

ECE 480 Senior Design Project Final Report

Smart Camera to Measure Coil Diameter

Design Team 5



Sponsor: Carlos Forjan

Facilitator: John Deller

Design Team Members:

Joe McAuliffe
James Quaglia
Ian Siekkinen
Petros Taskas
Matt Wesolowski
Poyuan Han

Executive Summary

ArcelorMittal requested a more accurate measurement system for the diameter of the steel coil that is being wound on the tension reel at the end of the pickling process. During the visit to ArcelorMittal, the team was told the current Programmable Logic Controller estimation for the diameter of the coil is not adequate. The reason is that the current method uses the number of rotations of the tension reel and the gauge (thickness) of the steel to measure the diameter of the coil. This is inaccurate because of the steel slipping over the reel. To better estimate the diameter of the coil, a “Smart” Camera that uses image segmentation to measure the pixel diameter is used. The hardware that was used to complete this project was an Axis Communications P1355-E Network Camera with a 15W PoE Midspan Injector. This is a high definition 1080p 30fps camera that is connected to a network which interfaces with existing software to provide live feed to the user. The software required for this project is MATLAB R2014b and OpenCV. After the hardware and software were set up, three different sizes of plates were used to test the accuracy of the smart camera. Once optimized and calibrated the smart camera effectively picked up the different pixel diameters of the different sized plates. The system measures the diameter of the plates to be within an error of 0.14%. This is approximately 6.8x better than the requirement given by ArcelorMittal.

Acknowledgements

Carlos Forjan: Sponsor, for providing video feed of the steel coil being wound, the Axis Communications P1355-E Network Camera and injector. This made it possible to jump right into the project and eliminate large cost and wait time associated with purchasing a network camera.

Dr. Jack Deller: Facilitator, for his mentorship during the project. His advice and feedback contributed greatly to our success on this project by helping us become a better team.

Dr. Timothy Grotjohn and Dr. Latia Udpa: ECE 480 professors, for their guidance and help they provided during the design process.

Brian Wright: ECE shop, for providing us with a significantly better Dell laptop that enabled us to handle the extreme CPU power needed for our image processing.

Dr. Robert J. McGough: MSU Professor, for providing us with permissions to access the Instrument Control Toolbox for MATLAB and Simulink

DECS staff members, most specifically Ryan Roberts and Dan Zoumbaris deserve a special thanks for assisting in the set up process for MATLAB onto the laptop, and helped the team solve other technical issues. This project could not have been completed without the exceptionally impressive expertise of Ryan Roberts.

Table of Contents

Executive Summary	2
Acknowledgment	3
Chapter 1 - Introduction & Background	5
1.1 Introduction	
1.2 Background	
Chapter 2 – Finding a Solution	5
2.1 Fast Diagram	
2.2 House of Quality	
2.3 Design Feasibility Matrix	
2.4 Chosen Solution	
2.5 Estimated Cost	
2.6 Gantt Chart	
Chapter 3 - Final Design	11
3.1 Hardware Setup	
3.1.1 Setup Axis Camera to Local Network	
3.1.2 Computer for Image Processing Specifications	
3.2 Software Setup	
3.2.1 Image Processing Toolbox MATLAB	
3.2.2 OpenCV	
3.2.3 MATLAB MEX-files	
3.3 Software Design	
3.3.1 Loading an Image from a Live Video Stream	
3.3.2 Segmenting an Image	
3.3.3 Measuring Objects within an Image	
3.3.4 Communications via TCP/IP	
3.3.5 Other Features (Bounding Box, Visual Ruler)	
3.4 Design Issues and Solutions	
3.4.1 No Blob Robustness	
Chapter 4 - Calibration & Testing	18
4.1 Calibration	
4.1.1 Calibration Data & Setup	
4.2 Testing & Proof of Functionality	
Chapter 5 - Summary and Conclusion	20
5.1 Summary and Conclusions	
5.2 Success and Failures	
5.3 Suggestions for the Future	
5.4 Final Costs	
5.5 Final Budget	
5.6 Final Schedule	
Appendix 1 - Individual Contributions	28
Appendix 2 – References	29
Appendix 3 - Technical Documents	30

Chapter 1 - Introduction & Background

1.1 Introduction

ArcelorMittal needs a more reliable and robust way to measure large coils of cold-rolled steel coming off their Pickle Line. According to ArcelorMittal, “The main objective of #5 Pickle Line is to remove scale from hot rolled carbon steel coils. Scale consists of three oxide layers: Fe_2O_3 , Fe_3O_4 , and FeO . Scale is removed by the chemical action of hydrochloric acid. This process is known as pickling. The operation of the line is continuous with the entering hot rolled coils welded head to tail, temper-rolled, pickled, inspected, side-trimmed, oiled and sheared into coils of the desired size at the end of the line.” Steel rolls of different sizes are required based on the needs of customers of ArcelorMittal. Currently, coils are cut to size based on calculations by their Programmable Logic Controller. As a result of the shortcomings in the current system, our team has been tasked with developing a “Smart” Camera to continuously feed a more accurate coil diameter to the Level Two Computer system that controls the steel pickle line. The goal is to improve the productivity of the line and thus improve product yield. A computer vision (CV) solution will be successful from its inherent robustness and survivability in the harsh industrial environment of the plant. In addition, this solution can quickly be adapted, relocated or installed on other tension reels and pickle lines on site.

1.2 Background

During the ArcelorMittal site visit to Indiana Harbor in South East Chicago, Carlos Forjan, Process Automation Manager, Project Sponsor and customer of the project provided our team with an Axis Communications P1355-E Network Camera. Several requirements were met by using the supplied camera including: an enclosure suitable to survive the harsh industrial environment, a proven and reliable product that is familiar to the customer, and a network camera that utilizes Power over Ethernet (PoE) for easy installation.

Chapter 2 - Finding a Solution

2.1 Fast Diagram

The main goal of this project, as shown in the fast diagram, was to be able to input an accurate measurement of the diameter of a coil of steel into the pickle line control system. In order for this to be accomplished two basic tasks needed to happen. The diameter needs to be accurately measured and this value has to be communicated to the control systems in a usable format. The

communication can be accomplished by a periodic TCP/IP ping. This ping contains the current diameter of the coil as an integer. Coming up with this integer is a more complicated process as is obvious in the fast diagram below. First, the camera has to be mounted in the desired position. Then it must be powered using a Power over Ethernet switch that can be conveniently located away from the camera itself. The other Ethernet port on the switch then has to be connected to the computer. The IP address of the camera then has to be determined, either by discovering the default address assigned by the computer or by manually assigning one. This IP address can then be used to obtain video from the camera. Once the video is obtained it needs to be processed so that it becomes something useful. This is done by eliminating all artifacts other than the desired circular blob representing the coil of steel in the image. The diameter of the coil can then be determined in pixels and converted to an actual diameter using the known geometry of the setup.



Figure 1 – FAST Diagram

2.2: House of Quality

Using the house of quality approach, it becomes obvious that the critical customer requirements are as follows:

1. Accurate measurement
 - a. The integer output of the measurement system needs to be within a half inch of the actual diameter in the positive or negative direction.
2. A measurement system separate from the moving parts of the coiler.
 - a. The measurement system needs to not physically interact with the coiler device
3. Ability to communicate with the pickle line control system.
 - a. The measurement system needs to be able to communicate the diameter measurement back to the control system so that decisions can be made based on the coil's diameter.
4. A measurement system with no moving parts.
 - a. The measurement system cannot have any moving parts that will require maintenance.

2.3 Design Feasibility Matrix

Feasibility Criteria	Wt.	Axis "Smart" Camera	Ultrasonic Sensor	Laser Sensor
Operational Feasibility Functionality. A description of how well suited the solution is to solve the problem. Political. A description of how well received this solution would be.	30%	This system will work exceptionally well since it will be away from moving parts and easily accessible. It also has functions to communication with the network and existing software. This system will be user friendly and organized due to its simplicity graphic interface software Score: 30	This system will measure well but will be subject to potential damage and dust collection This system will not be very user friendly as it will require many calibrations Score: 20	This system will work poorly due to the possible reflection and its vicinity to moving parts. This system will have a user friendly interface but will require to many calibrations by the user. Score: 15
Technical Feasibility Technology. An assessment of the maturity, availability, and desirability of the computer technology needed to support this solution. Expertise. An assessment of the technical expertise needed	30%	"State of the art" camera. Easily acquirable. Step by Step guide to interface with networks.	Old technology with a moderately cheap sensor. Very easily acquirable. May have a few issues communicating with networks due to no pre-developed hardware or software.	Moderately new technology. Moderately available. Easy to acquire. May have a few issues communicating with networks due to no pre-developed hardware or software.

to develop, operate, and maintain the solution.		Moderately easy to develop. Very easy to operate. Very easy to maintain since it will be installed near a catwalk. Score: 27	Moderately easy to develop. Difficult to operate. Difficult to maintain due to installation area near moving parts. Score: 20	Moderately easy to develop. Difficult to operate. Difficult to maintain due to installation area near moving parts. Score: 20
Economic Feasibility Cost to develop: Payback period (discounted): Payback Period= Initial Investment/Periodic Cash Flow	30%	\$1,000+ Payback period = 7 days Score: 28	\$100+ Payback period = 1.5 days Score: 30	\$200+ Payback period = 2.8 days Score: 29
Schedule Feasibility An assessment of how long the solution will take to design and implement.	10%	3 to 5 weeks Score: 10	4 to 6 weeks Score: 9	5 to 7 weeks Score: 8
Ranking:	100%	95	79	72

Table 1 – Design Feasibility Matrix

2.4 Chosen Solution

The team decided that the best solution was to use the Axis Communications P1355-E Camera to capture a 1080p live video feed of the steel being coiled and a dedicated Windows 7 computer to process the video produced by the Axis camera.

This solution is unique in that it uses an IP camera instead of a USB, Firewire, or GigE Vision camera which are more commonly used in image processing applications such as this one. There are a few simple reasons why an IP camera is ideal for this application. First, the location where the camera will be mounted is approximately 30 feet away from the control room on a straight line. With the cable running along structural beams, this corresponds to approximately 40 feet of cable. An Ethernet cable between these two locations already exists eliminating the need to spend time and resources running a new cable. Second, the camera will be replacing an existing security camera. Being an IP camera it will be easy for the security system to also make use of the streaming video so the camera will be able to double as the security camera that it is

displacing. Third, the camera will not need a power outlet nearby or a lengthy power cable to be ran to a remote outlet. The camera can easily be powered using Power over Ethernet (PoE). The existing Ethernet cable, running from the camera location to the control room, can be plugged into a PoE switch in the control room powering the camera. The other outlet of the PoE switch can then be connected to the computer for communication between the camera and the computer.

The IP camera solution also has some disadvantages. Ethernet cameras are not typically used for image processing and as a result most common image processing software does not include a way to communicate with this type of camera in real-time. The team solved this issue by image grabbing. Rather than analyzing a live video input the software periodically grabs an image from the IP stream and performs the analysis on this still image. The software then performs this process iteratively in a while loop fast enough to produce an accurate value of the diameter.

The camera will be mounted approximately twenty feet away from the coiler and about ten feet off the ground on one of the building supports. This area is an ideal place for the Axis camera to be mounted because it will have a clear path to the coiler for the camera to take unobstructed video. An operator or other person walking in the area will not be able to block the image because of the camera's position ten feet above the ground. The camera will also most importantly be out of the way of the moving components of the coiler assembly. One of the main concerns was how robust a measurement system could be as any physical sensor touching the coiler would be around many moving parts and would have a high risk of being damaged. The Axis camera, mounted in a location removed from the coiler, will not be subjected to moving parts and will not have to be moved around when maintenance needs to be done to the coiler. The location of the Axis camera is also just below a catwalk, providing for easy installation and easy access should any maintenance be needed.

The last part of the solution is the program that was written to do the actual processing of the video produced by the Axis P1355. The frame grabbing approach mentioned above not only allows for the use of an Ethernet camera, but also decreases the processing power needed to analyze the image as there will be far fewer frames to analyze than a live video. The program makes use of blob analysis to determine the diameter of the coil. This approach allows for many different types of variation, within the image. The coiling steel is the only large continuous object in the view of the camera and is easily identified through several steps. First the image is converted to a binary, black and white only, image. After the image is converted to binary, the image is segmented to separate the white parts of the image into blobs, or continuous areas of white pixels. All blobs that are touching the border of the image are then ignored as are all blobs that are below a desired area threshold. Once this threshold is calibrated all blobs aside from the desired blob representing the steel coil are removed from the image. When the steel coil blob is the only one remaining, measuring the diameter becomes a matter of counting the pixels that make up its diameter and converting this value to an actual diameter using the geometry of the setup. This blob analysis allows for variations in the image without losing the diameter

measurement. The blob representing the steel coil will always be able to be distinguished in different lighting conditions and it is just a matter of adjusting the threshold accordingly. This method also allows for varied placements of the Axis camera as the location of the steel coil in the frame does not have to be fixed for the software to isolate the correct blob.

The combination of the 1080p Axis camera and dedicated Windows based computer will allow for accuracy far beyond the needs of the line. For the coils to be within the specifications demanded by ArcelorMittal's customers, the diameter needs to be accurate within a half inch in either direction. In laboratory testing the smart camera system has been shown to have a maximum error of 0.06 inches in either direction. This error was recorded using a small scale model and is expected to increase when the system is implemented on the full scale line but is still expected to be well below 0.5 inches.

2.5 Estimated Cost

There was no cost to the MSU ECE department due to the building of this prototype. The Axis camera and switch were provided by ArcelorMittal, the laptop used as the dedicated pc during the development of the software was borrowed from the ECE Shop, the MATLAB software used by the team to develop the image processing program was already licensed to the Engineering department, and the software development required only man hours which incur no additional costs as all the team members are students.

2.6 Gantt Chart

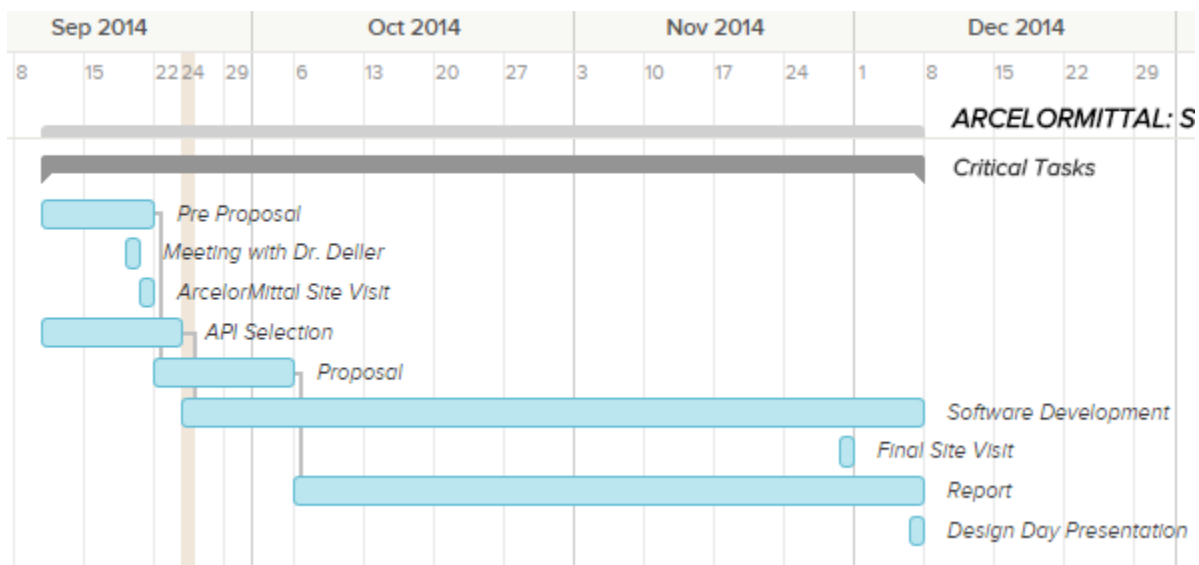


Figure 2 - Gantt Chart

Chapter 3 - Final Design

3.1 Hardware Setup

3.1.1 Setup Axis Camera to Local Network

The Axis Camera hardware must be powered and connected to a local network in order to access the streaming video for image processing purposes. Power is delivered to the camera via an Ethernet cable and a Power over Ethernet (PoE) injector. The injector takes power from a standard 120V outlet and couples it to the same cord used for data transfer. Then an additional Ethernet cord is connected from the injector into a network switch. The network will assign the camera to a local IP address. The default address for Axis cameras is 192.168.0.90. The camera configuration can be verified by typing that address into an internet browser located on the local network on which the camera is located.

3.1.2 Computer for Image Processing Specifications

Image processing is no easy task, especially with 1080p resolution at 30fps. The demand on the CPU is significant. Initially a laptop with a Centrino CPU was used for software development, but our design solution required a more powerful, current processor. Brian Wright from the ECE shop lent us a brand new Dell laptop. The development testing computer is equipped with a 64 bit version of Windows 7, 8.00 GB of RAM, and an Intel(R) Core™ i5-4310M CPU clocked at 2.70 GHz. For execution of the final solution application, it is recommended that a computer with at least these equivalent specifications be used. Accuracy and performance of the software solution on an inferior computer cannot intrinsically be trusted.

3.2 Software Setup

3.2.1 Image Processing Toolbox MATLAB

“Image Processing Toolbox provides a comprehensive set of reference-standard algorithms, functions, and apps for image processing, analysis, visualization, and algorithm development. You can perform image analysis, image segmentation, image enhancement, noise reduction, geometric transformations, and image registration. Many toolbox functions support multicore processors, GPUs, and C-code generation.”³ This addition to the basic MATLAB package is absolutely necessary because it contains many additional functions that are not included in MATLAB. The Image Processing Toolbox was favored over most OpenCV functions because of the extensive, readily available, and built in documentation and support of each image processing function as well as the intuitiveness of the MATLAB language. The Image Processing Toolbox was licensed and supported on all Division of Computing Services (DECS) computers. Installation of the toolbox on the Image Processing Computer (IPC) supplied by the ECE Department was provided by DECS.

3.2.2 OpenCV

“OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc.”¹ By utilizing this sophisticated library we have overcome the issue of accessing the video stream of the IP camera in MATLAB. Further discussion on the technical involvement of OpenCV with our software is covered in Section 3.3 Software Design. The version of OpenCV used with our software is 2.4.9. Reference Appendix 3 for the link to download OpenCV2.4.9. The .zip package was extracted to the C directory such that the file path became C:\opencv.

3.2.3 MATLAB MEX-files

“MEX stands for MATLAB Executable. A MEX-file (also written as MEX file) provides an interface between MATLAB and subroutines written in C, C++ or Fortran. When compiled, MEX files are dynamically loaded and allow non-MATLAB code to be invoked from within MATLAB as if it were a built-in function.”² Fortunately, there is a library of MEX-files available that allows for the pairing of OpenCV and MATLAB to even be possible. Reference Appendix 3 for the link to download the MEX-file Library. The Mex-file library folder must be located along with the MATLAB project code (script files and function files). A folder named Project on the desktop contains the previously mentioned files and libraries pertinent to the software solution. In MATLAB the MEX-file Library folder called *mexopencv-master* must be added to the path. In the *Current Folder* navigator within MATLAB the current folder was made to be *Project*. The folder *mexopencv-master* was then added to the path by right-clicking and selecting *Selected Folders and Subfolders*. This makes sure MATLAB utilizes all files that are a subset of *mexopencv-master*.

In order for the MEX-files to be correctly linked to the OpenCV API directory, local user and global system environment variables must be created. Adding these environmental variables allows MATLAB to access and utilize the powerful C++ image processing functions of the OpenCV API. Within the *System Properties* window edit the *Environmental Variables*. The following *User Variables* must be created:

Variable: LIB

Value: C:\opencv\build\x64\vc12\lib; C:\opencv\build\x64\vc12\staticlib

Variable: LIBPATH

Value: C:\opencv\build\x64\vc12\lib; C:\opencv\build\x64\vc12\staticlib

Variable: PATH

Value: C:\opencv\build\x64\vc12\bin

The following *System Variables* must be edited:

Variable: Path

Value: C:\opencv\build\x64\vc12\bin

The team completed the MEX-file installation for MATLAB by running the following code with the *Command Window* in MATLAB to compile the MEX-files:

mexopencv.make('opencv_path','C:\opencv\build')

The procedure of adding the *mexopencv-master* folder to the path and compiling the MEX-files must be done on each new session of running MATLAB.

3.3 Software Design

With the discovery of OpenCV and MEX-files, MATLAB became the primary platform in which to develop a software solution for ArcelorMittal. As previously stated, the pairing of the two allowed the team to overcome the issue accessing the video stream of the IP camera in MATLAB. This section of the report will cover the MATLAB software (m-code) in detail and describe the procedure for measuring a coil diameter and communicating with other systems via TCP/IP. The layout below shows the basic software structure.

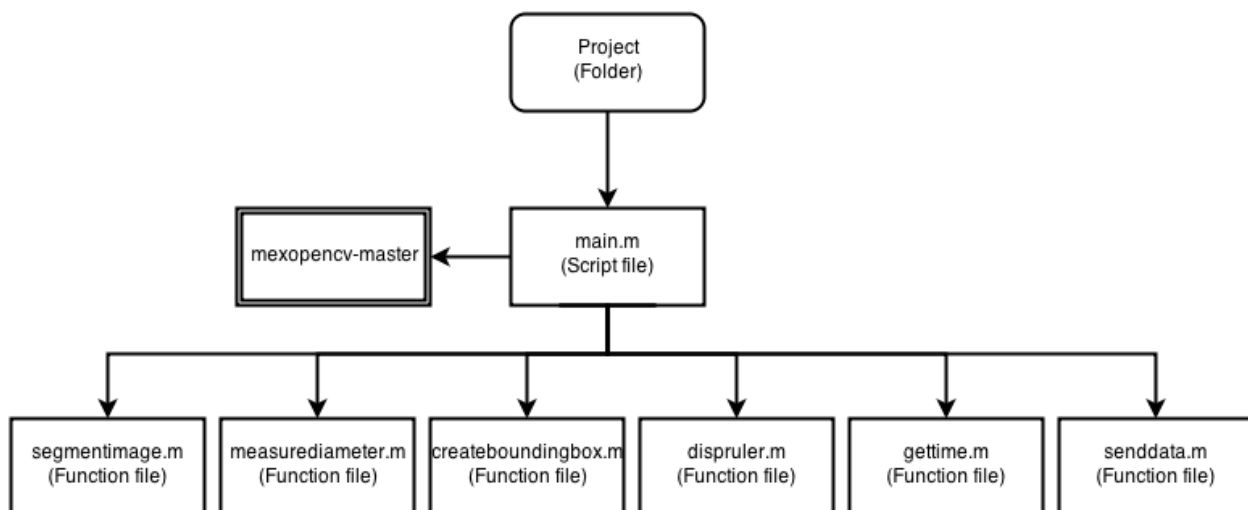


Figure 3 - Program Diagram

3.3.1 Loading an Image from a Live Video Stream

The benefits from OpenCV and MEX-files are displayed within the first few lines of the software. Within the MATLAB script file *main.m* the OpenCV function *cv.VideoCapture.open* enables us to open the live video stream from the Internet or local network. Another function called *cv.VideoCapture.read* loads each frame read by the function and saves it to a variable in the MATLAB workspace called *im*. The variable *im* is a matrix with dimensions that are identical to the resolution of the Axis IP Camera. Refer to the *main.m* script file in Appendix 3.

3.3.2 Segmenting an Image

Segmenting an image is confined to a function (*segmentimage.m*) written by the team. The function *segmentimage* takes variable *im* as an input and immediately converts the RGB color image into a binary image using the built in MATLAB function, *im2bw*. A binary image is one that is either black or white. The point at which color values are assigned to be either white or black is the image threshold. By experimenting with the image threshold level (between zero and one) the bright steel coil can be differentiated from the dark background. The image below depicts the simulation of a real world coil. The threshold chosen for the simulation was 0.6. A black poster board was used to simulate the dark background that will be encountered in the real world. To simulate a bright steel, coil cake board circles of different diameters were purchased. The 10 inch cake board circle is depicted in the image below. As a result of the contrast difference the image was segmented to isolate the desired object from the background.

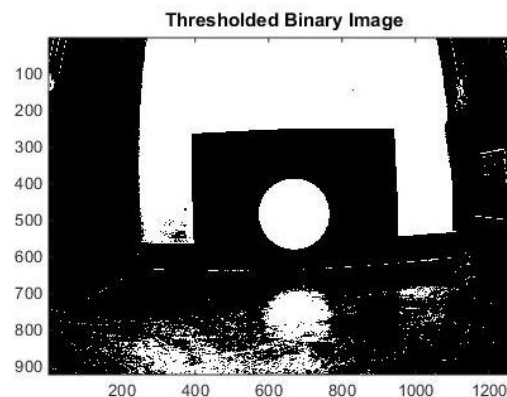


Figure 4 - Threshold Binary Image

To improve the image quality further filtering was performed on the above image. The MATLAB function *imclearborder* removes white Blobs/regions connected to the border of the image. This report defines Blobs to be any collection of white pixels that are adjacent to one another to create a cohesive and distinct object. The result of *imclearborder* is shown in the image below.

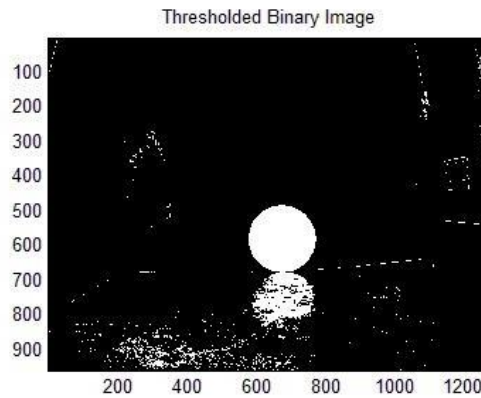


Figure 5 - Threshold Binary Image with *imclearborder*

Another filtering technique used is a combination of two MATLAB functions called *strel* and *imopen*. The function *strel* stands for structure element and creates a structured element called *se* in the shape of a disk with a radius of 20 pixels. By using this structured element in combination with function *imopen* removes all Blobs that *se* cannot fit inside. The final step to filtering was using the function *bwarea* which can eliminate all Blobs except the largest. This function provides a safety net in case there is more than one blob in the image since the remainder of the software cannot process more than one Blob. Refer to *segmentimage.m* in Appendix 3 for the correct syntax and exact usage of the variables and functions described above.

3.3.3 Measuring Objects within an Image

Once the image or frame has been completely segmented with only one Blob, the custom built function *measurediameter* was used to determine the major axis length. The major axis length is the desired length and is defined to be equivalent to the diameter of an object. The function *measurediameter* uses the built in MATLAB function *regionprops* to determine the major axis length of the image outputted from the custom function *segmentimage*. The function *regionprops* outputs a structured variable called *temp*. The variable *pixeldiameter* is then extracted from the struct *temp*. With *pixeldiameter* extracted, the calibration function then converts the pixel distance to a real measurement. The variable *actualdiameter* represents the real diameter of the blob being measured. The calibration function is $y = 0.0517x - 0.002639$ where the variable *y* represents the *actualdiameter* and the variable *x* represents *pixeldiameter*. Refer to *measurediameter.m* in Appendix 3 for the correct syntax and exact usage of the variables and functions described above.

3.3.4 Communication via TCP/IP

As previously stated, the goal of this design solution is to provide an accurate measurement of the produced coils of steel. ArcelorMittal's Programmable Logic Controller contains their customers' order along with all of the specifications of the roll. These specifications include steel gauge (thickness) and most importantly for this solution, the desired diameter length. This design solution needs automated and instantaneous access to these values. The desired diameter

length input into our software is paramount. This transfer of information between the PLC and our Image Processing Computer is accomplished by sending data packets with the TCP/IP or Transmission Control Protocol/Internet Protocol. The PLC computer will be outputting this string of values onto a network address and a port. The message format of the PLC computer is shown in Appendix 3.1. This message is retrieved from the address and port with the *getdata* function. *Getdata* can be referenced in Appendix 3.2.6. This function utilizes the *tcpip* function built into MATLAB's Instrument Control Toolbox. This toolbox does not come standard with MATLAB R2014b, but it was able to be utilized by our solution with the permission of Dr. Robert J. McGough who personally owns and lends out five licenses to the Engineering Department. This function instructs MATLAB to either **write** or **listen** to information on the specified address and port. Our code **listens** to the specified port and pulls the values into the MATLAB Workspace. Positioning *getdata* inside of *main* repeatedly references and reports the *desireddiameter*. The real time diameter value that the camera is measuring must also be sent back to the PLC. This is accomplished with *senddata*. *Senddata* can be referenced in Appendix 3.2.4. This function once again utilizes *tcpip*, except now the software is **writing** information to the specified address and port. This address and port can be easily accessed by the PLC in order to achieve the final goal of accurate steel coil diameter length verification.

3.3.5 Other Features (Bounding Box, Visual Ruler)

Powerful visual add-ons to help the operator or to demo the software are the functions *createboundingbox* and *dispruler*. Beginning with *createboundingbox*, the purpose of this function is to create a bounding box around the only and largest Blob in the original image. The bounding box is colored red as long as the *desireddiameter* variable from the Level Two Computer is larger than the variable *actualdiameter*. Once the *actualdiameter* is equal to or greater than the *desireddiameter* the bounding box becomes green signifying that the object has achieve the desired diameter. *createboundingbox* outputs variables *x* and *y*; more on their usage below..

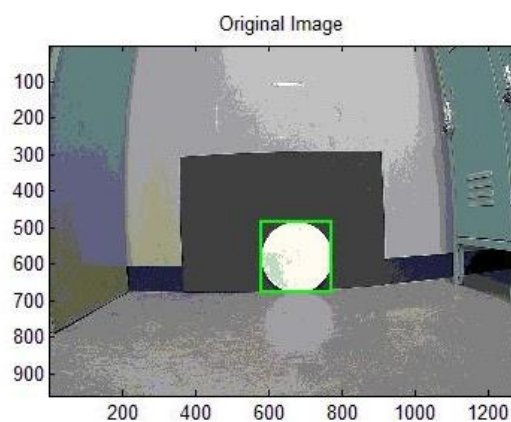


Figure 6 - Bounding Box Green

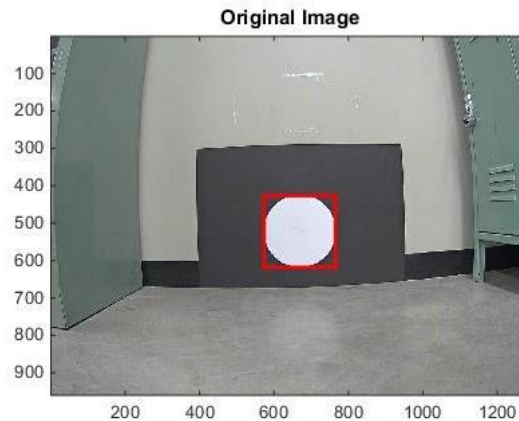


Figure 7 - Bounding Box Red

Another feature added is a ruler that moves and changes length with the identified Blob. Function *dispruler* takes inputs *x* and *y* outputted from the *createboundingbox* function and uses that as a reference to position the left hand endpoint of the ruler. Variables *x* and *y* are the coordinates of the top left corner of the bounding box. The length of the ruler is controlled by the variable *pixeldistance* which is the output of the function *measuredistance*. This variable is the third and final input to the function *dispruler*. Refer to *createboundingbox* and *dispruler* in Appendix 3 for the correct syntax and exact usage of the variables and functions described above.

3.4 Design Issues and Solutions

3.4.1 Blob Robustness

The program that the team wrote to analyze the incoming video and output the diameter of the steel coil uses blob analysis as mentioned in previous sections. This program works well when there is a blob present in the image but when there is no blob present the program fails. This flaw was corrected in laboratory tests by always having an object that will generate a blob present before the program begins to run. This flaw however will be detrimental when the program is applied to the pickle line as there will be a significant time at the beginning of each run when there will be no blobs present. Re-writing the program without this flaw would require starting over and possibly taking an entirely different approach so a different solution was needed. So that the program will not fail while the steel is beginning to coil a small artificial static blob was added to the image after the image was filtered. This artificial blob will stop the program from timing out as the pickle line begins to run but as soon as the blob representing the coiling steel grows to be larger than it, it will be ignored. This artificial blob is a failsafe against the image taken from the Axis P1355 camera not producing any blobs and causing the program to time out.

Chapter 4 - Calibration & Testing

4.1 Calibration

The design consisted of hardware and software components. The hardware products were all purchased so manufacturer calibration was assumed. This includes the Axis Communications P1355-E Network Camera, the 15W Midspan Injector for Power over Ethernet, and a laptop with enough computing power to run the software solution. Once the software and coding part of the design was completed, the calibration and testing process could begin.

4.1.1 Calibration Data & Setup

The calibration process was one of the most important parts of the design as it determined the overall accuracy of the measurements and results. In the factory, the camera is setup approximately 38.6 feet from where the steel is being coiled. This correlates to 463 inches. The largest coil that can be produced from the #5 pickle line is 98 inches in diameter. To make the scaling process simplified, it was assumed that the coil size could reach an even 100 inches. The team accepted this compensation since this diameter is outside of the bounds that the machine can coil. Overall the team's calibration setup was scaled down by a factor of 10.

The three objects being measured for calibration were white plastic disks with diameters of 6 inches, 8 inches, and 10 inches which correspond to a coil measuring 60 inches, 80 inches, and 100 inches respectively. The camera was then setup at increasing distances from the disk being measured. The distance started at 6 inches, taking 6 inches steps, and ended at 60 inches. The images of the disks created by the code are shown below. Figure 11 depicts the relationship of the measured object diameter (pixels) as a function of distance (inches) away from the object. This supplied the team with three data points of the disks from each of the distances.

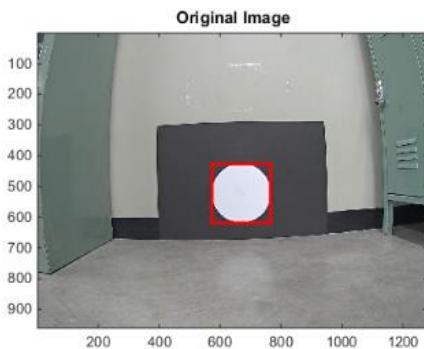


Figure 8 - Original Image

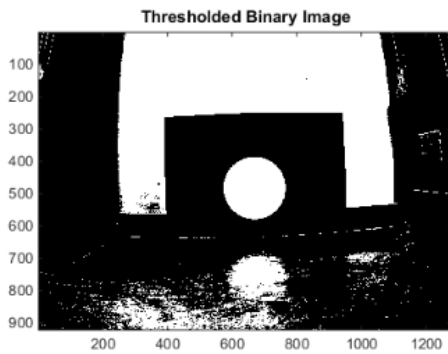


Figure 8 - Thresholded Binary Image 2

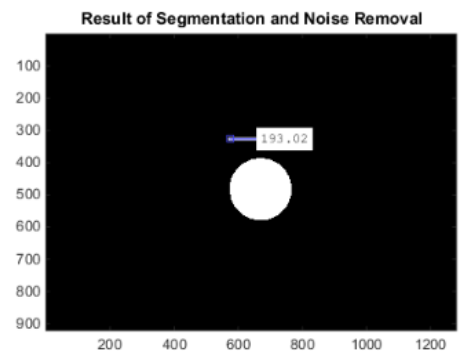


Figure 10 - Segmentation Complete

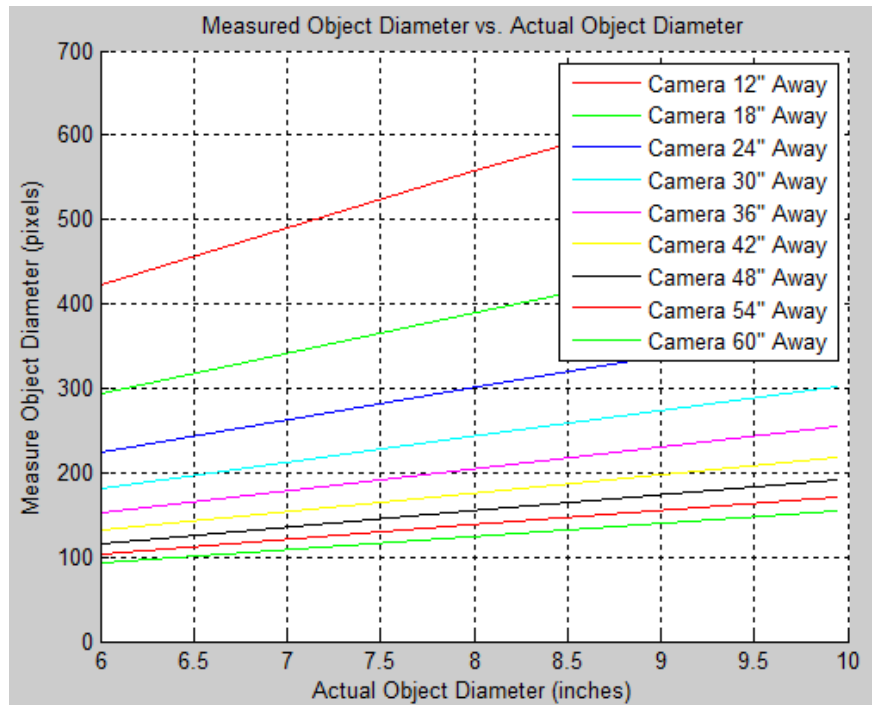


Figure 9 - Calibration Lines

Equations were created from these points by using lines of best fit. The equation that was chosen by the team was a modified version of when the camera was 48 inches away. The equation at 48 inches is $y = 0.0517x - 0.002648$. In order to make this equation fit for a distance of 46.3 inches the y-intercept was adjusted up, as to be closer to the equation for where the camera is 42 inches away. The modified equation for a distance of 46.3 inches is $y = 0.0517x - 0.002639$. This equation is used as the team's calibration for the design and put into the code.

4.2 Testing & Proof of Functionality

Testing of software was a repetitive process, but it was crucial to ensure the team's calibration for object diameter is correct. Many different objects were taken and measured by the computer vision system and compared to their actual measurement. The percent error of the team's code was calculated by using the three plastic disks. The code was modified so it would record the highest and lowest measurement by the system. This gave us the magnitude of the pixel ripple. With this code, the measurement that was furthest away from the actual measurement of the disk was used in the calculation of the percent error. The test was done on each disk. The camera was set to measure for five minutes where the code continuously ran and updated the highest and lowest values recorded. This test was completed 15 times for each disk. The maximum percent error allowed by ArcelorMittal is 1.020408%. The average percent error across all the completed tests is 0.14%. This verifies that the accuracy of the system is within the required value given by ArcelorMittal.

Chapter 5 - Summary & Conclusion

5.1 Summary and Conclusions

Overall, the project was successful and performed very well. The Axis P1355-E Network Camera was able to detect the specified object, a white cake board, and measure the diameter accurately to within 0.14% which is about 6.8x better than the requirement given by ArcelorMittal. The video input is very clear with no distortion seen in the objects edge resulting in excellent accuracy. With our measured accuracy our smart camera system solution is a viable coil diameter measurement technique.

5.2 Success and Failures

The team goal was to provide an accurate coil diameter measurement technique utilizing a “Smart” Camera solution. The team goal was to have a greater accuracy with the smart camera system compared to the current Programmable Logic Controller (PLC). The team was very successful because smart camera was tested and is shown to have an accuracy that is 6.8x better than the requirement given by ArcelorMittal. The current PLC has a large error due to slippage of the coil on the tension reel. Our design solution in MATLAB allows the team to remove that error completely from the system. The Axis camera measures the diameter of the steel coil by segmenting the image and getting rid of unwanted blobs/regions of the image for a better measured result. A “Smart” Camera measurement solution eliminates the variability of the steel’s gauge which will cause variability in the diameter measurement of their current system.

As a result of the robustness of the design the solution, Team 5 anticipates no failures on the execution of all the deliverables specified by ArcelorMittal.

5.3 Suggestions for the Future

There are improvements that should be made in future revisions. Instead of using the Axis P1355-E Network Camera, a GigE Vision standard camera should be used. MATLAB has built in support for GigE Vision type cameras. This would help software development by eliminating the use of OpenCV and MEX files. The only toolbox needed for this standard of camera is the Image Acquisition Toolbox. This would also allow a GPU hardware acceleration to be utilized for faster image processing. Lastly more research could also be done using different cameras, smartphones, and smartphone apps to see if there is anything better suited for the job with all aspects from cost to functionality and accuracy.

5.4 Final Costs

In order to construct this design project from start to finish, the total budget is listed below.

Components	Quantity	Costs
Axis P1355-E Network Camera	1	\$1100
15W PoE Midspan Injector	1	\$50
CAT5e Ethernet Cables	1	\$50
Windows 7 Desktop Computer	1	\$700
MATLAB/Simulink Software	1	\$2,150
Image Processing Toolbox	1	\$1,000
Instrument Control Toolbox	1	\$1,000
OpenCV API Computer Vision	1	\$0
Total Cost		\$6,050

Table 2 – Cost of Components

5.5 Final Budget:

The budget shown below is what a company would need to purchase in order to implement the designed Smart Camera solution Group 5 has produced.

Components	Quantity	Costs
Axis P1355-E Network Camera	1	\$1100
15W PoE Midspan Injector	1	\$50
CAT5e Ethernet Cables	1	\$50
Windows 7 Desktop Computer	1	\$700
Code	1	\$0
Total Cost		\$1,900

Table 3 – Cost of Product

5.6 Final Schedule

The final schedule is shown below. This table shows the progress throughout the semester and when each technical task was accomplished and the duration each task. Some delays in the project were waiting to get the Image Processing Toolbox into MATLAB and going on our site visit to receive the Axis P1355-E Network Camera with the 15W PoE Midspan Injector.

Task	Duration	Start Date	End Date
Research	16 days	Fri 9/5/14	Sun 9/21/14
Start	0 days	Fri 9/5/14	Fri 9/5/14
Problem statement	5 days	Fri 9/5/14	Wed 9/10/14
Work through designs	7 days	Thurs 9/11/14	Thurs. 9/18/14
Components list	7 days	Thurs 9/11/14	Thurs. 9/18/14
Compile all research	2 days	Fri 9/19/14	Sun 9/21/14
End	0 days	Sun 9/21/14	Sun 9/21/14
Design	71 days	Mon 9/22/14	Tues 12/2/14
Start	0 days	Mon 9/22/14	Mon 9/22/14
Schematic layout	2 days	Mon 9/22/14	Wed 9/24/14
Simulink programming	15 days	Thurs 9/25/14	Fri 10/10/14
MATLAB code	68 days	Thurs 9/25/14	Tues 12/2/14
MATLAB/OpenCV code implementation	20 days	Wed 11/12/14	Tues 12/2/14
Prototype hardware Design's	7 days	Mon 9/22/14	Mon 9/29/14
End	0 days	Tues 12/2/14	Tues 12/2/14
Parts Order	1 day	Thurs 10/2/14	Thurs 10/2/14
Start	1 day	Thurs 10/2/14	Thurs 10/2/14
End	1 day	Thurs 10/2/14	Thurs 10/2/14
Prototype Construction	71 days	Thurs 9/25/14	Tues 12/2/14
Start	0 days	Mon 10/6/14	Mon 10/6/14

Axis P1355-E Network Camera	3 days	Mon 10/6/14	Thurs 10/9/14
15W PoE Midspan Injector	1 day	Tues 10/7/14	Tues 10/7/14
Windows 7 Desktop Computer	6 days	Fri 10/10/14	Wed 10/16/14
Programming	68 days	Thurs 9/25/14	Tues 12/2/14
End	0 days	Tues 12/2/14	Tues 12/2/14
Testing	19 days	Mon 10/17/14	Wed 11/5/14
Start	0 days	Mon 10/17/14	Mon 10/17/14
Receiving Video	4 days	Fri 10/17/14	Tues 10/21/14
Image Segmentation	3 days	Wed 10/22/14	Sat 10/25/14
Measuring Diameter	5 days	Sun 10/26/14	Fri 10/31/14
Sending Data	4 days	Sat 11/1/14	Wed 11/5/14
End	0 days	Wed 11/5/14	Wed 11/5/14
Improvement Test Stage	25 days	Thurs 11/6/14	Tues 12/2/14
Start	0 days	Thurs 11/6/14	Thurs 11/6/14
Image Segmentation Accuracy	6 days	Thurs 11/6/14	Wed 11/12/14
Diameter Accuracy	19 days	Thurs 11/13/14	Tues 12/2/14
End	0 days	Tues 12/2/14	Tues 12/2/14
Presentation Preparation	37 days	Wed 10/29/14	Fri 12/5/14
Start	0 days	Wed 10/29/14	Wed 10/29/14
Prototype Presentation 1	1 day	Wed 10/29/14	Wed 10/29/14
Prototype Presentation 2	1 day	Wed 11/26/14	Wed 11/26/14
Final Report	1 day	Wed 12/3/14	Wed 12/3/14
Final Presentation	1 day	Fri 12/5/14	Fri 12/5/14
End	0 days	Fri 12/5/14	Fri 12/5/14

Table 4 - Schedule

Appendix 1 - Individual Contributions



Note: Individual contributions are ordered left to right in accordance with the picture above with exception to Paul Han (not pictured).

Joe McAuliffe

My first technical responsibility was connecting the Axis P1355 camera to the team's working laptop. To do this the camera first had to be connected to the Power over Ethernet switch. The other outlet of the switch then had to be connected to the laptop's Ethernet port. The correct ports on the laptop then had to be enabled so that Windows's firewall would allow the camera to communicate with software on the computer. This could be accomplished by allowing those ports in network preferences or setting the network type to a home network so that Windows automatically allows the ports that are needed.

Beyond the setup of the camera there was no physical circuitry or components in our project so the rest of the technical work was code writing. Throughout the software development we had many iterations, sections of code, and attempted strategies that did not make it into the final program or even make it into a functional program. One such strategy was a live video stream to a file location and then reading said file into Simulink. I set up a live stream but as a team we moved to MATLAB code instead of Simulink blocks and the live stream was not compatible with MATLAB code and therefore was not used. I also set up a way to turn an image into a binary matrix in MATLAB and then read that image into Simulink so that it could be processed but again all of our processing is now done in MATLAB so this quickly became obsolete. I also wrote a section of code to read numbers out of a MATLAB struct so that they can be utilized as integers later in the MATLAB code. As code writing is one of my weaker skills I did not write

significant portions of the code but more provided support to my teammates with stronger coding skills. I help troubleshoot code and bridge gaps between different sections of code. For example, near the end of our software development we had a software error that triggered when no blobs above the threshold were present in the camera's field of view. In lab testing we were able to remedy this by always having a large enough circular object present at the start of the program. We then realized that this approach would not work in our application because each run of the pickle line will begin with no blobs present in the filtered image. I came up with the idea to add a constant static blob to the center of the filtered image that the program will measure until a larger blob is present. This blob will be small enough that it will not affect the measurements once the steel starts coiling but will allow the program to run until it does.

James Quaglia

I had a very prominent technical role in the design and execution of our “Smart Camera” solution. As manager, I felt it was my responsibility to have a solid understanding of every technical aspect of this project. I did incredible amounts of initial research, in order to fully understand the capabilities of IP Cameras, network configurations, API integration, edge detection, pixel analysis, image processing, CPU demand, etc. Within the first two weeks, I completely built a Simulink model that could accomplish every one of our deliverables. This had to be scrapped because there was no viable way to live feed our Axis camera into the model. If we had been given a camera that complied with GigE Vision standards, live feed integration into the model would have been possible. Regardless, after we determined my initial solution was not viable, I went back to work. I researched the API OpenCV and configured the libraries to integrate into Visual Studios 2013 IDE on a Windows 7 computer and also a Virtual Machine running Ubuntu 14.04 in Eclipse IDE. The image processing on these solutions lagged far behind reality because only one CPU core was being utilized. These solutions would have required a profound knowledge of parallel processing commands in order to distribute computing responsibilities across multiple cores. I researched and found out that these techniques have entire CSE courses dedicated to them. Once again I went back to the drawing board. I discovered that MATLAB R2014b has built in multi-core support and Graphical Processing Unit hardware acceleration features. The only thing left to do was integrate the live video feed into MATLAB. MATLAB's built in methods for frame grabbing from an IP address are far inferior to OpenCV's powerful computer vision functions. It was at this time that I learned about MEX-files. I heavily researched and found a bridge between Visual Studios 2013, MATLAB, and OpenCV. By compiling OpenCV MEX-file libraries inside of our Current Folder in MATLAB, I was able to use the powerful OpenCV C++ computer vision function inside of the MATLAB environment. This allowed us to use the powerful frame grabbing power of OpenCV and Visual Studios 2013 along with the multi-core support of MATLAB R2014b to feed live video at a high framerate and resolution and perform real-time video analysis. There were many times during the semester that I truly believed a “Smart Camera” solution was not possible with the Axis P1355E. All of these problems could be avoided if a GigE Vision standard camera was used. Regardless, I kept researching and delivered the solution for a viable software development

environment. At this point function development was delegated to each of the group members, most specifically, Matt Wesolowski, whose knowledge of MATLAB syntax is quite impressive. I also envisioned the method with which we would calibrate our system, which was performed by Matt and Ian. In addition to the previously stated technical roles, I also successfully developed and tested the *getdata* and *senddata* functions for communication with the factories PLC. I had significant contributions to every technical writing assignment and also spoke during the technical presentation.

Ian Siekkinen

The two main technical roles I played in the project were calibrating the pixel distance and creating a bounding box around the found blob. In order to calibrate the pixel distance into a real diameter, I had to set up a series of tests that would in turn give us a function for our code. Using the three plastic disks with diameters of 6in, 8in, and 10in I created three similar points of reference when moving the camera to different distances. At each separate distance of the camera the pixel distance was measured for each disk. Doing this gave me a best fit line at each separate distance across all of the disks used. I decided to use the function of the best fit line when the camera was at a distance of 48in as a reference. Given that our scaled down version for the distance of the camera away from the coil is 46.3in, this function is close enough to one that can accurately measure the coil. From this function, and comparing to the best fit line at 42in away, I was able to modify the function of the 48in line and change the y-intercept to accurately represent when the camera was 46.3in away.

My next role was to create a bounding box that would surround the blob. This part of the code has two inputs, one being the segmented image, and the other being the box color. The box color is either red, which means that the blob has not reached the desired diameter, or green, which means the blob has reached the desired diameter. The function used to help create this box is called *regionprops* and creates the parameters of the bounding box, ie. exact points of the corners. The parameters are *x*, *y*, *width*, and *height* where *x* and *y* is the top left corner of the box, and from that, the *width* and *height* are used to create the rest of the box. With all these I used the function *rectangle* to grab the parameters and the box color to create the final box around the blob. I also had roles in every piece of technical writings and presentations that the team completed.

Petros Taskas

The main technical role I played in the project was to have the program measure the diameter of the steel coil. I used white plates of different sizes, 6, 8, and 10 inches in diameter to do the testing. These values were scaled down versions to correlate to the steel coil diameter sizes of 60, 80, and 100 inches in diameter. To do this I made a function in MATLAB that is called in the main program. This function uses the *regionprops()* toolbox built into MATLAB that creates a blob from the image. The toolbox has many built in definitions to measure different properties of image regions. What our project needed was a measurement along the major axis of the image to calculate the diameter of the steel coil as it is being wound. To do this I create a variable called

temp which is set equal to `regionprops(image,'MajorAxisLength')`. What this does is it creates a blob around the region of the image that is of interest. Where 'MajorAxisLength' is a, “Scalar specifying the length (in pixels) of the major axis of the ellipse that has the same normalized second central moments as the region” (*Measure Properties of Image Regions*). In the next step I create another variable for called `pixeldiameter` which should guarantee only 1 blob in the image at all times and get the pixel diameter. After these steps are done calibration is needed to get the actual diameter of the image. What I did next was multiply the `pixeldiameter` by a constant 0.0517 and then subtract 0.002639 to get the diameter in inches. This calibration of the image was to get the most accurate diameter measurement and was the most time consuming part of this code. There was quite a bit of tweaking of the scalers that went into this to get the average error down to 0.14% which is 6.8x better than the requirement given by ArcelorMittal. Other than that, I also had roles in the technical documents of the project including the proposal, design issues paper, technical presentation, and the final report.

Matthew Wesolowski

My primary responsibilities were creating an algorithm that allowed for the coil to be identified in an image and, as a result of my excellent MATLAB programming knowledge; I was responsible for managing the program development. Initial research led to a conclusion that the most appropriate method for detecting the coil was using edge detection via premade algorithms such as the Sobel and Canny methods. I identified that MATLAB when paired with the Image Processing Toolbox was a power platform to develop our “Smart” Camera. The images below show the result of the Sobel method using MATLAB.

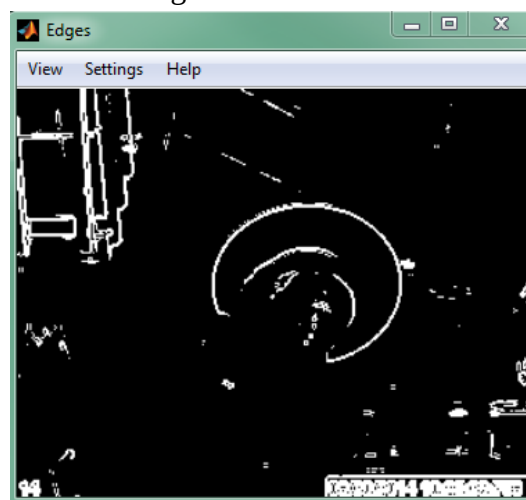


Figure 12 – Sobel Method Edge Detection

Unfortunately, there are many shortcomings with the Sobel Method. The Sobel Method did not distinguish the coil well enough from the rest of the image. As a result, identifying the coil proved difficult by inspection let alone with software. Additionally, the Sobel Method did not create Blobs/regions that are necessary for built in MATLAB functions such as *regionprops*. The

method for achieving the desired requirement came from thresholding and filtering an image based on differences in the intensity (contrast) of the image. As described in Chapter 3 Section 3.2 the process of segmenting involved using functions such as *im2bw* with a specified threshold level to differentiate the background from the desired image. Functions such as *imclearborder*, *imopen*, and *bwarea* provided methods for filtering/removing unwanted Blobs/regions from the image. The custom made function *segmentimage* provides a much less computationally intensive method of isolating the desired object, specifically the cake board circle stand-in, from the background. Additional contributions to solutions not making the final solution were linking the OpenCV library to Microsoft Visual Studios 2014.

Poyuan Han

The main technical role that I played in the design was programing a system that can display the current measured diameter's size. The method is evident in the function *dispruler*. The purpose of this method is to get the current value of the detected object's diameter even if it is changing rapidly. First, we need to have a starting position set up. In Appendix 3.2.8, it shows function[] = *dispruler*(x,y,length). The x and y is the coordinate set up for the image, remains in the same position in each measurement. The length is known as the size display. After setting up the initial coordinate points, the distance display set up is next. *Imdistline* creates a distance tool on the current axes. In order to display distance, both startpoint and endpoint need to be set up. The size only changes horizontally, so we set up x as a start point, and x+length as an endpoint while the size is changing. Since the size is only changing horizontally, there is no need to set up the y axis. [y-100 y-100] is to set the display ruler stay at the original position. The display ruler method made the design result more accurate. It would be useful for ArcelorMittal to get the desire size of coil.

Appendix 2 - References

Literature and website references

"Documentation." *C/C Source Files*. Web. 3 Dec. 2014.

<<http://www.mathworks.com/help/matlab/write-cc-mex-files.html>>.

"Documentation." *Call MEX-File Functions*. Web. 03 Dec. 2014.

<<http://www.mathworks.com/help/matlab/mex-file-functions.html>>.

"Documentation." *Measure Properties of Image Regions*. Web. 03 Dec. 2014.

<<http://www.mathworks.com/help/images/ref/regionprops.html>>.

"Documentation." *MEX Library*. Web. 03 Dec. 2014.

<<http://www.mathworks.com/help/matlab/mex-library.html>>.

"Highgui. High-level GUI and Media I/O¶." *Highgui. High-level GUI and Media I/O — OpenCV 2.4.9.0 Documentation*. N.p., n.d. Web. 03 Dec. 2014.

<<http://docs.opencv.org/modules/highgui/doc/highgui.html>>.

"Objdetect. Object Detection¶." *Objdetect. Object Detection — OpenCV 2.4.9.0 Documentation*.

N.p., n.d. Web. 03 Dec. 2014. <<http://docs.opencv.org/modules/objdetect/doc/objdetect.html>>.

"Video. Video Analysis¶." *Video. Video Analysis — OpenCV 2.4.9.0 Documentation*. N.p., n.d.

Web. 03 Dec. 2014. <<http://docs.opencv.org/modules/video/doc/video.html>>.

"Vision Online." *Vision Online*. Web. 03 Dec. 2014. <<http://www.visiononline.org/vision-standards-details.cfm?type=5>>.

Yamaguchi, Kota. "Mexopencv." - *Matlab Mex Functions for OpenCV*. Tohoku University, Japan, n.d. Web. 03 Dec. 2014. <<http://vision.is.tohoku.ac.jp/~kyamagu/software/mexopencv/>>.

Appendix 3 - Technical Documents

3.1 PLC Message Format

The basic message format is as follows:

"\$CID,FLD1,FLD2,FLD3,...<CR>"

Where:

'\$' = start of message indicator

'CID' = message command/response ID (three ASCII characters)

'FLDN' = one or more ASCII fields of variable length separated by commas

'<CR>' = ASCII carriage return character (0x0D) marks the end of the message

The "level 2 computer diameter" message will be transmitted to the camera computer at a defined interval of one second per tension reel.

Message Details:

N	Field Name	Description	Characters
1	\$	Start of message	1
2	MsgID	Message id (L2D)	3
3	,	Field separator	1
4	Date	YYYY-MM-DD	10
5	,	Field separator	1
6	Time	HH:MM:SS	8
7	,	Field separator	1
8	Reel	Tension Reel (1 or 2)	1
9	,	Field separator	1
10	Diameter	Coil Diameter 99.9"	4
11	,	Field separator	1
12	Speed	Line Speed 9999 fpm	4
13	CR	End of Message	1

Example: \$L2D,2014-01-05,15:49:03,2,68.3,1237<CR> which stands for a coil diameter of 68.3 inches on #2 tension reel and line speed of 1237 ft/min.

The "camera computer diameter" message will be transmitted to the level 2 computer at a defined interval of one second per tension reel.

Message Details:

N	Field Name	Description	Characters
1	\$	Start of message	1
2	MsgID	Message id (L2D)	3
3	,	Field separator	1
4	Date	YYYY-MM-DD	10
5	,	Field separator	1
6	Time	HH:MM:SS	8
7	,	Field separator	1
8	Reel	Tension Reel (1 or 2)	1
9	,	Field separator	1
10	Diameter	Coil Diameter 99.9"	4
11	,	Field separator	1
12	Speed	Line Speed 9999 fpm	4
13	CR	End of Message	1

Example: \$CAD,2014-09-08,23:59:59,1,96.9<CR> which stands for a coil diameter of 96.9 inches on #1 tension reel.

3.2 Matlab Functions/Scripts

3.2.1 main.m

```
clear;
clc;

warning('off','all');

% Desired Diameter
[desireddiameter] = getdata();

% Set up variables
URL1 = 'ece480group5-cam.dhcp.egr.msu.edu';
URL2 = 'plazacam.studentaffairs.duke.edu';

% Set up camera
camera = cv.VideoCapture;
camera.open(['http://' URL1 '/mjpg/video.mjpg']);
pause(1); % Necessary in some environment. See help cv.VideoCapture

% Set up display window
```

```

window = figure('KeyPressFcn',@(obj,evt)setappdata(obj,'flag',true));
setappdata(window,'flag',false);

% Start main loop
while true
    tic

    % Grab and process an image
    im = camera.read;
    subplot(2,2,1); subimage(im); title('Original Image');

    % Segment the original image
    [bw_done, bw_thres] = segmentimage(im);

    % Calculate the diameter of the blob within the inputted image
    [pixeldiameter, actualdiameter] = measurediameter(bw_done);

    % Changes the color of the bounding box if the desired diameter is achieved
    if pixeldiameter >= desireddiameter
        boxcolor = 'g'; else boxcolor = 'r'; end

    % Displays a Bounding Box around the blob in the most recent image created
    [x,y] = createboundingbox(bw_done, boxcolor); % Function must follow an image to display
    bounding box

    subplot(2,2,2); subimage(bw_thres); title('Thresholded Binary Image');
    subplot(2,2,3); subimage(bw_done); title('Result of Segmentation and Noise Removal');

    % Displays a ruler that automatically updates
    dispruler(x,y,pixeldiameter);

    clc;

    % Contains TCP/IP code for communicating with L2 Computer
    senddata(actualdiameter);
    disp(desireddiameter)

    % Terminate if any user input
    flag = getappdata(window,'flag');
    if isempty(flag)||flag, break; end

```

```

    pause(.000000000000000001);

    toc
end
% Close
close(window);

```

3.2.2 cv.videocapture.m

```

classdef VideoCapture < handle
    %VIDEOCAPTURE VideoCapture wrapper class
    %
    % Class for video capturing from video files or cameras. The class
    % provides Matlab API for capturing video from cameras or for reading
    % video files. Here is how the class can be used:
    %
    %   cap = cv.VideoCapture;
    %   pause(3); % Note below
    %   for t = 1:30
    %       imshow(cap.read);
    %       pause(0.1);
    %   end
    %
    % ## Note
    % In some environment, there is a concurrency issue during camera
    % initialization. To avoid unexpected crash, pause for seconds after
    % the initialization of VideoCapture object.
    %
    % See also cv.VideoCapture.VideoCapture cv.VideoCapture.read
    % cv.VideoCapture.get cv.VideoCapture.set
    %

    properties (SetAccess = private)
        % Object ID
        id
    end

```

methods

```
function this = VideoCapture(filename)
    %VIDEOCAPTURE Create a new VideoCapture object
    %
    % cap = cv.VideoCapture
    % cap = cv.VideoCapture(devid)
    % cap = cv.VideoCapture(filename)
    %
    % VideoCapture create a new instance. With no argument, it
    % connects to the default camera device found in the system.
    % You can specify camera devices by devid, an integer value
    % starting from 0. You can also specify a filename to open a
    % video file.
    %
    % See also cv.VideoCapture
    %
    if nargin < 1, filename = 0; end
    this.id = VideoCapture_(filename);
end
```

```
function delete(this)
    %DELETE Destructor of VideoCapture object
    VideoCapture_(this.id, 'delete');
end
```

```
function frame = read(this)
    %READ Grabs, decodes and returns the next video frame
    %
    % frame = cap.read()
    %
    % The method captures the next video frame and return it. If
    % capturing fails, empty array will be returned instead.
    %
    % See also cv.VideoCapture
    %
    frame = VideoCapture_(this.id, 'read');
end
```

```
function value = get(this, key)
    %GET Returns the specified VideoCapture property
    %
```

```

        value = cap.get('FPS');
    %
    % The method returns a Property value of VideoCapture.
    % PropertyName can be one of the followings:
    %
    % 'PosMsec'    Current position of the video file in milliseconds or video capture
timestamp.
    % 'PosFrames' 0-based index of the frame to be decoded/captured next.
    % 'AVIRatio'  Relative position of the video file: 0 - start of the film, 1 - end of the
film.
    % 'FrameWidth' Width of the frames in the video stream.
    % 'FrameHeight' Height of the frames in the video stream.
    % 'FPS'        Frame rate.
    % 'FourCC'     4-character code of codec.
    % 'FrameCount' Number of frames in the video file.
    % 'Format'     Format of the Mat objects returned by retrieve() .
    % 'Mode'       Backend-specific value indicating the current capture mode.
    % 'Brightness' Brightness of the image (only for cameras).
    % 'Contrast'   Contrast of the image (only for cameras).
    % 'Saturation' Saturation of the image (only for cameras).
    % 'Hue'        Hue of the image (only for cameras).
    % 'Gain'       Gain of the image (only for cameras).
    % 'Exposure'   Exposure (only for cameras).
    % 'ConvertRGB' Boolean flags indicating whether images should be converted to
RGB.
    % 'Rectification' Rectification flag for stereo cameras (note: only supported by DC1394
v 2.x backend currently)
    %
    % See also cv.VideoCapture
    %
    value = VideoCapture_(this.id, 'get', key);
end

function set(this, key, value)
    %SET Sets a property in the VideoCapture.
    %
    cap.set('FPS', value)
    %
    % The method set a Property value of VideoCapture.
    % PropertyName can be one of the followings:
    %

```

```

        % 'PosMsec'    Current position of the video file in milliseconds or video capture
timestamp.
        % 'PosFrames' 0-based index of the frame to be decoded/captured next.
        % 'AVIRatio'  Relative position of the video file: 0 - start of the film, 1 - end of the
film.
        % 'FrameWidth' Width of the frames in the video stream.
        % 'FrameHeight' Height of the frames in the video stream.
        % 'FPS'        Frame rate.
        % 'FourCC'     4-character code of codec.
        % 'FrameCount' Number of frames in the video file.
        % 'Format'     Format of the Mat objects returned by retrieve() .
        % 'Mode'       Backend-specific value indicating the current capture mode.
        % 'Brightness' Brightness of the image (only for cameras).
        % 'Contrast'   Contrast of the image (only for cameras).
        % 'Saturation' Saturation of the image (only for cameras).
        % 'Hue'        Hue of the image (only for cameras).
        % 'Gain'       Gain of the image (only for cameras).
        % 'Exposure'   Exposure (only for cameras).
        % 'ConvertRGB' Boolean flags indicating whether images should be converted to
RGB.
        % 'Rectification' Rectification flag for stereo cameras (note: only supported by DC1394
v 2.x backend currently)
        %
        % See also cv.VideoCapture
        %
        VideoCapture_(this.id, 'set', key, value);
    end
end

methods (Hidden = true)
    function successFlag = open(this, filename)
        %OPEN Open video file or a capturing device for video capturing
        %
        % ## Output
        % * __retval__ bool, true if successful
        %
        % The methods release() to close the already opened file or camera.
        % Parameters are the same as in the constructor.
        %
        % See also cv.VideoCapture.isOpened
        %

```

```

    if nargin < 1, filename = 0; end
    successFlag = VideoCapture_(this.id, 'open', filename);
end

function flag = isOpened(this)
    %ISOPENED Returns true if video capturing has been initialized already.
    %
    % ## Output
    % * __retval__ bool, return value
    %
    % If the previous call to constructor or open() succeeded, the method returns true.
    %
    % See also cv.VideoCapture.open
    %
    flag = VideoCapture_(this.id, 'isOpened');
end

function release(this)
    %RELEASE Closes video file or capturing device.
    %
    % The methods are automatically called by subsequent open() and by destructor.
    %
    % See also cv.VideoCapture.open
    %
    VideoCapture_(this.id, 'release');
end

function successFlag = grab(this)
    %GRAB Grabs the next frame from video file or capturing device.
    %
    % ## Output
    % * __successFlag__ bool, success flag
    %
    % The function grabs the next frame from video file or camera and returns
    % true (non-zero) in the case of success.
    %
    % The primary use of the function is in multi-camera environments, especially
    % when the cameras do not have hardware synchronization. That is, you call
    % grab() for each camera and after that call the slower method retrieve() to
    % decode and get frame from each camera. This way the overhead on demosaicing
    % or motion jpeg decompression etc. is eliminated and the retrieved frames

```

```

    % from different cameras will be closer in time.
    %
    % See also cv.VideoCapture.read cv.VideoCapture.retrieve
    %
    successFlag = VideoCapture_(this.id, 'grab');
end

function frame = retrieve(this)
    %RETRIEVE Decodes and returns the grabbed video frame.
    %
    % The function decodes and returns the just grabbed frame. If no frames has
    % been grabbed (camera has been disconnected, or there are no more frames
    % in video file), the function returns an empty matrix.
    %
    % See also cv.VideoCapture.read cv.VideoCapture.grab
    %
    frame = VideoCapture_(this.id, 'retrieve');
end
end
end
end

```

3.2.3 createboundingbox.m

```

function [x,y] = createboundingbox(inputimage,boxcolor)
% inputimage is of type mat
% boxcolor is of type str
% n is the number of bounding boxes of type num

st = regionprops(inputimage,'BoundingBox');
for k = 1:length(st)
    thisBB = st(k).BoundingBox;
    x = thisBB(1);
    y = thisBB(2);
    w = thisBB(3);
    h = thisBB(4);
    rectangle('Position', [x,y,w,h],'EdgeColor',boxcolor,'LineWidth',2 );
end
end

```

3.2.4 senddata.m

```
function [] = senddata(actualdiameter)
% acutaldiameter is of type float

[YYYY,DD,MM,HH,MN,SS] = gettime();

actualdiameterstring = num2str(actualdiameter, '%.1f');
dataout = ['$CAD,' YYYY '-' DD '-' MM ',' HH ':' MN ':' SS ',2,' actualdiameterstring '<CR>'];

disp('Data being sent to the L2 Comp:')
disp(dataout)

end
```

3.2.5 gettime.m

```
function [ year,day,month,hour,min,sec ] = gettime()
% All outputs of type str

t = fix(clock);
year = num2str(t(1)); %Year
day = num2str(t(2)); %Day
month = num2str(t(3)); %Month
hour = num2str(t(4)); %Hour
min = num2str(t(5)); %Minute
sec = num2str(t(6)); %Second

end
```

3.2.6 getdata.m

```
function [desirediameter] = getdata()

tcpipClient = tcpip('localhost',55000,'NetworkRole','Client');
set(tcpipClient,'InputBufferSize',320);
set(tcpipClient,'Timeout',30);
```

```

fopen(tcpipClient);
rawData = fread(tcpipClient,40,'char');
fclose(tcpipClient);
reshapedData = reshape(rawData,1,40);
temp=char(reshapedData(1, 28:31));
desireddiameter = str2double(temp);

end

```

3.2.7 tcpserver.m

```

data = '$L2D,2014-01-05,15:49:03,2,68.3,1237<CR>';
s = whos('data');
tcpipServer = tcpip('0.0.0.0',55000,'NetworkRole','Server');
set(tcpipServer,'OutputBufferSize',96);
fopen(tcpipServer);
fwrite(tcpipServer,data(:),'char');

```

3.2.8 dispruler.m

```

function [] = dispruler(x,y,length)
% x is of type int
% y is of type int
% length is of type int

imdistline(gca,[x x+length],[y-100 y-100]);

end

```

3.2.9 measurediameter.m

```

function [pixeldiameter,actualdiameter] = measurediameter(image)
% image is of type mat
% pixeldiameter is of type float
% diameter is of type float

temp = regionprops(image,'MajorAxisLength');

```

```

pixeldiameter = temp.MajorAxisLength; % segmentimage func should guarentee only 1 blob in
image at all times
% bwarea could be used for measurement
% MinorAxisLength could also be used

% Calibration code goes here
actualdiameter = pixeldiameter*.0517-0.002639;
end

```

3.2.10 segmentimage.m

```

function [bw3,bw1] = segmentimage(im)
% im is of type mat

% Remove the top black border. Necessary for Duke Live Cam
% im = im(40:end,:,:);

% Thresholding
level = 0.6;
bw1 = im2bw(im,level);
bw1(20:100,20:100) = 1;

% Noise removal
bw2 = imclearborder(bw1);
se = strel('disk',20);
bw3 = imopen(bw2,se);
% bwpropfilt is another option
% bwselect is another option
bw3 = bwareafilt(bw2,1); %This code is not supported with 2014a.
% Forces only largest blob to be displayed.

end

```