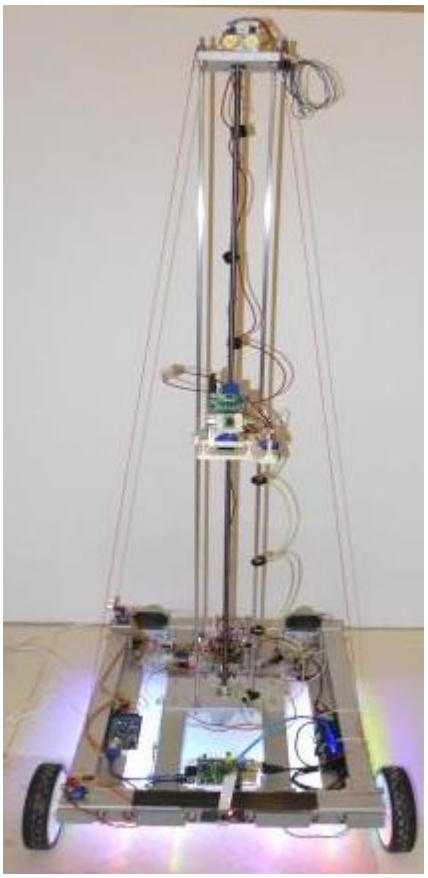
NEU Capstone 2014

Tunnel Inspection Robot



Matt Van Berlo
Sam Coe
Joshua Johnson
Joseph Robinson
Robert Watson

Professor Bahram Shafai

Table of Contents

Abstract 3

Problem Formulation 3

Analysis 4

Design 5

Mechanical 5

Simulations 6

Electrical 7

Software 8

System Software and Communications 8

Overview of System SW and Communications Design 8

Communications 9

System Servers 10

Interfacing 10

Parts and Implementation 13

Mechanical 13

Electrical 15

Software 18

System Software and Communications 18

Communications 18

Interfacing 19

Computer Vision 20

Hanger Alignment Estimation 22

Image Preprocessing 22

Color to Gray Scale Conversion 22

Subtractive Preprocessing 23

Image Segmentation 24

Image Hough Transform 25

Axis Rotation 25

Error Estimation 26

Algorithm Pseudo Code 26

Crack Detection 26

Image Preprocessing 27

Algorithm Pseudo Code 32

Cost Analysis 34

Table of Costs by Order Number 34

Cost Break Down by Category 35

Conclusion 36

References 37

Abstract

A mobile robotic system was developed as a tool to improve the efficiency and the overall accuracy of the tunnel inspection process. Important factors presented to our team by Massachusetts Department of Transportation (MassDOT) were to reduce cost, save time, and mitigate additional traffic congestion caused by these required annual inspections. Current methods for inspecting the Central Artery/Tunnel Project "Big Dig" fresh air intake and CO_2 exhaust tunnels (plenums) are time consuming, costly, and pose potential safety hazards to both the inspectors and drivers. Our system will provide a means for inspectors to conduct these inspections from a central inspection office without requiring onsite manual labor. In addition to reducing costs, time, and potential hazards, our proposed method enhances inspection reports through its user-friendly software package with integrated automated features.

While working through the engineering design process we ultimately broke the project into four categories: hardware, software, communications, and detection. During the design process we utilized 3D modeling and simulation software to develop a robust, reliable, cost efficient, easy-to-use inspection platform. Software requirements included an intuitive interface with real-time video stream and a means of controlling the mobile platform in a natural way, all over a reliable Wi-Fi communication scheme.

Problem Formulation

Starting back in 1991, the Big Dig remapped the 6-laned I93 from above the city to below via a 3.5-mile long tunnel. Intention was to rid Boston of severe traffic concerns, which caused up to ten hours of traffic per day for city drivers. Projections forecasted traffic reaching up to 16 hours a day by 2010.
In 2006, a 12t-ceiling tile fell on a car and the drivers. After, several anchors embedded in the tunnel's roof slab were discovered to be the cause [1]. Citywide tunnel inspections this problem as abundant, even in newly constructed areas [see Fig 1].

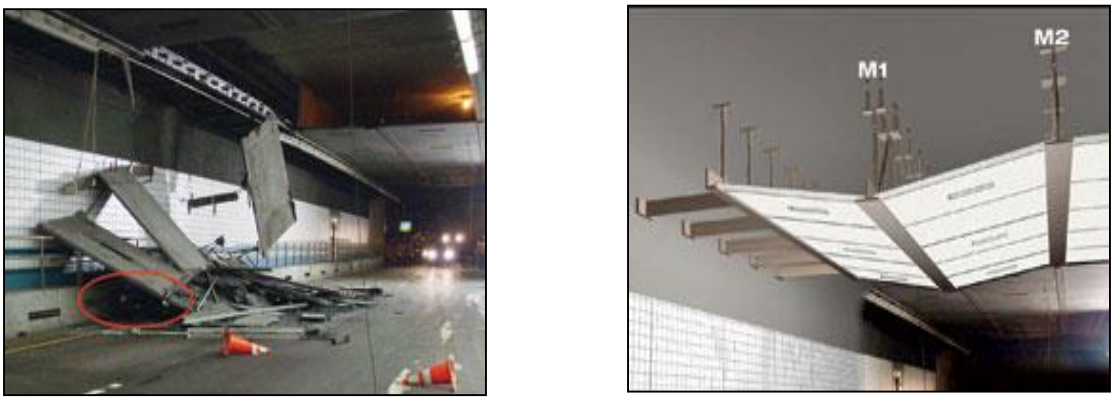


Figure 1 - The ceiling collapse (left) and a simulated view of the cause of the collapse (right)

MassDOT spends over $3M each year on plenum inspections [2]. Inspections call for lane closures, reintroducing traffic congestions. Plus, working conditions have proven unsafe for both the inspectors and the drivers alike.

In response to the presented problem we propose a feasible solution that takes the inspector out of the plenums, and puts them at the seat of their desks. The proposed solution is a mobile platform controllable from a host computer, running over WiFi. The proposed system addresses safety of inspectors and drivers alike, quality of inspections, overall costs, time per inspection, and reliability of results from beginning an inspection to the submission of the final report

Analysis

The next phase of our design includes implementing our plan for autonomy, integrating a battery pack, upgrading and enclosing the motors and electronics, and redesigning the chassis/lift structure for better rigidity and to meet necessary ingress protection concerns.

For system autonomy, we would employ a simple line following scheme to keep our robot on a specific path. On this line we would place RFID tags to indicate the location of various test points. These tags would be picked up by an RFID scanner integrated into the chassis and would store basic archival information on the test point. This scheme would be very robust since line followers are easy to make and we can guarantee test accuracy by placing the RFID tags along the line. This assures that images are taken from the same location every inspection, which yields the highest degree of accuracy with our software. Using the feedback from our motors and adding an IMU, we could improve upon the navigational precision even more. Finally, additional sensors like ultrasonic distance detectors and magnetic Hall sensors would be used in object avoidance schemes and to detect the steel used in the hangars. All this integrated with strong software and control structures, would a great degree of autonomy and free up inspector time or allow the inspector to control multiple units.

Integrating a battery pack and enclosing the electronics should be trivial tasks given funding or resources. We recommend a lightweight LiPo or NiCd battery pack, sufficient enough for four hours of continuous operation. The battery pack should be hot swappable and easy to change. Redesigning the chassis and lift system is more a task suited to mechanical designers and we don't have much input there aside from the IP concerns we identified in our visits to the plenums: dust, humidity, small puddles, and corrosives in the air/ground from exhaust/sea water. These concerns would likely need to be addressed in the enclosure design to ensure a long lifetime for the robot. To upgrade the electronics and motor system, we recommend more rugged, precision DC regulators and converters, higher power drive motors to account for extra weight from chassis and additional inspection equipment. We also

recommend moving to a higher powered motor controller, like the Sabertooth, to drive these more powerful motors. Additional sensors and test equipment would likely need to be integrated for MDOT's other inspection procedures.

We also noticed that the MDOT had no test or metric to monitor/track air quality or effectiveness of the plenum in completing its purpose. In the next phase, we'd integrate air quality and other environmental sensors to monitor the plenum as it exchanges exhaust air. This task is difficult and time consuming for a human inspector and would be costly to implement as a sensor network; however, it would be easy and cost effective as a part of our system. We could also integrate flame and fire sensors to provide an effective early warning system.

Design

Mechanical

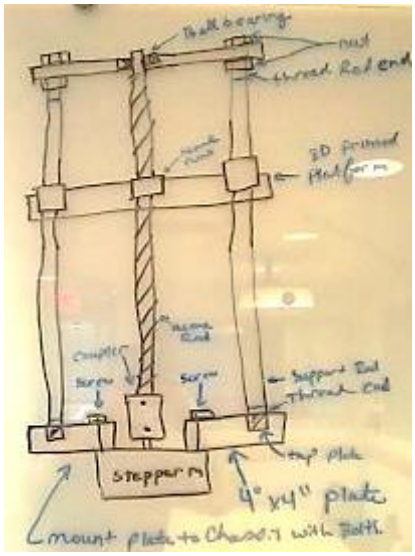


Figure 2 – Early white board design of lift system

Early designs of the structural elements of the robot were sketched out on a white board during group meetings, such as the one shown in figure 2. Once the initial design idea was flushed out we developed a 3D model of the robot component in Google SketchUp. Creating 3D parts with realistic dimensions was very important so that the 3D assembly of the robot could be used to identify potential problems, such as a bolt being slightly too long and hitting a spinning wheel. By creating a realistic 3D model we were able to drastically reduce the amount of time spent reworking the design once assembly started. Figure 3 shows the 3D assembly of the lift platform while figure 4 shows the 3D assembly of the robot chassis. Figure 5 shows the final assembly of the robot.

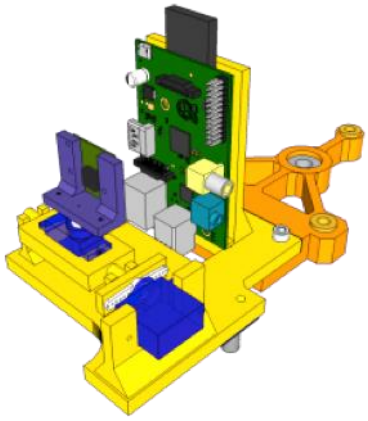


Figure 3 – Lift platform

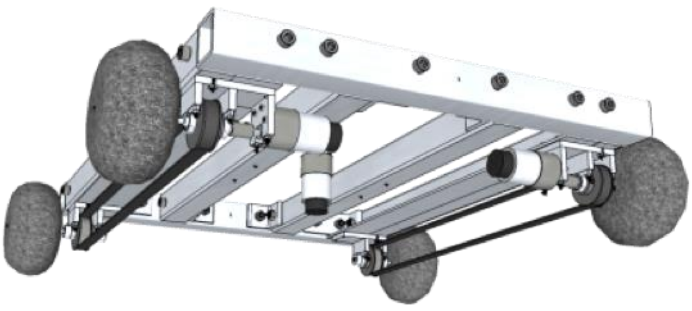


Figure 4 – Robot chassis

Figure 5 – Robot final assembly

In addition to a robust chassis, our solution also called for the inspection robot to be able to position its camera at varying heights in order to get the best view of whatever structural element it was inspecting. In order to accomplish this it was decided to develop a lift system. This lift system would consist of a tower, threaded acme rod, lift platform, and DC motor to spin the acme rod. While design this portion of the robot our team consulted with a graduate mechanical engineer Alex Irwin. Alex helped us develop the core idea behind the lift system, as well as picking out parts that would be needed. The lift tower which sits on top of the chassis can be seen in figure 5.

Simulations

After the initial design was completed our team set about creating a virtual Plenum (tunnel above the roadway in the Big Dig tunnel system) and virtual robot. This simulation environment would allow us to test our design and see how it would look from the robots point of view during an inspection. Figure 6 shows the virtual plenum and robot.

The 3D models of the plenum and the robot were both created in Google SketchUp, then with the help of a plugin called Sketchy Physics it was possible to allow a user to control the virtual inspection robot with an Xbox controller and perform a virtual inspection, all while obeying the laws of physics (i.e. if the robot hits an obstacle it will stop). Before moving on to the final design of the robot we performed many virtual inspections and learn a lot about how the robot should be driven, where the driving and inspection cameras should be placed to allow for the optimal field of view, and how to position the robot for each type of inspection.

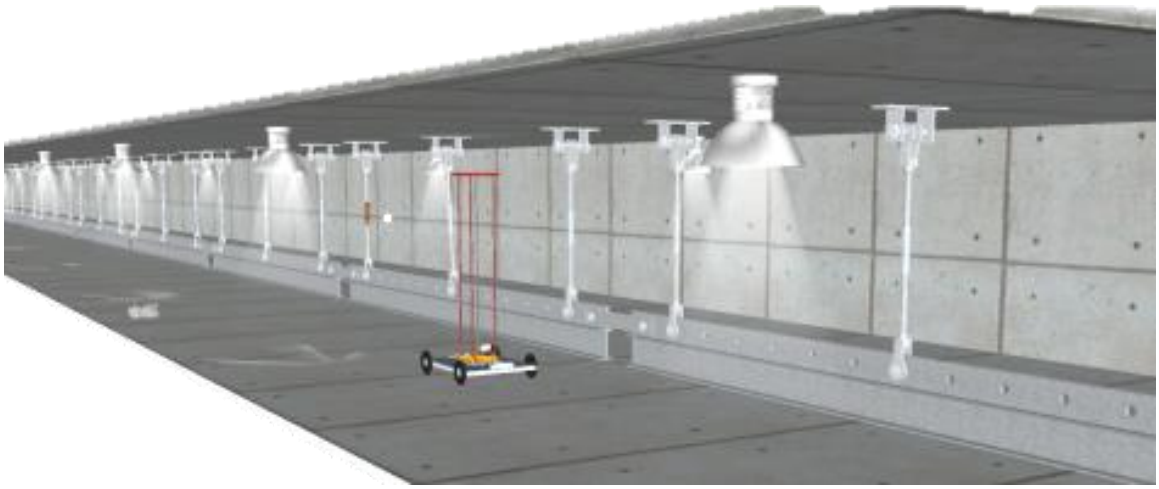


Figure 6 – 3D model of Upper Plenum with Simulation Robot

The 3D model of the plenum and the robot taught us a great deal about how to create a successful inspection robot, and we contribute the success of the mechanical platform, to our early simulation efforts.

Electrical

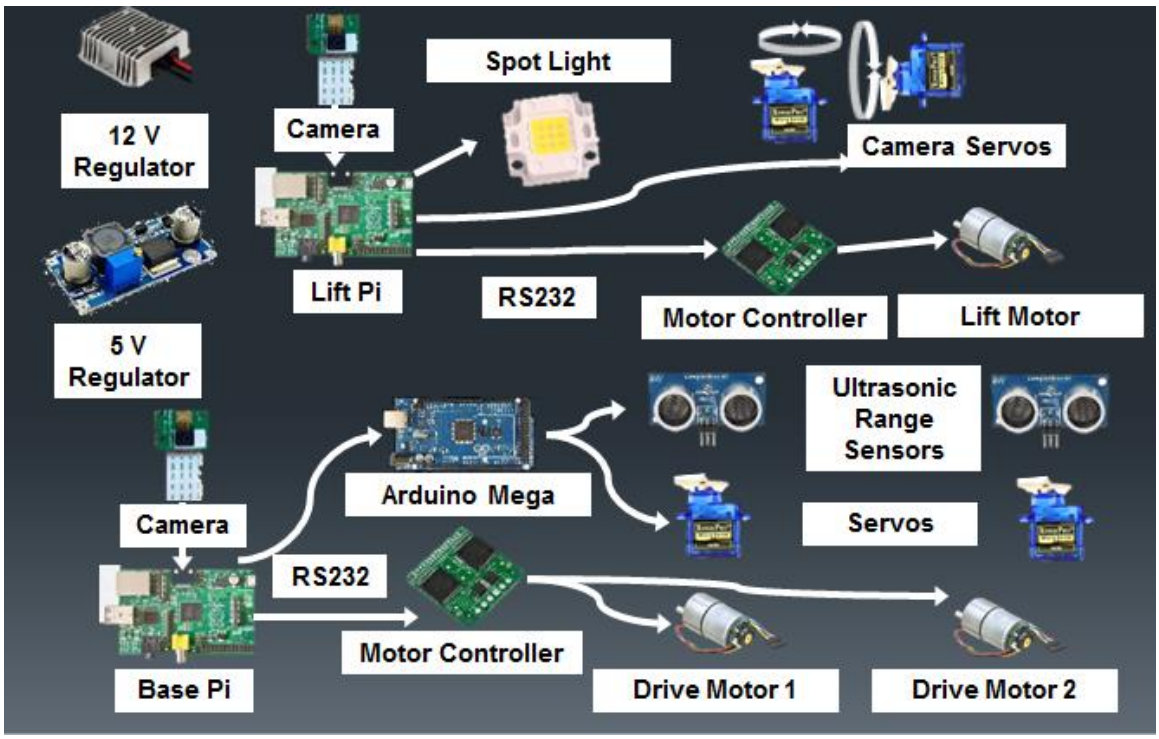


Figure 7- Basic System Overview

From a system level, the Raspberry Pi's each have charge of a motor controller and camera module. One Pi is designated the "Base" and the other we call the "Lift." Base and Lift receive commands over Wifi and relay those commands to their motor

controller or appropriate GPIO pin. Base needs to control both channels on the 15A Roboclaw and a Pi Camera Module; Lift needs to control a single channel on its motor controller, two servos for pan and tilting the camera, the Pi Camera Module, and a 5V signal to trigger the spotlight.

Also included on the system was an ArduinoMega2560, which we included to increase I/O capabilities, since we were limited to the number of GPIO pins provided by the Pi's. This controller also allows for easy integration of sensors and has ready-made boards for integration and development further down the line.

To power all this, we intended to use a 24V or 36V rechargeable LiPo or NiCd battery but could not due to funding limitations. This battery would then feed into our 12V DC regulator, which would be converted by the LM2596 step down regulators. For now, we provide power over a two-wire umbilical that feeds 24VDC to the 12VDC regulator.

Software

System Software and Communications

The system software and communications comprised of providing a means for the user to interface with the mobile platform over WiFi from a host computer. On the side of the host, a user interface with real-time media streamed video embedded within was a must. Why a must? Having a delayed video stream would put the user's perspective in the past of the current, which comes with the potential for one of many possible issues to arise (e.g. user drives into the wall, only realizing when the mobile robot has already been spinning its tires head onto concrete for the amount of time the video stream was delayed). In addition to this, a means of controlling the robot was a requirement on the side of the host. On the side of the mobile platform, a video camera for live media streaming is needed to provide a field of view from the robot's perspective. There is also the obvious need to be able to interface with the robot and all installed HW it carries.

Overview of System SW and Communications Design

In response to the requirements we had defined, finding the appropriate components, along with the appropriate methodology to use for its purpose, took a combination of research and price matching. The initial component list was the final list of components used, as far as the system SW and communications were concerned. A high-level view of the system is shown in Figure 8.

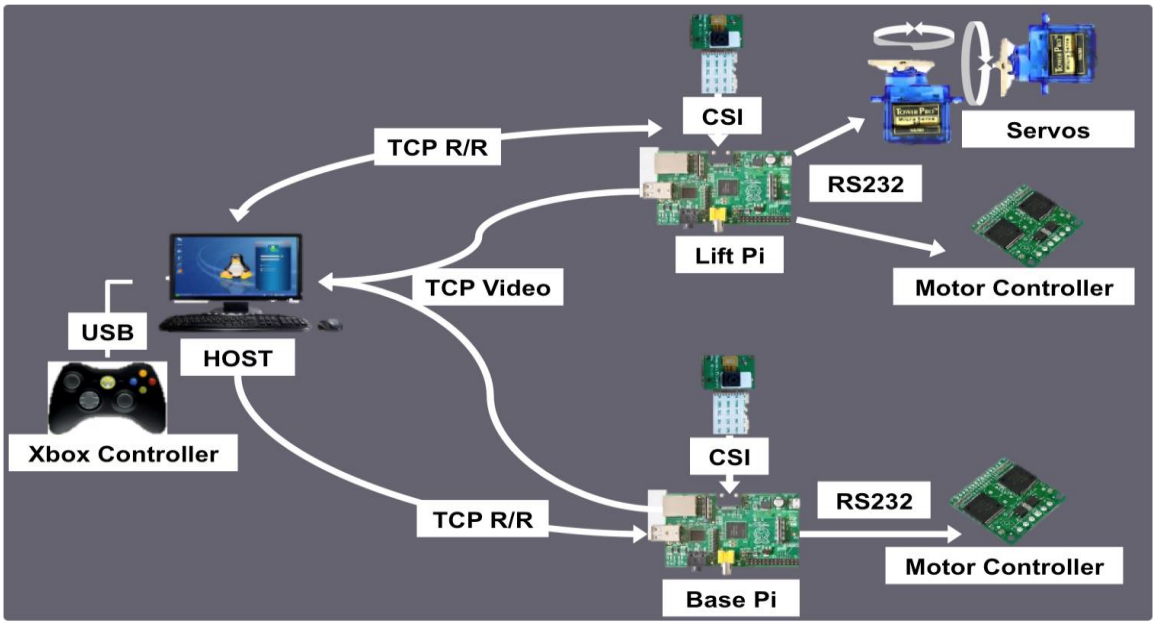


Figure 8 - Overview of System SW and Communication Scheme

The remaining subsections provide details on each aspect of the shown diagram.

Communications

Our design is a rendition of the client-server architecture model. The client-server model is comprised of two parts the client and the server, translating as our host and mobile platform respectively. It works by setting the client is responsible for sending requests to the server that then processes the request and responds to the client to confirm it has received. In addition to the confirmation, the server also sends back any data represented as that requested by the client. Figure 9 depicts the Client-Server Model as implemented for our system's communication scheme.

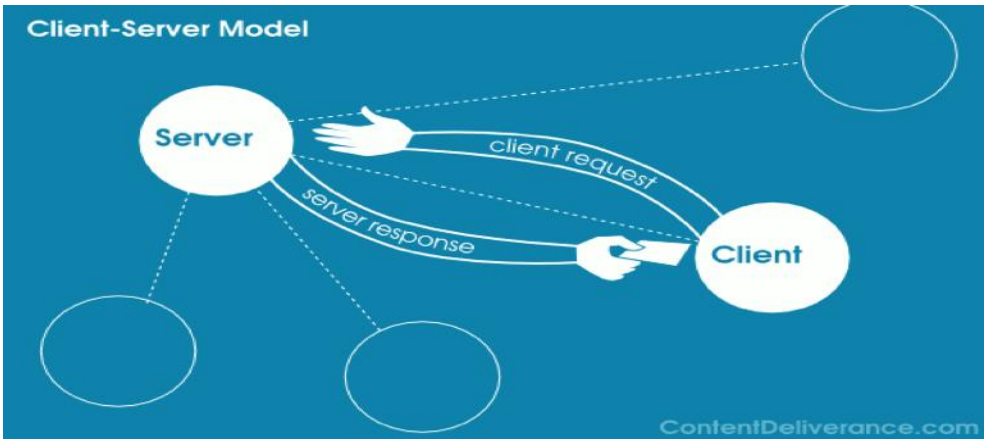


Figure 9 - Client-Server Architecture Model

In our system design, the robot consists of two Raspberry Pies (RPIs), each modeled as the servers. All communication and functionality of the RPIs are independently of one another, with no communication going between them. We defined one of the

RPIs as being the *Base Pi*, the other as the *Lift Pi*. The reason for this will be made clearer throughout the remaining contents of this section.
As hinted upon prior, the host computer needs to be able to control all aspects of the robot such as moving the base, lift system, camera system and any other features that could be added in the future. For this reason the host computer has the ability to talk to both of the servers on the robot. The request/response protocol is also constructed to handle the various request types and be extensible for any future features.

System Servers

The system has two servers, each on a separate RPI, one that is responsible for handling requests for the base system and one for the lift system. The reason for this is that both servers need to be responsive, and a single RPI would not have enough CPU power to handle all communications. Each server is designed to be as lightweight and efficient as possible. Consequently, the only real responsibility the servers have are to decode requests and then relay the data in the requests to their proper hardware components such as the motor controller or servers. The servers are not responsible for any processing of data, and are really just a go-between for the hardware components and the host computer.

The software system is architected to include a video streaming solution so the host could see what is happening on the robot and inspectors can carry out inspections.

Ideally both servers have video streams that the client can see what is happening in the tunnels from two different viewpoints. The video streams are built to be real-time with minimal lag, high quality, and easily integrated into the host GUI.

Interfacing

The Controller

It was imperative that we implemented an intuitive method for controlling the robot and all its functions. Controlling a mobile platform through the eyes of live media streaming carries a strong resemblance to video games. This factor, along with the fact Microsoft spent $100M on R&D for their Xbox controller [3]; we chose to go with just that.

As shown in Figure 10, the buttons on the Xbox controller are mapped to specific requests used to control the robot. As claimed in the figure, nearly all functionality of the system is accessible from the controller itself, i.e., all functions appropriately suited for the controller was incorporated. Key assignments were mapped out carefully, and tested/ modified excessively. This was to ensure controlling the robot was done in the most effective/ natural way for the user.

Figure 10 - XBox Controller Key Map

A visual of the robot is provided in Figure 10 to assistant on the following elaboration concerning controller mapping. In summary, controller was mapped as follows:

- The right joystick controlled the base motion of the platform guided by the *Base Pi* and its camera.
- The left joystick controlled rotational motions of the *Lift Pi* camera.
- Front triggers (2x) located at the top of the controller control the lift itself, the lift platform's vertical movement up/down mapped to right/left triggers, respectively.
- Middle button enables the controller, acting as an i/o switch.

- Buttons provide access to features such as taking a picture, record current field of view as a movie, set/get flags to move on to the next hanger, and launch our *Image Analyzer* to further inspect the most recent of pictures captured.

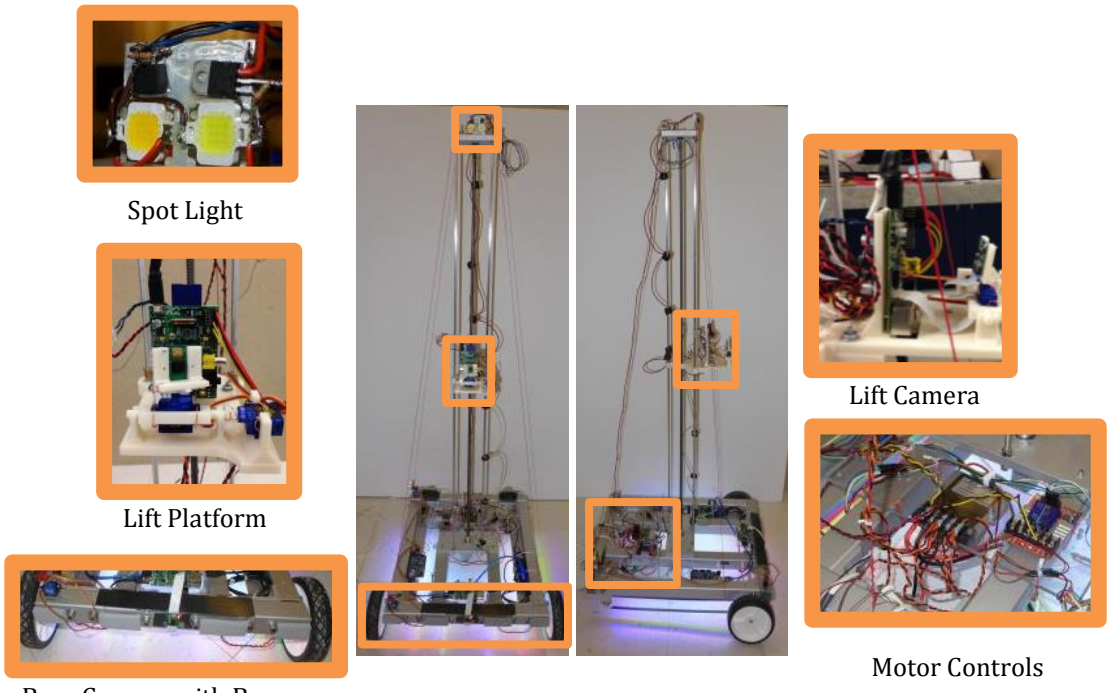


Figure 11 - System Level View

The User Interface

The graphical user interface (GUI) offered with our system software package was designed to allow the user complete control over all features and settings, while maintaining simplicity for the purpose of ease-of-use. Figure 12 shows the main window.

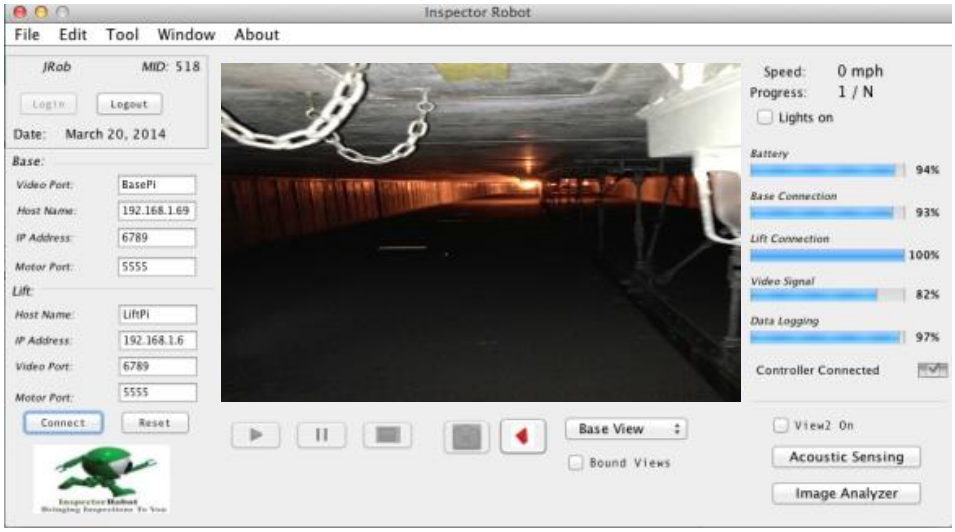


Figure 12 - Main Interface (Base Pi Camera)

Further elaboration on the system SW and communication scheme are providing in the following section [see Parts and Implementation].

Parts and Implementation

Mechanical

When designing the robot we knew that we wanted a very strong, yet light chassis, which would withstand the dirty tunnel environment and resist corrosion. Since corrosion is a big problem in the tunnels since it is often humid and damp we chose aluminum 80/20 extruded square tubing. This tubing is corrosion resistant, strong, and easy to rework in case future sensor packages are introduced. Aluminum was chosen for as many of the robots elements as possible in order to reduce weight and protect against corrosion. In the even that aluminum was not available then anodized steel, or stainless steel was utilized.

Figure 13

The drive system of the robot is a differential two-wheel front drive system with a caster in the back. The motors are in the back of the robot, but are connected to the front wheels via a timing belt and timing gears. Initially the robot was intended to have a

Figure 14

four wheel differential drive system, but the wheels that were initially chosen where too rubbery (figure 13) for that application, therefore we substituted two plastic lawn mower wheels (figure 14) and a caster in their place. This system worked very well, and allows for great maneuverability and is very stable.

Figure 15

To drive the robot we chose two 50:1 Metal Gearmotors with 64 CPR Encoder from Pololu, figure 15. These motors each can produce 12 kg-cm of torque, our tests showed that this is more than adequate to propel the robot up any gradient in the Big Dig tunnel system with a top speed of 30.5 meters/minute. The encoders on these motors allow us to precisely control the speed of the robot, as well as the ability to perform autonomous movements in the future.

To drive the lift system we chose one 19:1 Metal Gearmotor with 64 CPR Encoder from Pololu. Our tests revealed that at least 4 kg-cm of torque was needed in order to drive the lift acme rod and propel the lift platform upward. This motor is capable of producing 5 kg-cm of torque.

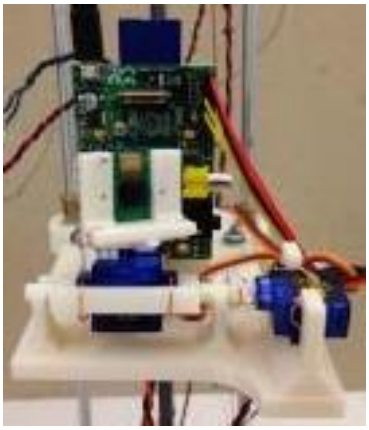


Figure 16 – Lift platform

Everything was assembled using either bolts/nuts with lock-tight, or couplers with lock-tight. Epoxy was also utilized if neither bolts or couplers were appropriate.

The lift platform (figure 16) was designed in Google SketchUp and printed at the NEU 3D printing studio. The lift platform has a mount for a raspberry-pi, DC-DC regulator, two micro servos, and a raspberry-pi camera. The servos control the motion of the raspberry-pi camera and allow for a complete control of the camera in the x and y directions for 180 degrees.

Electrical

Raspberry Pi Model B Rev. 2

We utilized dual Raspberry Pi's in our design as our embedded devices. We found them to cheap and easy to use, yet powerful enough to meet our video streaming and controller needs. Power to this device was delivered through a LM2596 StepDown DC Converter set to 5V.

- SoC = Broadcom BCM2835
- CPU = 700 MHz ARM 1176JZF-S
- GPU = Broadcom VideoCore IV supporting OpenGL, MPEG-2, 1080P, h.264/MPEG-4 AVC
- RAM = 512 MB RAM
- USB = 2 - USB 2.0 ports
- Video outputs = Composite RCA and HDMI
- Audio outputs = 3.5mm and HDMI
- Onboard storage = SD/MMC/SDIO card slot (4GB recommended)
- Onboard network = 10/100 RJ-45
- Low-level peripherals = 8 × GPIO, UART, I^2C bus, SPI bus with two chip selects, +3.3 V, +5 V, ground
- Power rating = 700 mA (3.5 W)
- Power source = 5 volt via MicroUSB or GPIO header for power (1.0A recommended)
- Size = 85.60mm × 53.98mm (3.370 × 2.125 in)
- Weight = 45g (1.6oz)

Figure 17 - Raspberry Pi Model B Rev. 2

ArduinoMega 2560

We included an Arduino microcontroller to increase our I/O capabilities, since we were limited by the small number of GPIO pins provided by our RaspberryPi's. The Arduino added many extra digital and analog I/O pins for us to integrate extra sensors and supplementary electronics, and was a device familiar to most of us. Power to this device was delivered through a LM2596 StepDown DC Converter set to 5V.

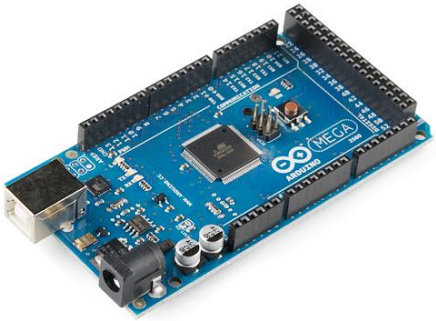


Figure 18 - ArduinoMega 2560

- Processor: ATmega2560 @ 16 MHz
- RAM size: 8 Kbytes
- Program memory size: 248 Kbytes
- Motor channels: 0
- User I/O lines: 7
- Max current on a single I/O: 40 mA
- Minimum operating voltage: 7 V
- Maximum output voltage: 12 V

Roboclaw 15A

This RoboClaw motor controller was connected to the Base Pi using simple serial and controlled the drive motors. Power to this device was delivered through the 12V DC regulator.

Figure 19 – Roboclaw 15A

- Motor channels:2
- Control interface:non-inverted TTL serial (2-way);
  RC servo pulses; analog voltage
- Minimum operating voltage:6 V
- Maximum operating voltage:34 V
- Continuous output current per channel:15 A

RoboClaw 5A

This RoboClaw motor controller was connected to the Lift Pi using simple serial and controlled the lift motor which advanced the Lift System Payload up or down depending on the direction of the motor. Power to this device was delivered through the 12V DC regulator.

Figure 20 - RoboClaw 5A

- Motor channels:2
- Control interface:non-inverted TTL serial (2-way);
  RC servo pulses; analog voltage
- Minimum operating voltage:6 V
- Maximum operating voltage:34 V
- Continuous output current per channel:5 A
- Peak output current per channel:10 A

Raspberry Pi Camera Module

We used two of these camera modules to capture live video for the operator and pictures for our software.

Figure 21 - Raspberry Pi Camera Module

- 5 megapixel native resolution sensor-capable of 2592 x 1944 pixel static images
- Supports 1080p30, 720p60 and 640x480p60/90 video
- Camera is supported in the latest version of Raspbian, Raspberry Pi's preferred operating system

Micro Servos

We utilized four of these Micro servos on our platform. Two were driven using GPIO pins on Lift Pi and the software library ServoBlaster; the other two were located on the base and were driven by the Arduino. The lift servos were used to pan and tilt the lift camera and the base servos rotated ultrasonic sensors across a 180 degree sweep. Power to these devices was delivered through their controllers.

- Coreless motor
- Dead Band Width : 2 usec
- Stall Torque : 1.5kg/cm at 4.8V
- Operation Voltage : 3.0 - 7.2Volts
- Dimension : 22mm x 12mm x 29mm

Figure 22 – micro servo

Edimax Wireless USB Adapter

- Currently smallest wireless adapter to be hidden well in USB port
- Supports 150 Mbps 802.11n (Up to 6 times the speed and 3 times the coverage of 802.11b.).
- Channels (FCC) 2.4GHz : 1~11. Power Input USB Port (Self-Powered). Dimensions 0.28" x 0.59" x 0.73". Temperature 0 -40 degree C (32-104 degree F). Humidity 10 ~ 90% Non-Condensing. System XP/Vista/Win7, Mac, Linux
- Port 1 x 2.0 USB Type A. Wireless Data Rates Up to 150 Mbps. Modulation OFDM: BPSK, QPSK, 16-QAM, 64-QAM, DSSS. Frequency Band 2.4GHz - 2.4835GHz. Antenna internal chip antenna
- Spec Standards IEEE 802.11n; backward compatible with 802.11b/g Wi-Fi Certified. Security 64/128 bit WEP Encryption and WPA-PSK, WPA2-PSK security; WPS compatible IEEE 802.1X

12V DC Regulator

This was the main power regulation device on the platform and it was intended to regulate power coming off the battery for all devices. Though this device claimed over/under voltage protection, two of these devices ended up breaking when being underdriven.

- Input voltage: 24V DC
- Input range: 12-40V DC
- Output voltage: 12V DC
- Output current: 20A (Max) / 240W
- Case material: die-cast aluminum
- Potting material: epoxy sealed

Figure 23 - 12V DC Regulator

LM2596 Step-Down Converter

Figure 24 - LM2596 StepDown Converter

We used these devices as a quick and cheap way to set up different voltage rails and simply our platform's wiring scheme.

- Input: DC 3V to 40V (input voltage must be higher than the output voltage to 1.5v above can not boost)
- Output: DC 1.5V to 35V voltage continuously adjustable, high-efficiency maximum output current of 3A.

In addition to main electronics listed above, we also created a few small circuits like the camera spotlight (figure 25), roadway lights, and speakers. The lighting circuits used LM317s in constant current configuration to regulate current to the LEDS and a simple MOSFET based logic switches to turn on/off. The speaker circuits use LM358s as dual stage audio amplifiers to boost the 8bit tones sent from the Arduino.

Figure 25 – Spot light

Software

System Software and Communications

To implement the client design we chose to use the JAVA language. By using JAVA, we were able to develop the software package that is portable in terms of host computer platforms. The host communicates to the servers via TCP. We chose TCP for it is a reliable protocol with transfer rates suited for real-time control.

Communications

To do the actual requests/responses we used the Google Protobuf library [4]. This library allows for the encoding/decoding of structured data for transmission over a network, as needed. The message structure can be fine tuned by the developer and is able to be built upon, allowing for new request types for future features. The Protobuf library has API for both Java and C++. The only viable options to be used as the host used Java and the servers used C++.

As the goal of the server architecture was to be fast, efficient and lightweight it is built using the C++ Boost ASIO library. The Boost ASIO is an asynchronous I/O architecture that is intended to for use in managing long running operations with primary focus on networking. Boost ASIO provides support for asynchronous operations using the Proactor design pattern [see Figure 26].

The general flow of the servers is that they first listen for a connection from the host. They set up a new TCP session for each incoming connection. Then when the servers received requests they use Protobuf to decode them. Each request has a header that specifies which request type it is and the server uses this type to route the data in the request to hardware components. The hardware components are all connected to the RPIs using various protocols such as serial RS232. The server knows how to send data to each peripheral using whichever protocol is needed.

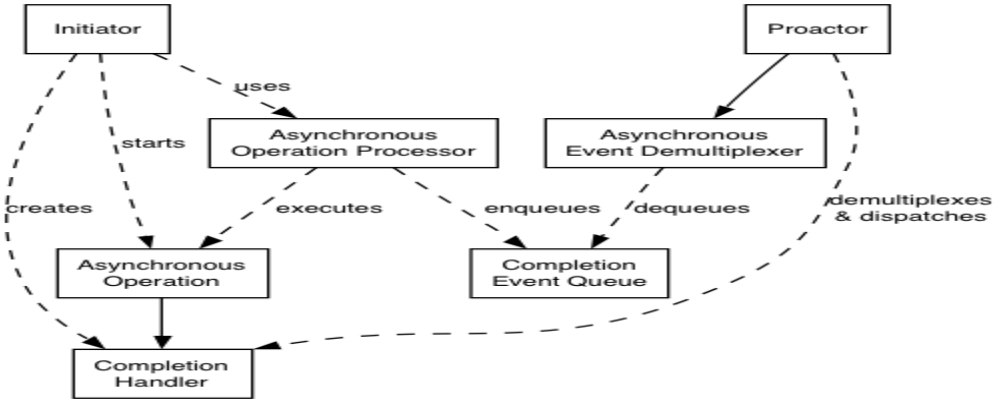


Figure 26 - Proactor Design Pattern

Real-time Media Streaming

In order to realize our design goals of the video streaming solution we used the Gstreamer library to do the heavy lifting [5]. The Gstreamer library is used because it has bindings for both C++ and Java, and could be used on both the client and server. The Gstreamer library takes care of encoding, packetizing, and broadcasting the video stream from both servers. In order to achieve the fastest and most stable video stream we configured Gstreamer to use the RTSP protocol over a TCP connection. On the client Gstreamer receives, de-packets, decodes the video streams from the servers. The library even has video playing components that made it easy to integrate into the client GUI.

Interfacing

The Controller

The JInput library was used to link the Xbox controller to the host computer [6]. The JInput library polled the Xbox controller every few milliseconds and told the host what has changed in the Xbox controller state. An example of this would be if a button was clicked or a joystick is moved. The host then created requests to send to the server based on the state changes of the Xbox controller. Key assignments were mapped to provide an intuitive interface at the hands of the inspector.

The User Interface

All GUIs were developed using the JAVA Swing API. This API provides containers and components to physically interact with that are equipped with EventListeners that respond to an event triggered from within the GUI (e.g. in the event a button is clicked and the response the application should hereby). Figure 27 shows all involved interfaces.

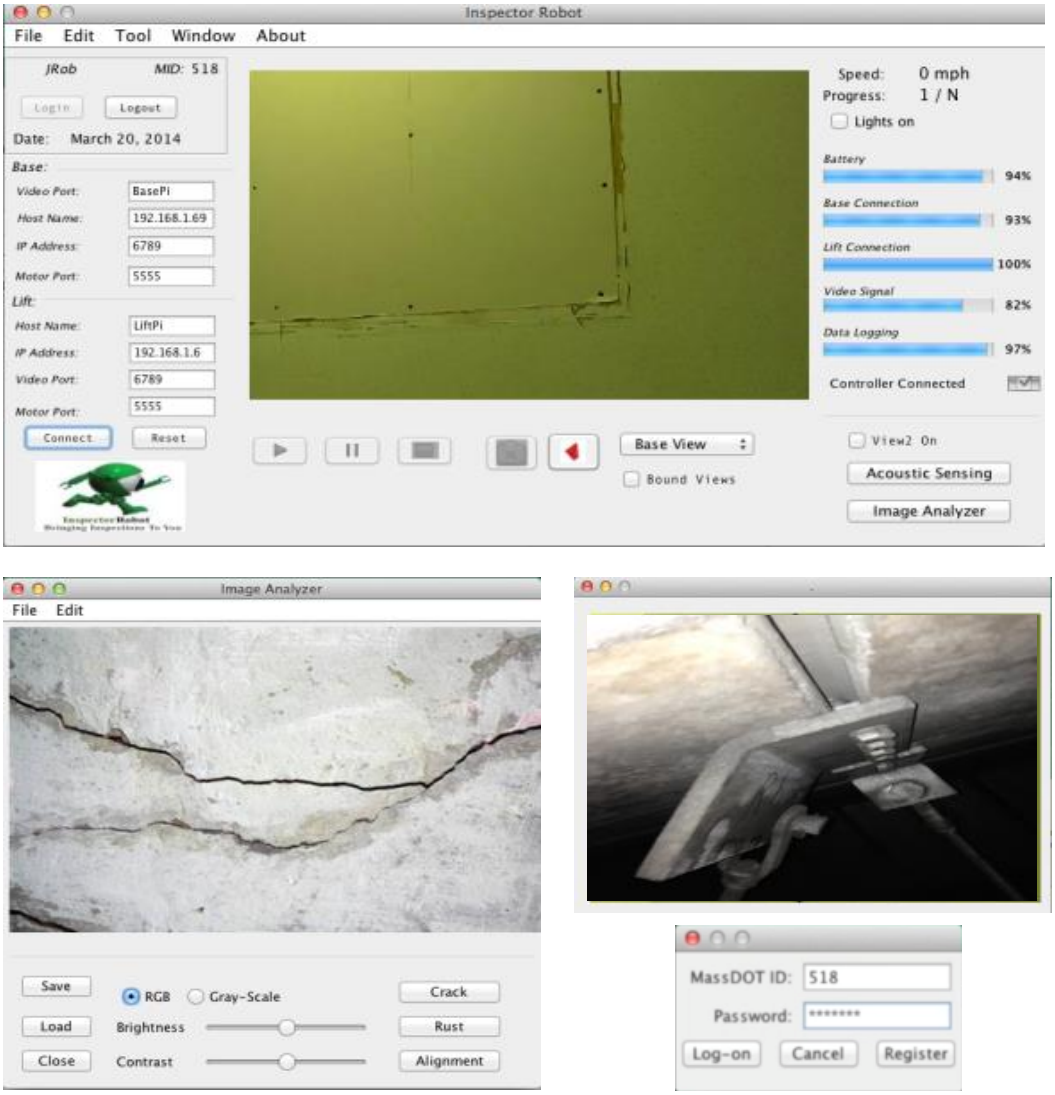


Figure 27 - User interfaces offered as apart of the SW package. The main window provides control of all SW functions/ features, with video stream from Base Pi embedded in the center (Top). Image Analyzer allows for image inspection and analysis by offering access to image processing and computer vision functions (Bottom Left); A second embedded video provides live stream of rotating lift camera to allow for a close-up view, and also a login window for

As shown in the top of Figure 27, the user has complete control over network-related settings, making for the simplest method to connect the user to the remote

mobile platform over Wi-Fi. The panel controlling the connection [network] settings is located on the left side of the main window.

The main window also provides real-time system status updates. The system status updates are visually provided by the status bars on the right side of the GUI, covering anything from *Battery* life, to the *Connection* status itself.
In case it not evident, the live media stream plays in the middle of the main window, with various controls pertaining to the feed accessible via the buttons below.

The ability to login as a user was implemented as a means of tracking the inspectors performing a given inspection. In addition, it is assumed managers would take interest in inspectors progress, and there is no better way than to log it digitally. Notice the login window shown bottom right of the figure.

Notice, the two buttons located bottom right of the main window, *Acoustic Sensing* and *Image Analyzer*. The former was added for future functionality to be added, the latter opens the *Image Analyzer* application window included as a part of the SW package provided with the system.

The *Image Analyzer* was designed to provide inspectors image processing capabilities. By default, the application opens displaying the most recent image captured during the inspection; however, the user can easily open an image from the past via an Open Dialog search window. This interface is where the all post-processing is performed on the images (e.g. *Crack Detection*, brightness/contrast adjustment, image type conversion, amongst a few other capabilities). This application was designed such that additional functionalities are easily added in the future.

The last of the window views provided allows the user to view both the *Base Pi* and the *Lift Pi*, simultaneously. Initially, we provided a toggle button in the main window for the user to switch between views as desired. However, after testing the system it was realized that it would be much more efficient to have both video streams viewed at the same time, both in terms of *look and feel* and time-efficiency. With that being said, a window was developed to play the video stream from the *Lift Pi*, in parallel to the main window playing the stream from the *Base Pi*. The user has the ability to toggle the *Lift Pi* I/O, as it is not always needed during the inspection process.

Computer Vision

Hanger Alignment Estimation

Hangers are ideally plum. A plum hanger has the minimum amount tension on it caused by a load being pulled down by gravity. The tension on a hanger that is not plum increases as the angle, theta, it is leaning increase, as per the following equation:

$$T_{hanger} = T_{gravity} * \sqrt{1 + \tan^2 \theta} \quad (1.1)$$

A computer vision technique using stereo vision is use to estimate the angle that a hanger is leaning at. Two images are taken by the system as it is passing a hanger. The system is two meters from the row of hangers. Image one is taken 45 degrees before the hanger, and image two 45 degrees after, as shown in figure 28. The X and Y directions correspond to the width, and length of the tunnel, respectively.

The algorithm is divided into four steps; image preprocessing, image segmentation, Hough transformation, and Z rotation.

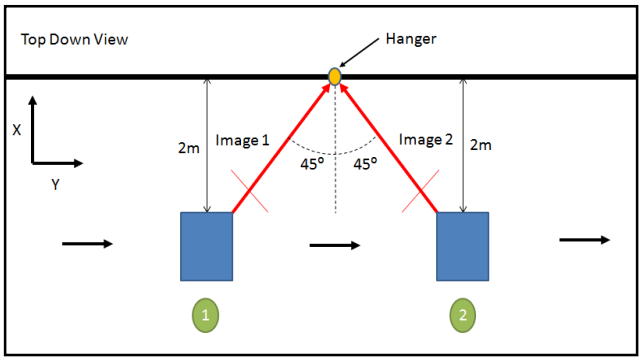


Figure 28 - Top down view of positions image 1 and image 2 of the hanger are taken from by the robotic system

Image Preprocessing

The images taken of the hanger using the Raspberry Pi camera are quite noisy. In addition to that, elements in the background, such as joints between concrete panels, and the T beam appear as straight lines, as does the hanger. A series of preprocessing steps are used to make the hanger stand out from the background elements.

Color to Gray Scale Conversion

Images taken by the Raspberry Pi camera are in color. The following equation is used to convert the colored image, I(R,G,B), to a gray scale image, g(x,y) [6]. R, G, and B refer to the color channels of the colored image. This conversion reduces the number of pixels operations are done on to a third, without loss of information about the location of the hanger.

$$g(x,y) = 0.2989\, I(R(x,y)) + 0.5870 * I(G(x,y)) + 0.1140 * I(B(x,y)) \quad (1.2)$$

Subtractive Preprocessing

The pixel values of a line like structure tend to stand out in comparison to the background in the immediate area. Using a median filter on the image of a carefully selected kernel size can therefore remove lines that pass through a small window that are of a particular width without blurring lines that are larger. When the width of a line in a window causes it to take up more than half the window, it will remain. Conversely, if the width of a line in the window causes it to take up less than half of the window, it will be removed.

The test hanger was approximately 2.8 meters from the camera, which resulted in the test hanger appearing as 5 pixels in width, H. The median filtered image, m(x,y), of g(x,y) with a kernel size H results in an image where lines 2 pixels in width or less are removed, and the general camera noise and lighting and color gradients are somewhat blurred. The median filtered image, M(x,y), of g(x,y) with a kernel size W such that a line of width H is removed results in an image where the general camera noise and lighting and color gradients are very blurred. W three times the size of H is the minimum that will work and will remain odd valued as long as H is odd valued.

$$W = 3 * H \quad (1.3)$$

$$m(x,y) = medianFilter(g(x,y),[H,H]) \quad (1.4)$$

$$M(x,y) = medianFilter(g(x,y),[W,W]) \quad (1.5)$$

Subtraction of the two median filtered images results in an image, L(x,y), where the background is removed, leaving only structures that are of a width A, where:

$$H \leq A < W/2 \quad (1.6)$$

$$L(x,y) = M(x,y) - m(x,y) \quad (1.7)$$

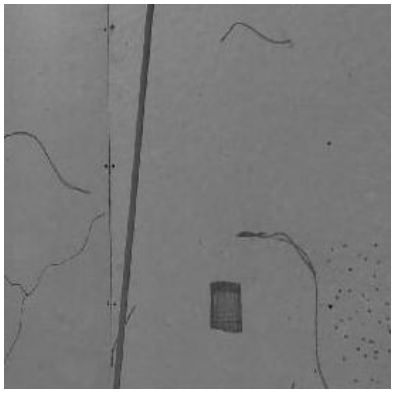


Figure 29 - G(x,y)

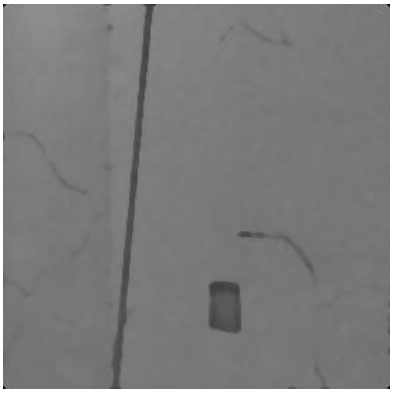


Figure 30 - m(x,y)

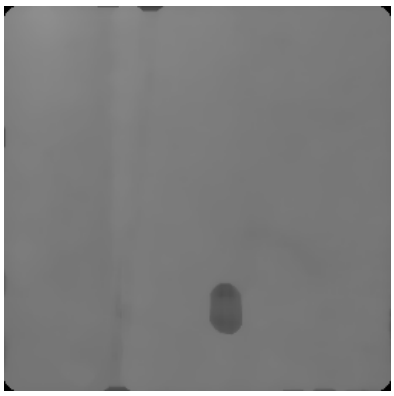


Figure 32 - M(x,y)

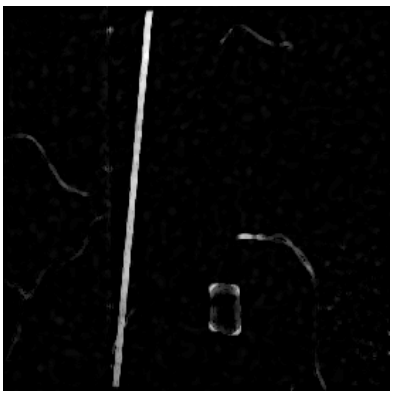


Figure 31 - L(x,y)

Image Segmentation

The image is converted to a binary image, B(x,y) before the Hough transform is used. The threshold value was selected using Otsu's method [7]. The Otsu's method algorithm minimizes the intra-class variance between the regions above and below the threshold. For the set of test images used, the optimal value for T was 12.

$$B(x,y) = \begin{cases} 1, L(x,y) \geq T \\ 0, L(x,y) < T \end{cases} \quad (1.8)$$

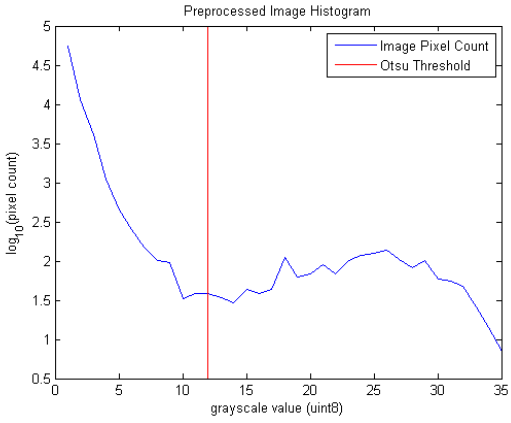


Figure 33 - Histogram of L(x,y) and threshold

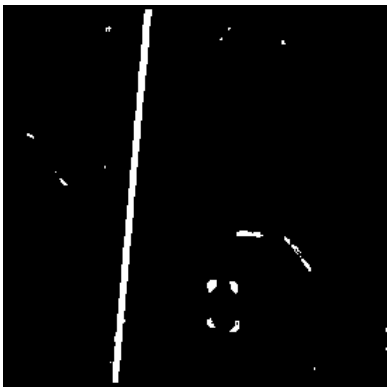


Figure 34 - B(x,y) created with selected using Otsu's Method. T = 12.

Image Hough Transform

The preprocessing technique used results in an image where the hanger is the longest, continuous line in B(x,y). The Hough transform [3] of the image is to find the angle of the hanger, relative to vertical. This is done to an accuracy of half a degree, which yields and error of ± 0.25 degrees. The maximum value of HT(θ, ρ), corresponds to the hanger angle of the hanger relative to vertical, and the length of it, as seen in the image.

$$HT(\theta, \rho) = hough\ transform\big(B(x, y)\big) \tag{1.9}$$

$$-180\ degrees\ \leq\ \theta_n \leq 180\ degrees \tag{1.10}$$

$$\theta_n = n * 0.5\ degrees \quad ,\quad -360\ \leq n\ \leq 360 \tag{1.11}$$

$$\theta_{error} = \pm 0.25\ degrees \tag{1.12}$$

Figure 35 - HT(θ, ρ) of B(x,y) from figure 35. Maximum response at θ = 7 degrees

Axis Rotation

The angles calculated for each image are in reference to image planes that are rotated by γ, 45 degrees, around the vertical axis relative to the length and width of the tunnel. A Euclidean transformation is used to calculate the angles in reference to the length and width of the tunnel.

$$R_Z = \begin{bmatrix} \cos\gamma & -\sin\gamma \\ \sin\gamma & \cos\gamma \end{bmatrix} \tag{1.13}$$

$$\begin{bmatrix} \tan\theta_{width} \\ \tan\theta_{length} \end{bmatrix} = R_Z * \begin{bmatrix} \tan\theta_{image\ 1} \\ \tan\theta_{image\ 2} \end{bmatrix} \tag{1.14}$$

Error Estimation

The error propagation formula for the tunnel width and length angles can be calculated using the following two formulas:

$$\delta\theta_{width} = \frac{\delta\theta_{image\,1}\frac{\cos\gamma}{\cos\theta_{image\,1}{}^2} - \delta\gamma\sin\gamma\tan\theta_{image\,1} + \delta\gamma\cos\gamma\tan\theta_{image\,2} + \delta\theta_{image\,2}\frac{\sin\gamma}{\cos\theta_{image\,2}{}^2}}{1 + \left(\cos\gamma\tan\theta_{image\,1} - \sin\gamma\tan\theta_{image\,2}\right)^2} \quad (1.15)$$

$$\delta\theta_{length} = \frac{\delta\theta_{image\,1}\frac{\cos\gamma}{\cos\theta_{image\,1}{}^2} + \delta\gamma\cos\gamma\tan\theta_{image\,1} - \delta\gamma\sin\gamma\tan\theta_{image\,2} + \delta\theta_{image\,2}\frac{\sin\gamma}{\cos\theta_{image\,2}{}^2}}{1 + \left(\sin\gamma\tan\theta_{image\,1} + \cos\gamma\tan\theta_{image\,2}\right)^2} \quad (1.16)$$

Where the error in γ results from the error in the position of the system relative to the tunnel width. The camera cannot rotate about the axis perpendicular to the image, so it is assumed this error is zero and does not change. The error in the image angle come from the resolution of the Hough transform calculation.

Algorithm Pseudo Code

A summary of the algorithm written in pseudo code follows:

Hanger Alignment Estimation Algorithm

for both RGB PNG Images do
    I(r,g,b) = load RGB PNG Image
    g(x,y) = convert I(r,g,b) to grayscale image
    m(x,y) = median filter g(x,y) with the kernel size [H,H]
    M(x,y) = median filter g(x,y) with the kernel size [W,W]
    L(x,y) = M(x,y) – m(x,y)
    B(x,y) = threshold L(x,y) at T
    [HT(theta, rho), theta, rho] = Hough transform of B(x,y) with angle step width of A
    [maxThetaIndex] = Theta index of maximum in hough response
    maxTheta = value of theta at max index theta(maxThetaIndex)
end for
return [thetaWidth, thetaLength] = Z rotation of angles from each image

H = 5, W = 15, A = 0.5 degrees, T = 12

Crack Detection

Cracking and crack growth rate of the structural and non structural concrete of the upper plenum roof slab is monitored by contract inspectors. The presence and growth rate of cracks that form around the hanger anchor points of are of particular interest. A growing crack compromises the structural integrity of the anchor connection to the roof slab. Inspection is done visually and manually sketching cracks that are found and noting their location. During each inspection cycle, sketches are compared to assess the growth rate of cracks, and appearance of new ones. This method of sketching and comparison is subjective to person performing the inspection. Several computer vision based methods were developed in the past

that aimed to automate this task [8, 10, 11]. The proposed algorithm improves the true and false positive detection rates by combining elements from previously developed algorithms.

Image Preprocessing

The images taken of the hanger using the Raspberry Pi camera are quite noisy. In addition to that, elements in the background, such as joints between concrete panels, the hangers, and the T beam appear as lines, as do surface cracks on the concrete. A series of preprocessing steps are used to make the surface cracks stand out from the background elements.

Color to Gray Scale Conversion

Images taken by the Raspberry Pi camera are in color. The following equation is used to convert the colored image, I(R,G,B), to a gray scale image, g(x,y) [6]. R, G, and B refer to the color channels of the colored image. This conversion reduces the number of pixels operations are done on to a third, without loss of information about the location of the hanger.

$$g(x,y) = 0.2989\, I(R(x,y)) + 0.5870 * I(G(x,y)) + 0.1140 * I(B(x,y)) \quad (2.1)$$

Subtractive Preprocessing

The pixel values of a line like structure tend to stand out in comparison to the background in the immediate area. Using a median filter on the image of a carefully selected kernel size can therefore remove lines that pass through a small window that are of a particular width without blurring lines that are larger. When the width of a line in a window causes it to take up more than half the window, it will remain. Conversely, if the width of a line in the window causes it to take up less than half of the window, it will be removed.

Cracks tend to be thin lines, so by setting the kernel size, H, such that all cracks are removed when the median filter is applied, m(x,y), and then subtracting the gray scale image, g(x,y), from it, the result is an image, s(x,y), in which the background elements are removed, leaving just the cracks. This subtractive preprocessing method comes from the approach in [14]

A kernel size, H, of 11 was used. This will result in lines greater than 5 pixels in width being removed.

$$m(x,y) = medianFilter(g(x,y),[H,H]) \quad (2.2)$$

$$s(x,y) = m(x,y) - g(x,y) \quad (2.3)$$

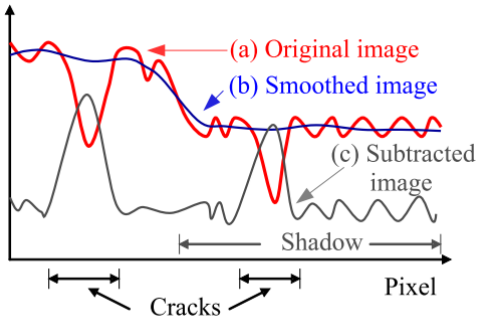


Figure 36 - Removal of slight variation like an irregularly illuminated condition, shading, or blemish. [1]

Figure 38 - g(x,y) image of cracked Concrete

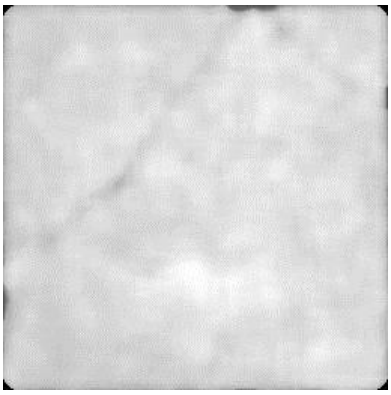


Figure 37 - m(x,y) median s(x,y) shadow image

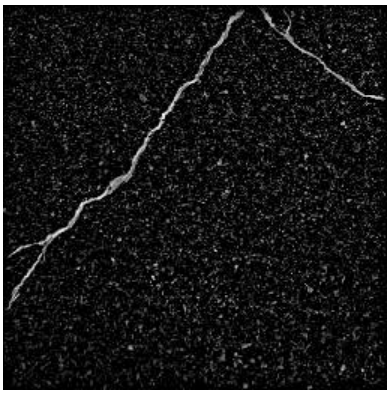


Figure 39 - filtered image

Fujita Line Emphasis Filter

The Fujita line emphasis filter [7] was selected as the most accurate technique for emphasizing cracks of varying width. Three kernels are constructed from the convolution of the Hessian matrix (2.4) with a Gaussian function (2.3)

$$\nabla^2 I(x,y) = \begin{bmatrix} I_{xx}(x,y) & I_{xy}(x,y) \\ I_{yx}(x,y) & I_{yy}(x,y) \end{bmatrix} \quad (2.4)$$

$$G(x,y;\ \sigma_n) = \frac{1}{2\pi\sigma_f} e^{-\left(\frac{x^2+y^2}{\sigma_n^2}\right)} \quad (2.5)$$

The three kernels, (2.7, 2.8, and 2.9) are the second order partial derivatives of the Gaussian function of size HxH. Where σ_n is the standard deviation of the Gaussian. These kernels are calculated for N scale factors, which follows equation (2.6).

$$\sigma_n = \sigma_1 s^{n-1}, n = 1, 2, \ldots N \quad (2.6)$$

$$\sigma_1 = s = \sqrt{2}, N = 4, H = 21$$

The values for s, σ_1, H, and N were determined experimentally in [1].

$$G_{xx}(x,y) = G(x,y;\ \sigma_n) * I_{xx}(x,y) \quad (2.7)$$

$$G_{yy}(x,y) = G(x,y;\ \sigma_n) * I_{yy}(x,y) \quad (2.8)$$

$$G_{yx}(x,y) = G_{xy}(x,y;\ \sigma_n) = G(x,y;\ \sigma_n) * I_{xy}(x,y) \quad (2.9)$$

At each coordinate of this 2x2 matrix, the eigen values are calculated from the values given by equations 2.7-9, Where $\lambda_1(x,y)$ and $\lambda_2(x,y)$ are the eigen values at a particular coordinate in I(x,y), and $\lambda_1(x,y) > \lambda_2(x,y)$. A particular pixel in I(x,y) being part of a line is given by equation (2.10). This is calculated for N scale factors to take into account that cracks may be of various widths. The line response of the image is given by equation (2.11), which is the maximum of normalized response from the set of scale factors. It was determined experimentally in [1] that α = 0.25. R(x,y) is shown in figure 41.

$$\lambda_{12}(x,y) = \begin{cases} |\lambda_2(x,y)| + \lambda_1(x,y) & , \quad \lambda_2(x,y) \le \lambda_1(x,y) \le 0 \\ |\lambda_2(x,y)| - \alpha\lambda_1(x,y) & , \quad \lambda_2(x,y) < 0 < \lambda_1(x,y) < \frac{|\lambda_2(x,y)|}{\alpha} \\ 0 & , \quad otherwise \end{cases} \quad (2.10)$$

$$R(x,y) = \max_{\sigma_n} \sigma_n^2 \lambda_{12}(x,y;\sigma_n) \quad (2.11)$$

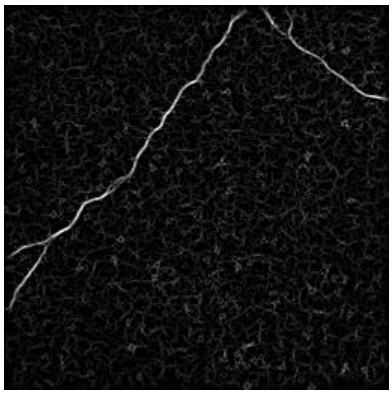


Figure 40 - R(x,y) line response

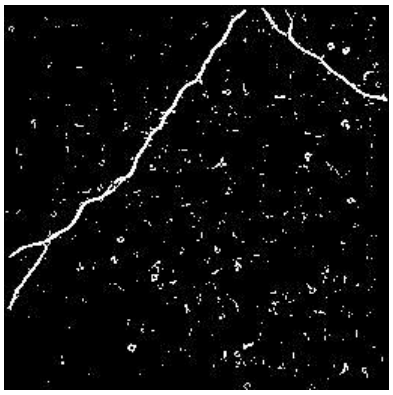


Figure 41 - B(x,y) binary segmented image

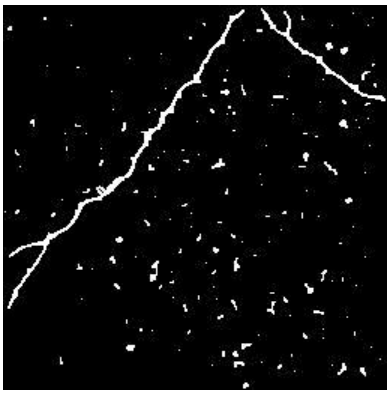


Figure 42 - L(x,y) cleaned image

Image Segmentation

The line response image, R(x,y), is converted to a binary image, B(x,y), shown in figure 42. A binary threshold value of 164 was selected by analysis of the ROC curve shown in figure 44, and false positive curve shown in figure 45. The threshold value of 164 yields a false positive rate of 1%. The corresponding true positive rate is 59.4%.

Threshold values below 70 were not considered. Below that, the regions are very blob like, and are all removed by the region classifier. The ROC curve was constructed from a data set of 20 images of cracks. First, truth images were created, then for each image, the algorithm was run with a binary threshold value from 70 to 255. For each threshold value, the number of true and false positives were counted. The true positive rate, or sensitivity, was calculated by dividing all the pixels correctly found to be in a crack by the sum of crack pixels in the truth images. The false positive rate, or 1-specificity, was calculated similarly.

Empirical analysis showed that although the calculated true positive rate was low, when looking at the images, all cracks were correctly identified, sans some spurious regions, and there were zero false positives. This inconsistency is due to the subjective nature of the truth image labeling process. A crack could be labeled correctly, but shifted to the side slightly in the truth image, thus resulting in a large number of pixels being falsely labeled as being a false negative, and decreasing the true positive rate.

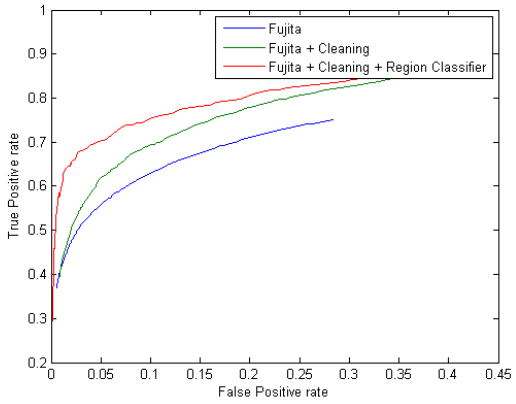


Figure 43 - ROC curve performance comparison

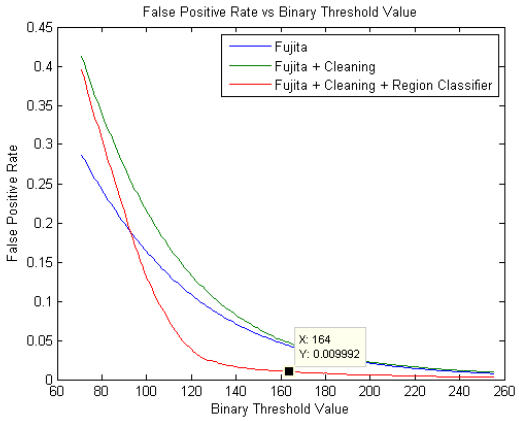


Figure 44 - False Positive Rate Curve of binary threshold levels

Morphological Operations

Five morphological operations are applied in the following order. These reduce noise in B(x,y) and connect lines that are fragmented. [4] The cleaned image, L(x,y), is shown in figure 43

1. Close: performs a dilation, followed by an erosion operation. [7]
2. Bridge: Fills in pixel if it has two unconnected neighbors. [7]
3. Diagonal: Fills in to make 8 connected patterns 4 connected. [7]
4. Spur: Removes pixels that are only singly connected. [7]
5. Clean: Removes isolated pixels. [7]

Region Classifier

Cracks are characterized as having a perimeter that is much greater than its area. This can be expressed by equation (2.12), and a perimeter that is of at least a certain length. [12] It was determined experimentally that there was a 98.2% chance of a region being a crack if the perimeter is greater than 40 pixels, and a 92.7% change if the circularity, F, is less than 0.08. [12] These values were used in our implementation as well. The aspect ratio measurement taken in [12] was determined not to be reliable aspect to consider due to the high possibility of a crack being complexly shaped in the data set we used, rather than linear in the data set used in [5]. The output of the region classifier is the image D(x,y), shown in figure 46.

$$F = \frac{4\pi * Area}{Perimeter^2} < 0.08 \qquad (2.12)$$

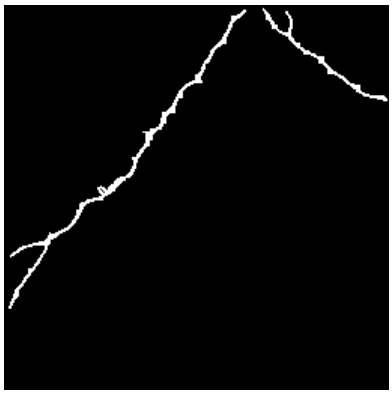


Figure 45 - D(x,y) output of region classifier

Figure 46 - Original image of concrete surface

Algorithm Pseudo Code

A summary of the algorithm written in pseudo code follows:

Crack Detection Algorithm

```
I(r,g,b) = load RGB PNG Image
g(x,y) = convert I(r,g,b) to grayscale image
m(x,y) = median filtered g(x,y) with kernel size WxW
S(x,y) = m(x,y) – g(x,y), create shadow image
for n = 1:N
        Calculate sigma of Gaussian at n
        Construct Gaussian second partial derivative kernels of size H
        Calculate the second order partial derivatives of Gaussian smoothed image
        for (i,j) = dimensions of image
                Calculate the two eigen values of the hessian at I(I,j)
                Calculate line response at (i,j), normalized for Gaussian variance
                R(i,j) = max(Rold(i,j), Rnew(i,j))
        end for
end for
B(x,y) = threshold R(x,y) at T
L(x,y) = cleaned B(x,y) using binary morphological operations
for all white regions in L(x,y)
        calculate circularity and perimeter
        if (circularity < C) and (perimeter > P)
                Add region to D(x,y)
        else
                discard region
        end if
end for
return D(x,y)
```

W = 11, A = 0.5 degrees, T = 164, C = 0.08, P = 40, α = 0.25, $\sigma_1 = \ s = \sqrt{2}$, $N = 4, H = 21$
reported system status. Due to the nature of the application, the interface provides support for username registration and log-in. This currently is only referenced in logging, but has obvious potential for more advanced database and record keeping in future works.

Cost Analysis

Table of Costs by Order Number

Vendor	Part Number	Order Number	Total
Amazon	B003ZSN600	1	$73.36
Amazon	RASPBRRY-PCBA512		
Amazon	B00E1GGE40	2	$34.44
Amazon	LM2596	3	$9.99
Amazon	B00DCAIRIC	4	$6.23
Amazon	RASPBRRY-PCBA512	5	$62.23
Amazon	B003MTTJOY		
Amazon	B00AO0PCMW	6	$20.00
Amazon	B00E1GGE40	7	$32.00
Amazon	B008Q6Z36Q	7A	$7.61
Amazon	B0050G71ZG	8	$5.59
Amazon	B00AO0PCMW	9	$20.00
Amazon	B008Q6Z36Q	10	$9.98
Amazon	B00AO0PCMW	11	$28.97
Boston Eengineering	Shipping	12	$29.49
Ebay	HNM8X1.25SS	13	$5.00
80/20 Inc	330297622005	14	$96.41
80/20 Inc	220327629990		
80/20 Inc	330503519751		
80/20 Inc	370132603224		
Pololu	1444	15	$201.19
Pololu	1493		
Pololu	1308		
Pololu	1939		
Pololu	1208		
Pololu	1555	16	$39.85
Pololu	1492	17	$131.75
Pololu	1442		
Pololu	1308		
McMaster-Carr	91292A156	18	$367.39
McMaster-Carr	8600N4		
McMaster-Carr	57105K22		
McMaster-Carr	6750K141		
McMaster-Carr	6412K11		
McMaster-Carr	8975K563		
McMaster-Carr	9946K12		

McMaster-Carr	92185A118		
McMaster-Carr	91841A005		
McMaster-Carr	92185A131		
McMaster-Carr	92855A309		
McMaster-Carr	6484K446		
McMaster-Carr	95100A101		
McMaster-Carr	93410A110		
McMaster-Carr	6384K39		
McMaster-Carr	86985K31		
McMaster-Carr	7445A12		
McMaster-Carr	8975K429		
McMaster-Carr	91292A022		
McMaster-Carr	92185A998		
McMaster-Carr	9062K273	19	$24.90
McMaster-Carr	92185A991		
McMaster-Carr	91841A195		
McMaster-Carr	6655K72	20	$20.48
McMaster-Carr	91292A029		
McMaster-Carr	6338K413		
McMaster-Carr	92185A124		
Home Depot	0000-841-583	21	$15.94
Home Depot	PR1088001	22	$5.98
Home Depot	PR1088001	23	
Home Depot	1/2 inch Dowel		$7.54
Micro Center	HDMI-DVI	24	$11.96
Micro Center	P-SDH32G10H-GE	25	$19.99
NEU 3D Printing Studio	Printed Parts	26	$70.00
Star Market	Grease Remover	27	$4.29
Star Market	HL EE Gloves MED		$1.49
		Total	**$1,364.05**

Cost Break Down by Category

Mechanical	$613.64
Electrical	$703.18
Misc.	$47.23

We believe that the cost of our project considering its capabilities is extremely low. Before deciding to develop our own robotic platform we researched many consumer available platforms. There were two problems that we discovered during our

research. First, there were not many platforms that were larger enough to handle a lift tower of the height we required and still be able. Second, the cost of any platform that was larger enough was $3,000+ and often did not include electronics. Knowing this, the $1,364 that was spent over the course of the project is remarkably low.

Conclusion

If fully deployed our project would have a profound impact on the standard way that tunnel inspection are performed. Our robotic solution would provide a means to inspect a tunnel from a remote location while also providing a fully automated way to classify, track, and diagnose tunnel structural and nonstructural cracks. Providing a safe, efficient, and smart way of inspecting our tunnel infrastructure would go a long way to preventing further tragedies from tunnel failures in the future.

By developing this platform we were able to successfully apply many concepts that we learned from classes, self-studies, and coop. We all felt that the process from idea conception until final working prototype was incredibly exciting and rewarding and by successfully completing this process we are indeed prepared for the future.

References

[1] Central Artery Project Tunnel Inspection Program http://www.massdot.state.ma.us/portals/8/docs/TunnelSafety/FHWA_CAT_InspectionFinalRpt20111013.pdf

[2] http://www.tunneltalk.com/Safety-Sep2006-Ceiling-panel-collapse-in-Boston-Big-Dig-tunnel.php

[3] http://www.gamepolitics.com/2013/11/21/microsoft-spent-100-million-rd-xbox-one-controller#.U1LT1OZdVTZ

[4] https://code.google.com/p/protobuf/

[5] http://gstreamer.freedesktop.org/

[6] https://java.net/projects/jinput

[7] N. Otsu. Threshold selection method from gray-level histograms. IEEE Trans. Syst. Man Cybern, SMC-9(1):62-66,1979

[8] Fujita, Y.; Mitani, Y.; Hamamoto, Yoshihiko, "A Method for Crack Detection on a Concrete Structure," Pattern Recognition, 2006. ICPR 2006. 18th International Conference on , vol.3, no., pp.901,904, 0-0 0

[9] Richard O. Duda and Peter E. Hart. 1972. Use of the Hough transformation to detect lines and curves in pictures. *Commun. ACM* 15, 1 (January 1972), 11-15. DOI=10.1145/361237.361242 http://doi.acm.org/10.1145/361237.361242

[10] Choudhary, G.K.; Dey, S., "Crack detection in concrete surfaces using image processing, fuzzy logic, and neural networks," Advanced Computational Intelligence (ICACI), 2012 IEEE Fifth International Conference on , vol., no., pp.404,411, 18-20 Oct. 2012

[11] Xuhang Tong; Jie Guo; Yun Ling; Zhouping Yin, "A new image-based method for concrete bridge bottom crack detection," Image Analysis and Signal Processing (IASP), 2011 International Conference on , vol., no., pp.568,571, 21-23 Oct. 2011

[12] Convert RGB image or colormap to grayscale – MATLAB rgb2gray. (2014). Retrieved April 19, 2014, from http://www.mathworks.com/help/images/ref/rgb2gray.html

[13] Morphological Operations on binary images – MATALAB bwmorph. (2014). Retrieved April 19, 2014, from http://www.mathworks.com/help/images/ref/bwmorph.html