the complete system. As a whole the vision system software has showed it is able to track the sheep's legs accurately enough to guide an industrial robot to specified leg pick up and placement positions.

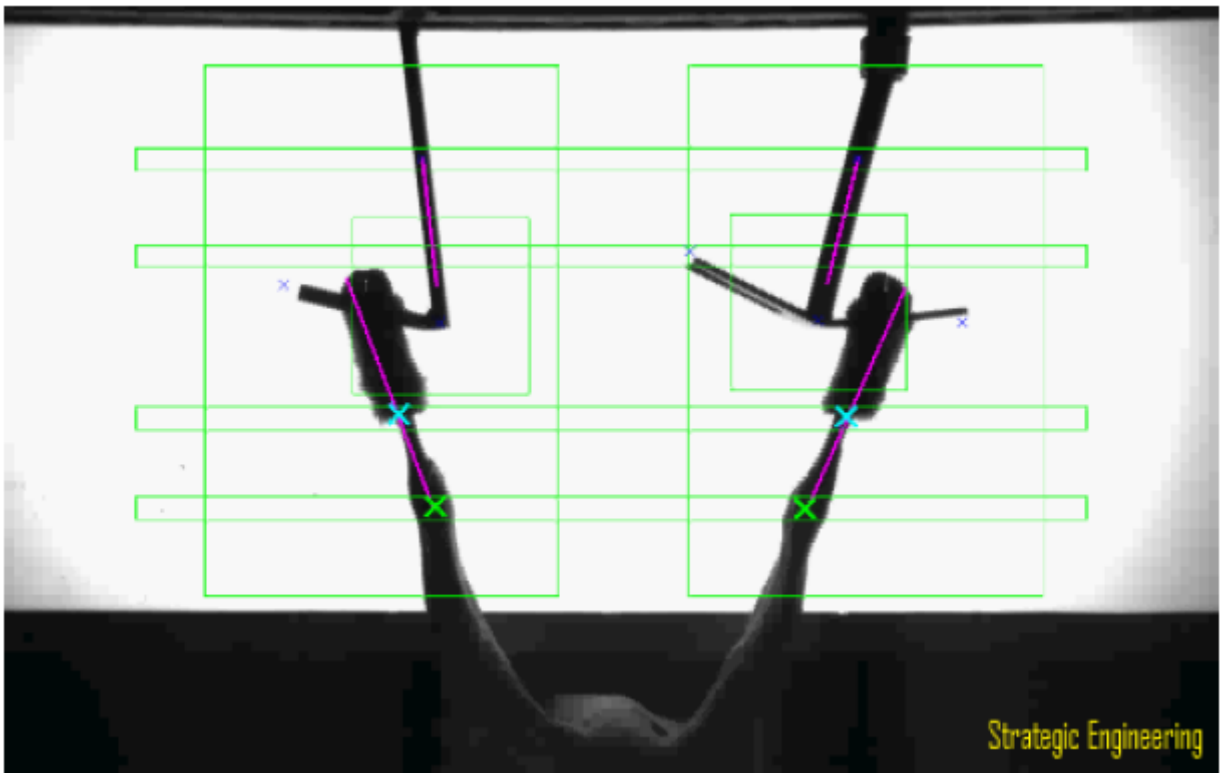

Figure 6: Screen capture of system successfully tracking

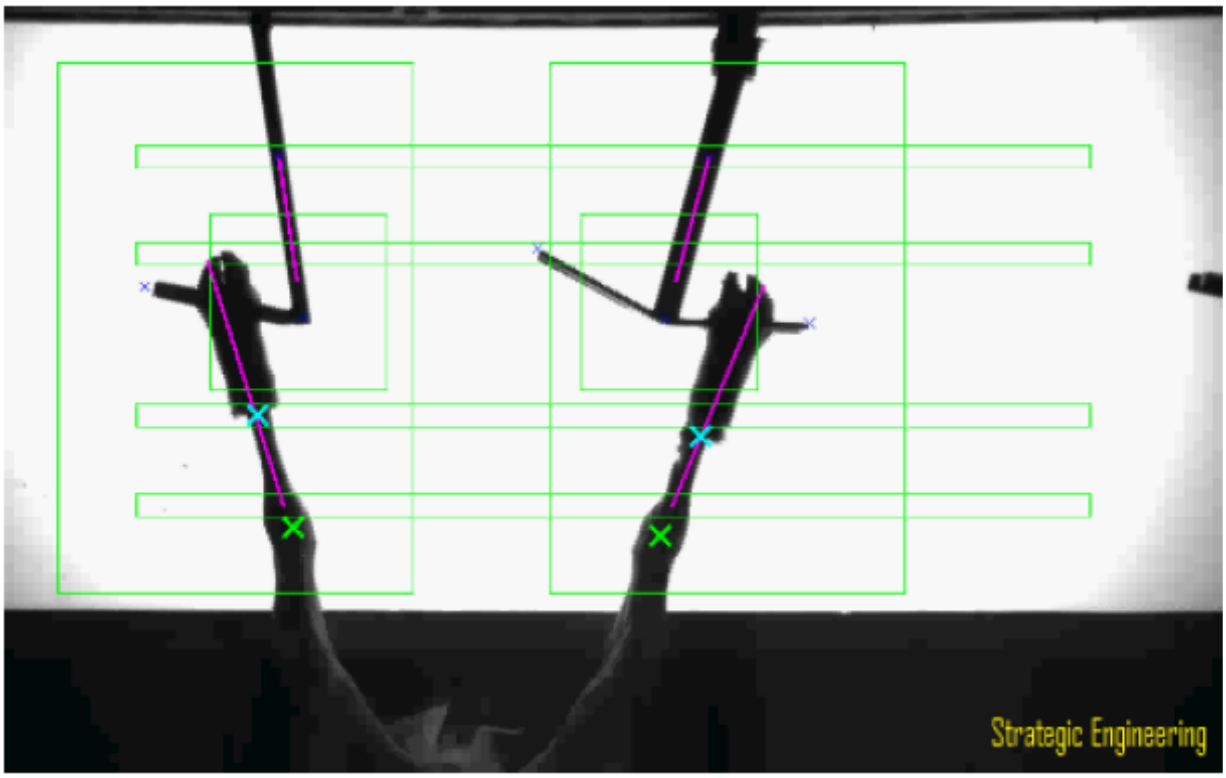

Figure 7: Screen capture of system successfully tracking

6 Development path

Taking this research further would involve the implementation of the vision software in a complete Robotic Hock Repositioning System. This robotic cell would require in-house trials before installing the system on site and running comprehensive trials to assess the system and to identify and solve any performance issues that may arise.

Upon successful completion and trial of the system, the path for new research may head in the direction of adapting the vision software for new tasks. One such path is the tracking of the sheep's front legs before they are attached to the front gambrels of the production line. A robust system able to effectively track the legs may provide a basis for the automation of the process in which the front legs are lifted and placed into the front gambrels.

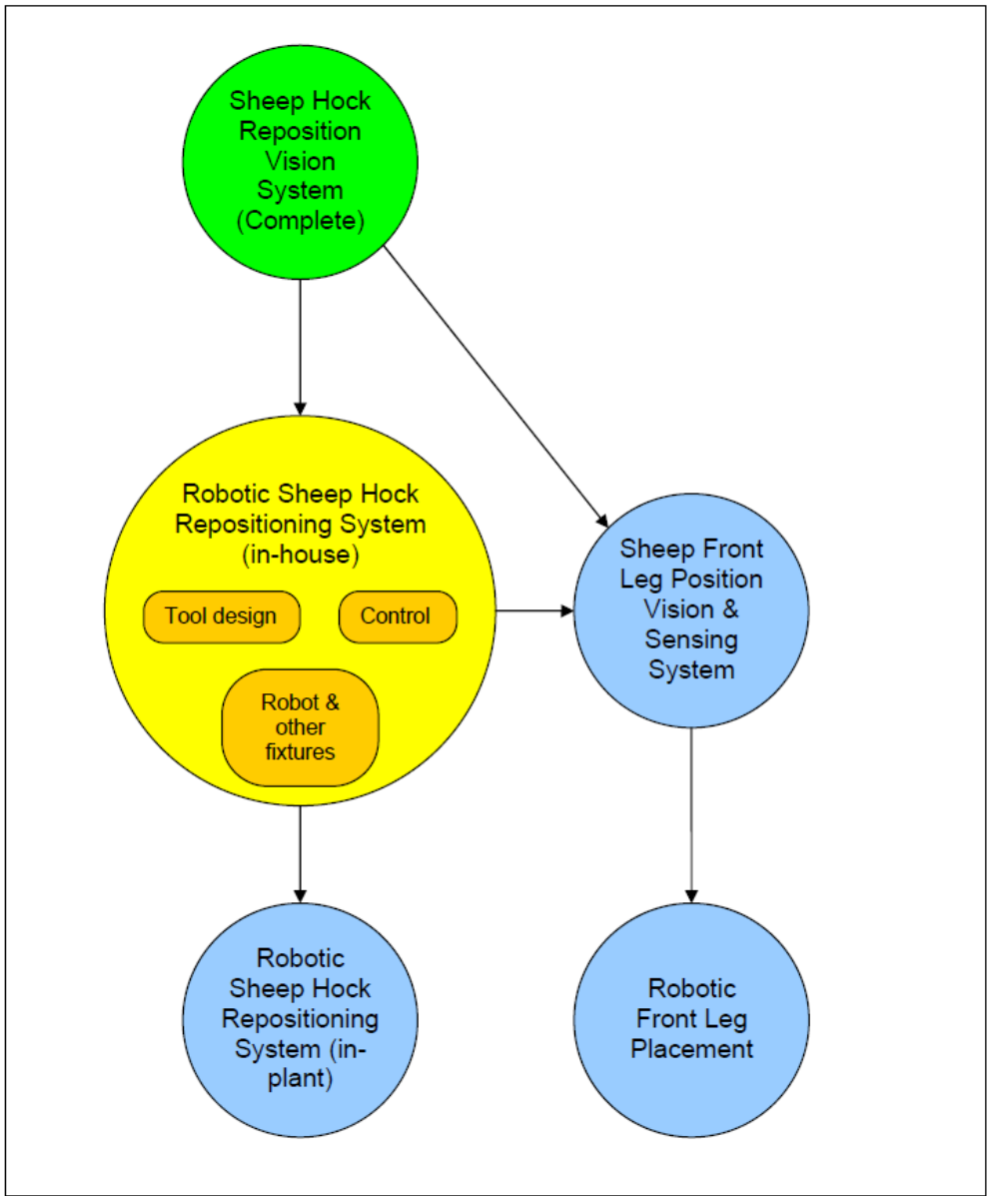

Figure 8: Possible project path for further research

7 Milestone 4 - Summary

During development of the Sheep Hock Repositioning Vision System milestones five and two were merged and completed as one, thus completion of milestone four sees the conclusion of all required research and development of the vision system. All software has been successfully tested in a real-world environment and it is believed that the software in its current state has the ability to robustly track sheep's legs in real-time. Real-time tracking enables the Sheep Hock Repositioning Vision System to provide a robot with real-time position co-ordinates. The methodology in which the system tracks in real-time has been detailed and possible future paths for research and development outlined.