AUTOMOTIVE GESTURE RECOGNITION

MAY 13-07

Scott Schroeder, Thomas Moser, Ethan Little

Client: Garmin

Advisor: Dr. Tom Daniels

4/28/2013

EXECUTIVE SUMMARY

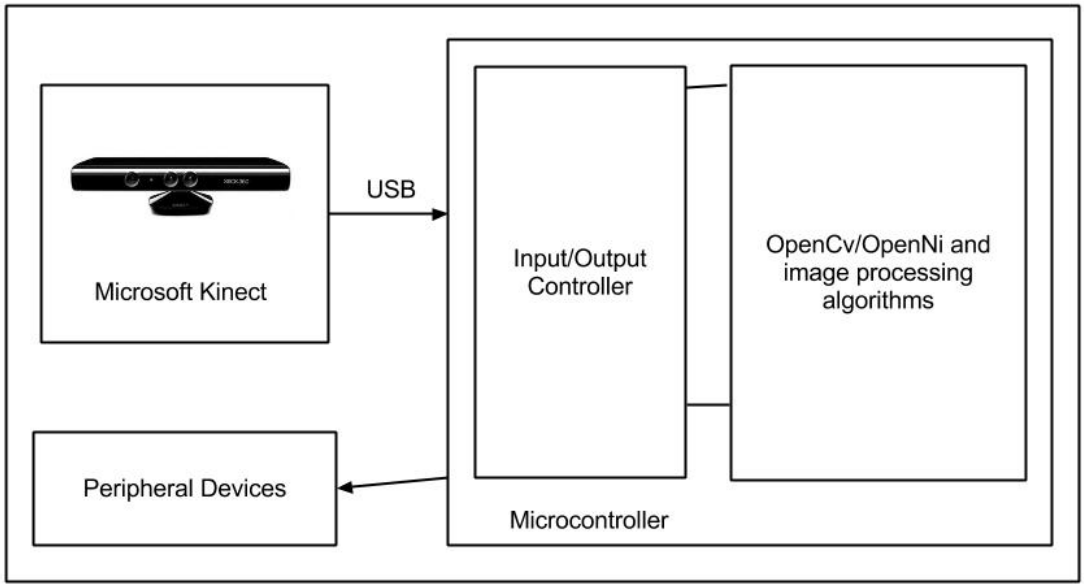
Gesture control is an increasingly popular feature in devices from TVs to computers to video games. Allowing users to interact with their device using natural motions in a three dimensional space gives a hands on feel. Safety while driving has also become a major issue in recent years, with distracted driving being a focus. Preventing drivers from simple distractions could be one step in the prevention of accidents.

Gestures in a vehicle would allow a user to maintain focus on driving and not move attention to finding the correct button or knob to make changes on the stereo system. Multiple methods of gesture and hand recognition were developed and tested during the development of this product. The final version of this product is capable of recognizing simple control gestures and displaying an output to the user.

PROBLEM STATEMENT

Some in the industry see gesture recognition as a technology that will make in-vehicle systems easier and less distracting to operate. Garmin wants to explore the best applications of gesture technology for its next generation original equipment manufacturer infotainment systems. These systems will include an array of modes/features including navigation, audio/entertainment, communication, vehicle information and other miscellaneous applications (streaming music applications, social media, etc). The system must provide usability advantages over switches, buttons and touchscreens, providing either safety (i.e., less distraction) or convenience advantages by using gesture recognition technologies.

CONCEPT SKETCH



OPERATING ENVIRONMENT

The main purpose of the product is to make driving an automobile more intuitive and safer. Given this information, the final operating environment in which the product will be used is inside of an automobile. Specifically, the product will be imbedded in the automobile's infotainment system.

When designing this product there were many considerations that directly corresponded to the operating environment. For instance, an automobile can be used in any type of weather and any type of lighting. This means that the end product has to

be able to withstand a varying range of temperatures and it must be able to work during both the day and night. In an automobile environment there will also be a lot of background noise during image capture due to occupants in the back seat of the automobile.

Another very important aspect of the operating environment that was considered during the development of the product was automobile safety. The main implication from this is that the product must be reliable and must not constitute a distraction for the automobile operator. This has a significant impact on the non-functional requirements of the product because it must have an almost non-existent false-positive rate as well as a relatively high rate of positive recognition.

INTENDED USE AND INTENDED USERS

The intended use of the product is to allow an automobile operator to dynamically interact with their automobile through the use of predefined gestures. The unit must act as an interface between the user and the automobile's center console infotainment system. This will allow the user to focus on the road instead of searching for and pressing a certain button.

With the product, users should be able to:

1. Interact with center stack display interface with the ability to scroll lists, pan/zoom maps, seek up/down (radio, tracks), and switch modes (nav, audio, communication, etc.) with the users arm and hand in a comfortable position (i.e., without reaching for the display or other controls). Gesture recognition is not meant to completely replace a touchscreen or other buttons, but rather to be a complementary technology.

2. Interact with automobile menus to make settings changes and request information. This means the driver does not need to take their eyes off the road to look at the dashboard for information.

The intended users of the product consist of anyone operating an automobile and anyone riding in the front passenger seat.

ASSUMPTIONS

When initially approaching the design it was determined that a few assumptions would be made about the environment the system would be set up in in order to proceed further.

First, it was decided to focus on the typical North American car setup, that is, the driver in the front left of the vehicle with the center of the vehicle to the driver's right hand side. This meant only the driver's right hand would be looked at. The final system eventually allowed detection of both hands so either the driver or the passenger could control the system. In order to cut out noise from the depth image it was assumed the system would only be used by a user in the front seat of the vehicle and therefore decided to limit the maximum depth that was read by the sensor. This would prevent things in the backseat from being detected and causing interference. The minimum depth was determined by the Kinect, which we assumed would be adequate. With the depth taken care of, the width of the viewing area was focused on. It was assumed that the Kinect's native resolution would work well for this application and give the user enough room to make their gestures.

For our hand detection algorithm we assumed the driver of the vehicle would have hands above a certain size. This affected our calculations during the concavity detection phase of the algorithm and led us to have trouble recognizing smaller hands. We also made the assumption that the driver would have all five digits intact in order to decrease the likelihood of false positives.

DELIVERABLES

The final product design is an operating prototype for a gesture recognition system consisting of an infrared depth array and a processing platform.

The depth array currently used for the prototype was a Microsoft Kinect. This was chosen because it was the easiest to obtain and the most well documented depth array available at the beginning of the project. It also offers some unique benefits when it comes to background segmentation. Further benefits are discussed in the hardware description section.

The processing platform that is currently used for the prototype is a desktop computer with a Linux operating system. In the future the plan is to scale down the software platform so that the product will run on a smaller computer such as a Raspberry Pi or on a microprocessor.

APPROACH USED

During development of the product there were many requirements that had to be met. In the pursuit of meeting these requirements there were many design constraints that needed consideration. It was also important to consider what products are already on the market as well as what types of algorithms could be useful in detecting gestures.

The following subsections go into detail on the entire approach that was used while designing the final product.

FUNCTIONAL REQUIREMENTS

- Recognize 4-5 distinct hand gestures

NON-FUNCTIONAL REQUIREMENTS

- Limit processing power required to run gesture recognition
- Minimize cost
- False positive and false negative rate less than 5 in 100
- Shallow consumer learning curve (i.e. easy gestures)

DESIGN CONSTRAINTS

The constraints the design was put under can be broadly classified into three different categories: sensor, hardware, and environment.

The Kinect sensor has a set resolution, below what was wanted, that limits the gestures that can be used. For this reason, simple, broad movements were used instead of intricate finger motions.

With the system successfully running on a desktop computer it was a goal to scale down the program to the point where it could be transfered to a microcontroller to save space and money. This is where one of the largest constraints, processing power, was encountered.

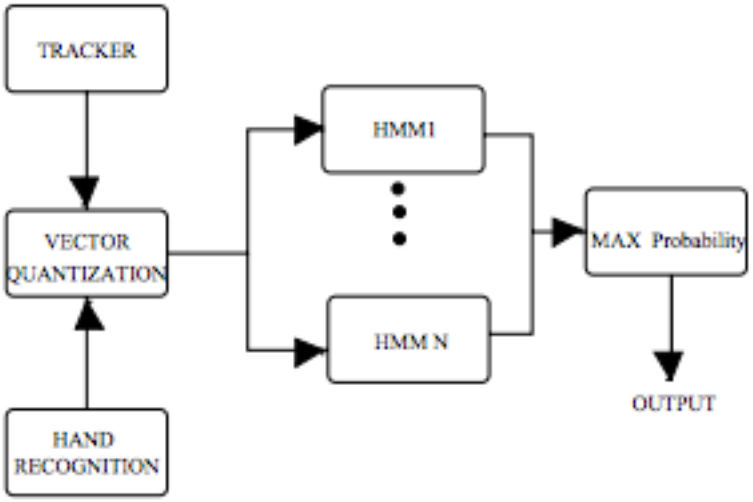
The final design constraint was the vehicle environment itself. The gestures were limited to single hand motions in a limited space, including a limited depth

in order to cut out noise. Most of these risks were able to be mitigated by making some assumptions about our hardware and environment.

MARKET/LITERATURE

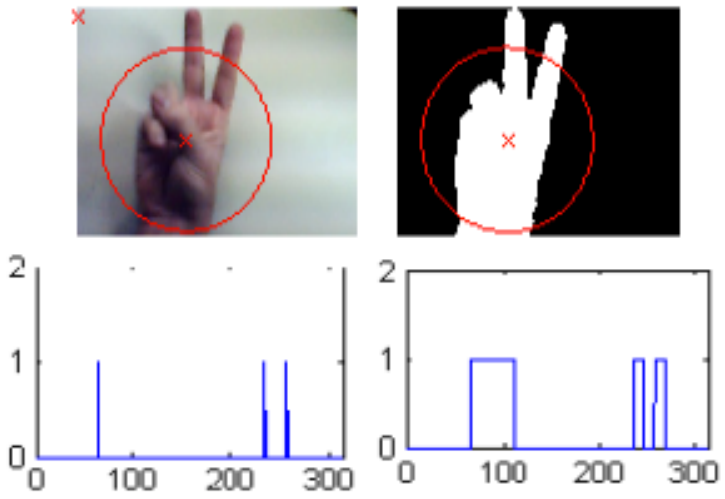
Gesture recognition is a topic which constantly inspires new research and developments with the eventual goal of completely natural interaction with the devices people use every day. While many systems in existence work for gesture recognition, very few are designed to work in automobiles with such accuracy and with the processing constraints found in the system requirements. After researching and testing the following methods were used as a basis for this program.

An approach was designed back in 2003 using hidden Markov models (HMM) as its basis [4]. The system was designed to track a hand shape through a path and recognize what path it followed. Once the shape the hand was making changed a new gesture would be recorded and tracked. The detection of a hand shape was done using a system similar to a template matching. Once the shape was detected it was followed using a Kalman filter to trace out a path. A Kalman filter uses a series of predictions and corrections to obtain a smooth estimated path of travel, giving a means of correcting small variances and losses of data. The path is then recognized using HMM as the main method and reported as a probability of occurrence. Accuracy of the system was reported to be around 97%, which is easily in the range acceptable for this project.



HMM is proven to be a good way to do gesture recognition and work with a high accuracy. However, this system's requirements are not able to afford the processing power required to perform such calculations. The system needed in a car should also allow a user to have a varying hand shape to allow natural motions, which the HMM system doesn't allow for. Because this system has a more limited range of gestures it is possible to design a system without using a more advanced gesture recognition algorithm such as using HMM's.

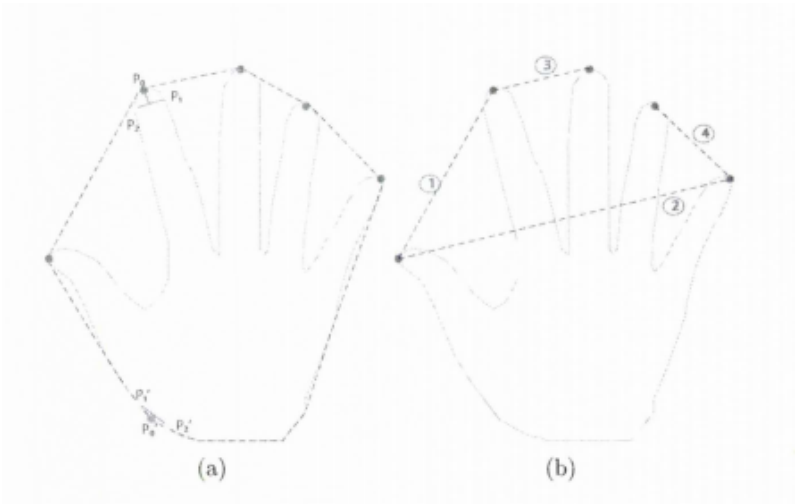
Fast and low power methods for hand interactions have been developed for use in controlling robots[5]. To increase the processing speed of the system, only static hand recognition with finger counting is implemented. A hand is separated from the background by using color segmentation of the skin. The center of mass of the hand image is then calculated to find a spot close to the center of the palm. A circle is created with the calculated blob center of mass as its center and a radius large enough to cut through the fingers. The path around the circle is followed and the number of displayed fingers is calculated as the number of intersections. A gesture is finally determined to be the number of fingers displayed.



Their system, as many others, uses color segmentation to determine a hand. This does not account for other hands or people in the background as may occur in an automobile setting. The reported accuracy number of 91% was due to lighting issues with the images, something that is a common problem in the cabin of a vehicle. The current project also needs to deal with natural moving gestures and not relying on a driver thinking about which number to display. These differences prevent the system from being a basis for this project but the speed of hand recognition is something desirable. Also, the

method of determining the number of fingers could be used in future iterations to detect an open hand as the initialization gesture.

A system using Kinect as an input device was used by Yi Li in a Master's thesis [3]. The project was created as an improvement to existing sign language recognition systems and therefore is used with static gestures. The hand is segmented using depth instead of skin color, allowing for the use of gloves. This also eliminates issues with lighting and color capture. Once the hand blobs are segmented out a K-clustering algorithm is used to group the points into blobs that can be assumed to be hands. The shape of the hands is found using a contour tracing algorithm that iterates around the outside edge of a blob. Graham Scan algorithm is used to find a convex hull of the hand points. This bounding hull shape is the smallest shape that contains all of the points and has no concave edges. Using this hull and the hand shape the points located on both are saved and a three point alignment is used to determine what angles are sharp enough to be classified as a finger. Further calculations are done to label the fingers and then the gesture is recorded.



The idea of using a hand contour and a convex hull to determine a hand was decided to be a basis for the hand recognition portion of this project, however it was decided to try to improve the finger detection by using defects as described later. The sign language recognition system created performed at over 90% accuracy over all of the gestures with a near 100% hand detection rate [3].

HARDWARE DESCRIPTION

When choosing hardware for the design a variety of sensors using different technologies were looked at. The main technologies researched were complementary metal-oxide-semiconductor (CMOS) cameras, IR cameras, and sonar sensors. All three types provide a means of recognizing gestures but each has its own set of drawbacks as well. The criteria for selecting the sensor included attributes such as cost, performance in varying lighting, and detail of gestures that could be recognized. Low cost of the sensor is a factor not only in this project, but a necessity for future implementation in OEM products. Automobiles are driven at all hours and in all lighting settings,so this system needs to operate with the same degree of accuracy regardless of what the lighting situation is. All three systems are capable of recognizing gestures but the degree that they can do so varies.

CMOS cameras are the least expensive of the options and readily available. Using the RGB images from the camera hands can be extracted using the skin color and then used for tracking. A CMOS sensor provides a simple, inexpensive solution but doesn't meet all of the requirements. Vehicle environments are almost never constantly lit and sometimes not much light is available at all, providing huge challenges for image capture. Interference from the rear passengers also comes into play with a standard camera. Events such as children playing games in the back seat or a person reaching across the rear cabin for something could cause accidental gestures and false positives with a standard camera, which would likely result in a greater distraction than initially looking at the dash controls would have.

Non-visual options such as a sonar array were discussed in the hardware decision stage as well. Cutting out the camera all together would remove any issues with the lighting in the cabin as well as reduce interference from passengers in the back by localizing the sensors to the front of the vehicle. However, precision is greatly reduced with this method however, to the point that only the general location of the hand in the cabin can be determined. This could be a useful way to detect proximity or to control a few items but our system needs to be expandable and the ceiling is not as high with sonar.

The hardware chosen for the project is being used for research in many areas and has revolutionized computer interaction by removing the price barrier of entry. Microsoft's Kinect is an inexpensive sensor capable of capturing depth data, giving a three dimensional view of the environment. The hardware consists of a VGA camera, a microphone, and an IR depth sensor. Both cameras are capable of capturing video at 30fps at a resolution of 640x400,which is more than enough for this application. The IR depth sensor works with two parts to capture and determine depth. An emitter projects a scattered array of

points out from the device while a separate filtered camera looks at them. The dots are arranged so that each "pixel" has a unique pattern and each pattern then has an expected angle that the sensor should find it at. Using this expected angle, Kinect calculates the distance to the spot where the marker was actually found. The work that the Kinect does allows us to retrieve a depth array from the device that is the base of our system. The IR pattern works with no external light and would allow the system to work at night. The depth information also allows for the filtering out of objects that would be in the back seat or would be considered noise.

Microsoft provides an SDK to ease development and demonstrate the functionality of the Kinect. This kit includes applications and methods for body tracking and gesture recognition and is designed for a Windows environment. This project needs to be based on a Linux platform to attempt to be as close as possible to the final environment so it was decided not to use the SDK or any of the premade gesture systems. Instead, to obtain the depth data from the Kinect, a driver made by PrimeSense and the OpenNI library was used.

SOFTWARE DESCRIPTION

The system was designed to be able to run on any computer operating system and developed primarily on Linux to ease a move to a microcontroller in the future. All libraries used are open source and move easily between operating systems. OpenNI and OpenCV are the libraries chosen to include and fit all of the requirements.

OpenNI is a framework used to interact with 3D sensing devices and provides the tools to develop applications using these sensors. The SDK includes device drivers, middleware libraries, and premade applications. The OpenNI framework is used as a replacement to the Microsoft SDK to allow portability across systems. This program makes use of the framework to retrieve the depth data from the device and return it as a matrix of depths.

OpenCV is a real time image-processing library originally developed by Intel. The library is used across all platforms and even on mobile devices. In developing the gesture system OpenCV was used to process each frame and used the structures included in the library for storage. OpenCV contains many optimized versions of algorithms that allow fast processing of images without powerful computing power, fitting in perfectly with the requirements.

TECHNICAL APPROACH

Development of this project began with narrowing the scope to match the qualifications of the team. Hardware design was removed from the scope early in the project and the software aspect of the project was chosen for the primary focus. Since hardware was no longer being designed it was necessary to choose a platform and existing hardware to work with. After consideration of sensor choices the Microsoft Kinect was chosen and Linux was used to ease the movement to a microcontroller. Libraries were chosen shortly after and the design of the software was started.

The system should be designed so that a user would not need to think about the gesture they want to perform. Initial ideas for gestures included swiping left or right to change tracks or menu items and moving a finger in a circular motion to control volume. The main functions the project should have are volume control and track selection as well as play/pause functionality. Looking at these controls a base of four starting gestures were selected with play/pause being omitted. With the gestures restricted to being simple and natural motions, the problem of filtering out all incidental motions arose. These motions could include things like waving at a passing car or reaching to use a manual control on the stereo. Detecting these accidental gestures would significantly increase the false positive rate so a way to distinguish the start of a gesture was needed. A gesture initialization step, such as requiring the user to display an open hand, was decided as a starting point for all gestures. The user would be required to display a fully opened hand to the sensor for a set period of time. A beep would then signal that a hand was recognized and the gesture would be started from there. This was done to prevent accidental interference and to give the user a feeling of confidence that they will not be interfering with the system while going about normal activities.

With the start of a gesture determined there was also the problem of finishing a gesture and being able to remove the hand from the view of the sensor without causing another inadvertent gesture. To fix this issue a timeout and hand travel distance limit were implemented. Once a gesture is started, a counter is also started, which limits the amount of time a user has to perform a gesture. If the timer runs out, the hand recognition process is started again with the open hand required to start another gesture. Another way a user could exit the gesture loop is by moving their hand too far in any direction from the initialization point.

With the full gesture captured, an algorithm needs to be run to determine what the gesture was. The original idea was to use a template matching by first creating templates of ideal gestures and then matching them to the recorded

gestures, calculating a percent match. This would allow for many shapes and directions of gestures, but was more than necessary to accomplish recognition of four base gestures. Since two of the decided gestures were horizontal swipes it was decided that the other two volume gestures could also be swipes, but in the vertical direction. With only four directional gestures template matching could be removed and a much faster linear regression could be performed. This idea limits the gestures to directions but could be expanded to include circular regressions for the initial proposal of circular gestures. The linear regression also allows for the correlation coefficient to be calculated in order to determine how close the fit was.

TESTING APPROACH

Testing of the project was to be divided into code testing, hands on testing, and outside user testing. Early plans also included testing with recordings of gestures to ease development without using the sensor hardware but this was not implemented. Functions and sections of code were to be individually tested while most testing would need to be done by interacting with the sensor. With a working product established, the most useful testing data would be collected through documenting unfamiliar users interacting with the product. Recording what things they attempt to do and what problems they have would provide insight into how to make the system as natural as possible.

DETAILED DESIGN

The following section discusses, in detail, the final hardware configuration for the project. The algorithms and steps that are involved in the project are also discussed in great detail.

SETUP

The final product can be broken down into four main sections: Image capture and preparation, hand detection and recognition, gesture detection and recognition, and output. Each of these sections contains smaller steps, which can be seen in the data flow diagram in Figure 1.

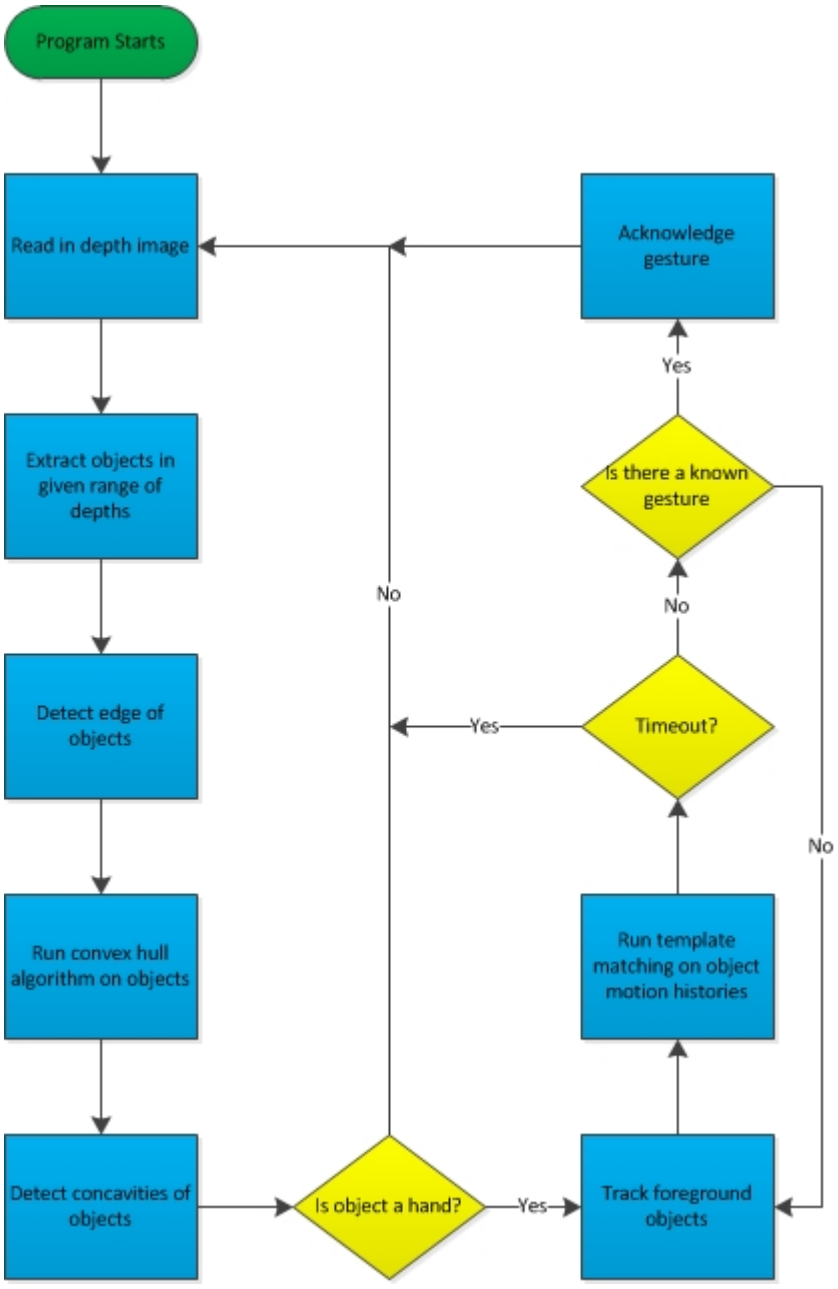

Program Starts
Read in depth image
Extract objects in given range of depths
Detect edge of objects
Run convex hull algorithm on objects
Detect concavities of objects
Is object a hand?
No
Yes
Track foreground objects
Run template matching on object motion histories
Timeout?
Yes
No
Is there a known gesture
No
Yes
Acknowledge gesture

Figure 1.

In order to facilitate image capture the Kinect sensor is placed approximately two feet in front of the intended gesture zone. While the product is running, the Kinect will constantly be recording depth video to be processed with the algorithms described in the following sections.

IMAGE CAPTURE AND PREPARATION

At the beginning of each input frame processing cycle, the Kinect captures a depth image and then sends it to the program for pre-processing.

The first step that is taken is to filter the depth input to concentrate only on a given range of depths. By default, the input image from the Kinect is a 16-bit depth image that corresponds to depths in a range from 800 mm to 4000 mm. In order to decrease this range so that it goes from 800 mm to 1400 mm, the captured frame is sent through a couple of depth filters which look through all of the pixels in the frame and set those pixels not in the given range to 0. This process makes it very easy and efficient to filter out the background noise in a car environment.

The second step is to filter the image further so that the only object left in the depth image is the closest object. This is done so that objects don't overlap and combine to create larger nonsense objects. To find the closest object, the depth image is iterated over to find the minimum value that is greater than 0. This corresponds to the closest point of the closest object. The depth image is then filtered so that it only contains object in a range starting at this point and going back 152 mm.

At this point all of the image preparation is finished and the hand detection and recognition phase is ready to begin.

HAND DETECTION AND RECOGNITION

Detection and recognition of a hand was done through a series of algorithms and used as the starting point of all gestures. An open hand needs to be displayed for a time period of two seconds to signal the start of a gesture. During this time when the system sees a hand, it is outlined with an orange frame, telling the user a hand is seen. At the end of the two seconds the user is prompted via a change in color of the hand frame outline to red, which is when the gesture recording is started.

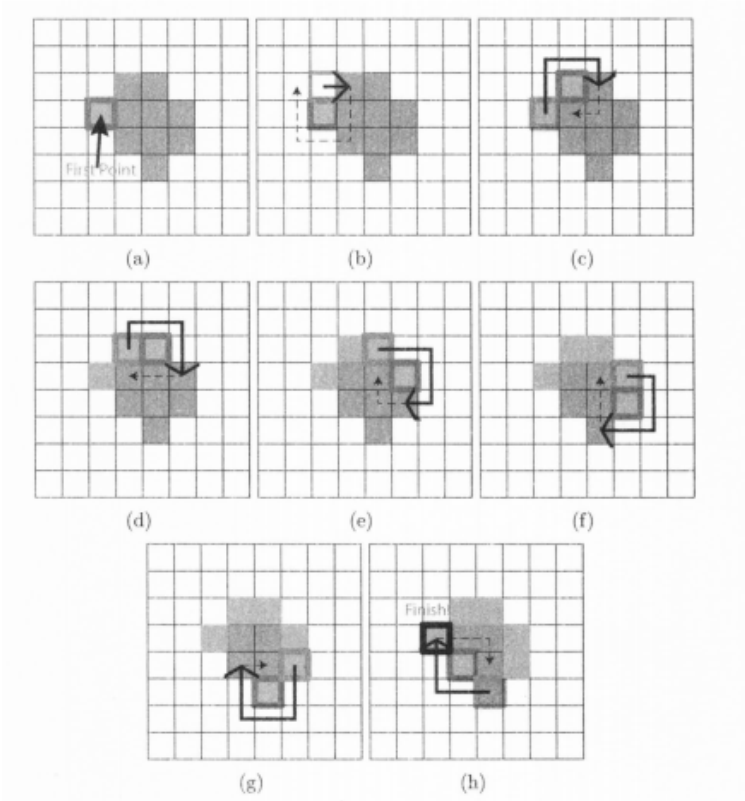
To accomplish the recognition of a hand, the algorithm starts with the filtered image containing the closest 152 mm. This image should contain only a hand

and a portion of the wrist of the user. This image is then passed through a series of steps with the final output being a yes/no determination of a hand being in the frame. The following steps are required in our hand recognition:

CONTOUR DETECTION

Outlining shapes of the hands are detected using an algorithm by Suzuki [1]; The algorithm finds the outermost edge to the shape and returns it in a matrix. To find the outermost contour, the following steps are used:

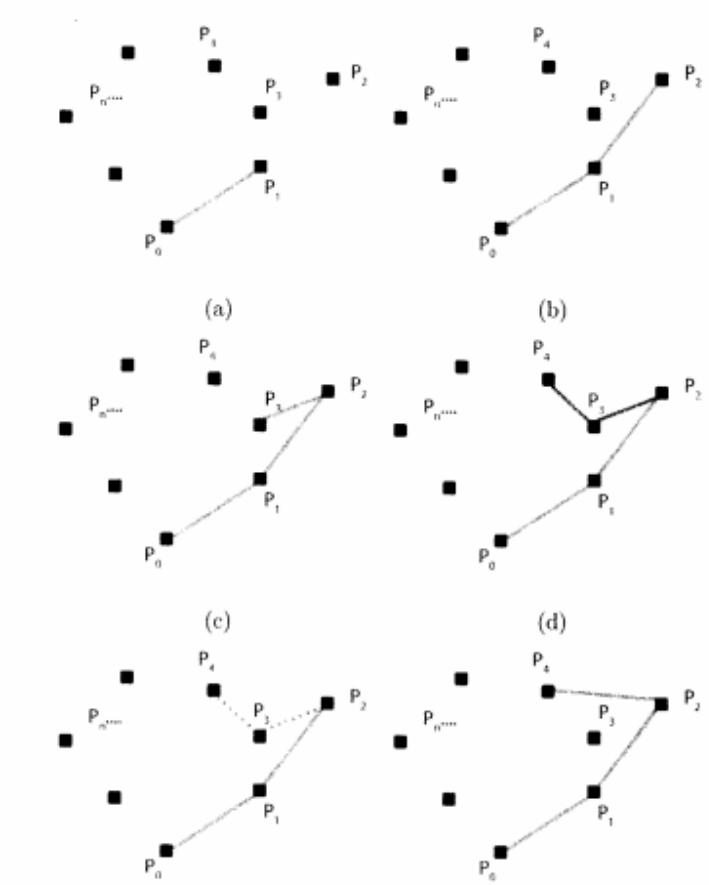
- Iterate through the pixels, left to right, from the top of the frame. Stop when the first pixel of the shape is found.
- With the first pixel selected, choose the spot immediately above it as the starting point and iterate around the first pixel in the clockwise direction looking for the next active pixel.
- Add the next active pixel to the contour and move to that pixel to repeat the process.



CONVEX HULL

Calculating the convex hull is done using a Graham Scan algorithm [2]. This algorithm gives a bounding box around the image that contains no concave angles. The convex hull is obtained with the following method:

- Choose the bottom most pixel of the active blob as a starting point.
- Calculate the angles to all of the other active pixels and arrange them by angle.
- Choose the angle that corresponds the to the pixel closest to the right.
- Move to that chosen point and repeat.
- If the angle selected is a clockwise turn from the previous angle, remove the previous point and go back.

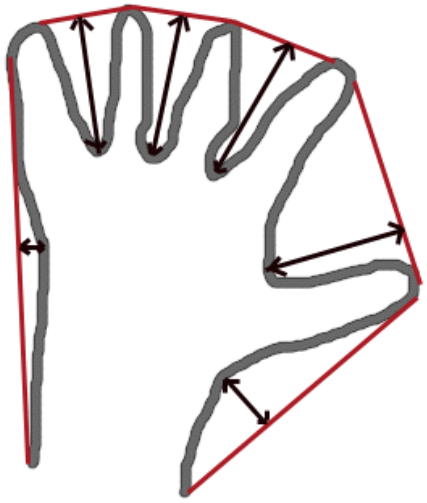


CONCAVITY DETECTION

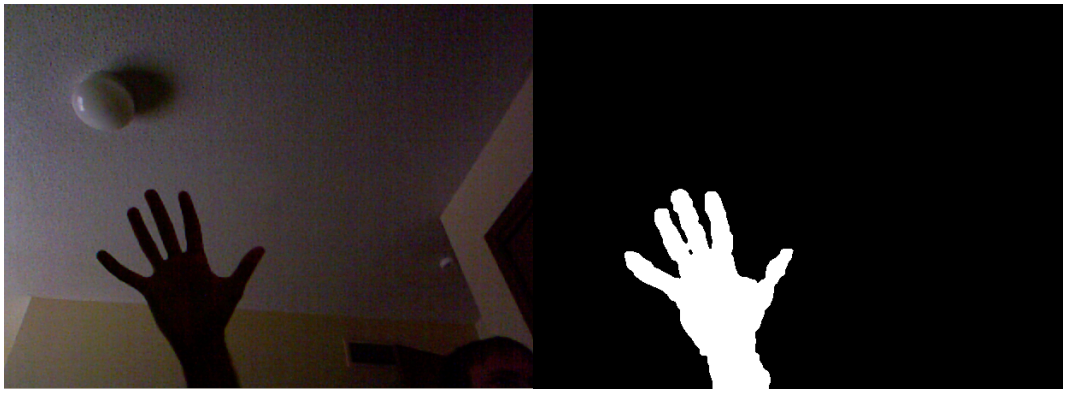
When looking to determine where the fingers are, we examine the defects in the convex hull of the shape. A defect is a spot where the hull is not on the hand contour and there is a gap in between. When looking at the defects we can check to see if there are enough of them, and if they have the correct depth to indicate fingers being present. The concavity detection algorithm works in the following way:

- Start with the point furthest towards the bottom that is on both the convex hull and the hand contour.
- Moving around the convex hull in a clockwise direction, record the distance from the current hull point to the point on the contour that is perpendicular to it.

- Record the depth and location of the deepest spot between each two points that are on the contour and the convex hull.



Final determination of a hand shape is done by looking at the number, as well as the size of the defects. The size of the object is also considered when looking at the contours. Anything above or beneath a certain threshold is thrown out and not considered. When a hand has been recognized for 10 consecutive frames an indication is made to the user and the function returns a true result as well as the location of the center of the hand, found through calculating the mean of the blob.



GESTURE DETECTION AND RECOGNITION

OBJECT TRACKING

Once a hand has been recognized, the product switches over to gesture detection mode. In this mode the product still goes through the image capture and preparation step, but now skips over the hand detection step and instead does blob tracking.

In order to track the different foreground blobs on the screen, (in this case there should only be one blob corresponding to the hand) the program uses an OpenCV function to find all the contours in the current image. Once the contours have been found the program calculates where the centroid is for each object. The centroid for each moment in time is then stored in an array to be accessed later.

LINEAR REGRESSION

Once the centroid of the object has moved past either the horizontal or vertical gesture distance thresholds the program enters a gesture matching/recognition phase. In this phase the array of centroids is used to determine what gesture, if any, was being made. For this project we concentrated on only four gestures:

left swipe, right swipe, up swipe, and down swipe. To determine if these swipes were being made the program runs a simple linear regression algorithm on the centroids to determine if the slope of the resultant line lies in the horizontal range or the vertical range. The correlation coefficient is also calculated to ensure that the data is somewhat linear.

GESTURE MATCHING

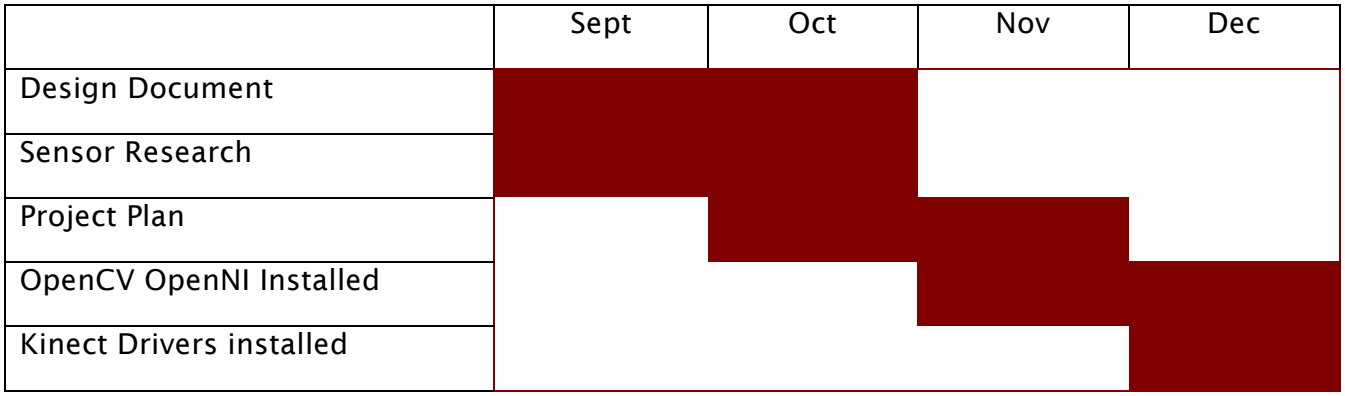
If the linear regression data shows that the motion corresponds to any one of the four swipes, the program then outputs the recognition to the standard output device and returns to hand detection mode. Consequently, if the program detects that none of the swipes has been detected the program also returns to hand detection mode, but no output is generated at this time.

Throughout the whole motion detection phase the program also keeps a timer going. This timer is used to determine if the user is taking too long to make a gesture. The timer is important because it stops the program from being stuck in an infinite loop in the case that no gesture is ever detected.

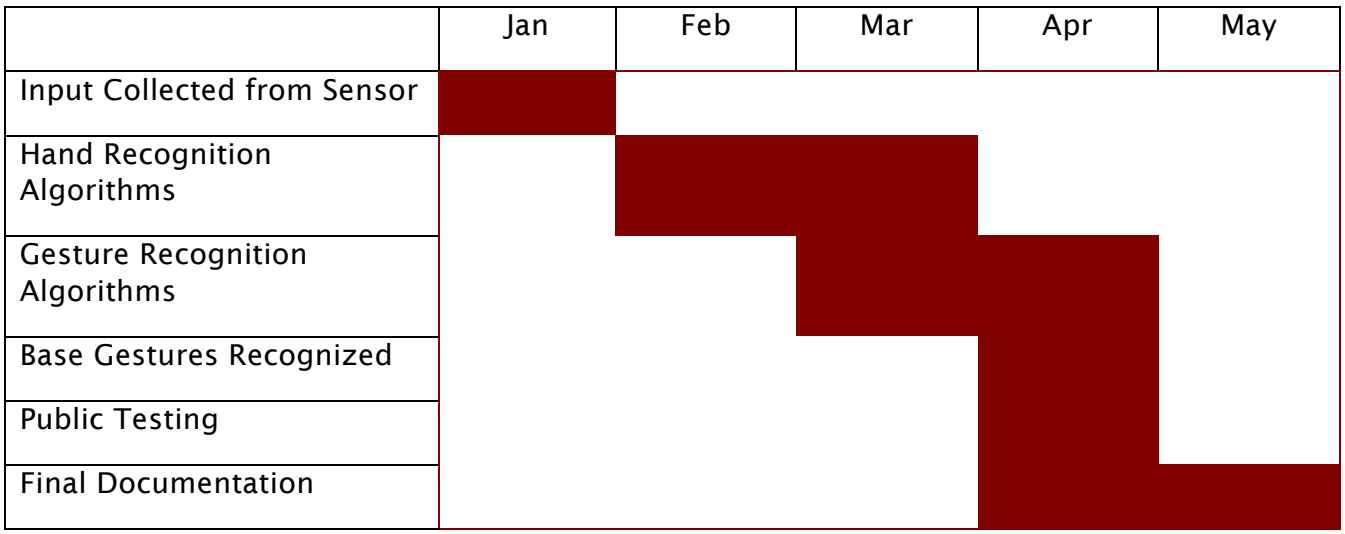
OUTPUT

The product only produces output when one of the predefined gestures is recognized. Once a gesture is recognized the product will send out a signal to the device it is meant to interact with (in this case the infotainment console of an automobile). This signal will then be read by the infotainment console and the action corresponding to the signal will take place.

PROJECT SCHEDULE



f



WORK BREAKDOWN

Work was divided between group members according to their experience and preferences. Scott and Tom worked on a majority of the programming due to being enrolled in an OpenCV/Computer Vision graduate course.

NAME	DOCUMENTS	PROGRAMMING	RESEARCH	WEBSITE	PRESENTATIONS	MEETINGS	TOTAL
Scott	15	100	29	12	15	16	187
Tom	15	95	31	0	15	16	171
Ethan	15	10	26	20	15	16	102

EXPENSES

The anticipated cost stated in the Garmin proposal for this project was $200. A total of $145 was assessed by the design team, but the $100 Kinect was provided by the Iowa State Computer Engineering Department. At the end of this project the Raspberry Pi and SD card have not been used, but they will be necessary for future iterations of this project.

PRICE BREAKDOWN	COST
Microsoft Kinect	$100.00 (provided)
Raspberry Pi	$35.00
SD Card	$10.00
Total	$145.00

STANDARDS

By implementing a gesture recognition system, the goal is to make the driving experience safer for the vehicle's occupants. This system helps to avoid some of the potential distractions that the modern driver faces. One of the goals was to make sure everything asked of the driver would make them safer, and also not put them at risks of violating automobile safety laws.

Other standards our system implemented were USB in the sensor interface, the C++ standard library, and the OpenCV and OpenNI libraries.

TESTING

Testing was conducted in three main steps: code testing, hands-on testing, and external user testing. Each type of testing provided verifications of the product's working state as well as improvements that need to be made. The majority of the testing was hands on testing as user interaction is a large part of the system.

CODE TESTING

Functions were tested individually using hard coded test cases, which allowed for verification of working sections and didn't require input from the cameras. Traditional unit testing was not much of an option in this project as the input is constantly changing and few individual functions are needed.

HANDS-ON TESTING

During development of the product hands-on regression testing was used to make sure that the project was headed in the right direction. After each new step/algorithm was implemented in the C++ code, the overall product was tested to make sure that the existing code interacted correctly with the newly implemented code.

Once all of the algorithms were implemented in code, the product was tested thoroughly to ensure that the functional and nonfunctional requirements were met. Specifically, it was important to make sure that the false positive rate remained around 1% and the correct recognition rate remained around 95%.

The experiences taken from hands-on testing helped to polish the interaction with the product so that it could easily and consistently detect idealized gestures. The data collected from hands-on testing can be seen in the test results table in Figure 2 under External User Testing.

EXTERNAL USER TESTING

After hands-on testing by the developer team was completed the product was tested by the general public to see how well it fared in an uncontrolled environment. The main external user testing took place during the VEISHEA 2013 demo of the product. During this demo, roughly 60 people tested the product. The collected data from this demo is given in Figure 2.

One of the main points of interest is the disparity in the correct recognition rate between uncoached attempts at using the product and attempts where coaching had taken place. On average, the first untrained attempts at making gestures received around a 50% recognition rate, which was well below the desired 95% nonfunctional requirement. After the test subjects were told how the program worked and coached on how to use it, however, the recognition rate jumped up to around 85%. One of the main reasons that this rate stayed below the 95% goal was that people had a hard time getting the upwards gesture to work correctly. This is mainly due to that fact that while using this gesture many people pushed their forearm forward so that it was included in the object tracking. This led to erratic centroid locations. It is also important to note that after more time practicing, this rate continued to increase.

One of the most important things to note, however, is that even without coaching the false positive rate of the product only averaged around 2%, which is very close to the 1% rate being aimed for in the nonfunctional requirements.

Gestures	Internal Testing	External Testing	
		Before Training	After Training
Left Swipe	93%	53%	91%
Right Swipe	95%	60%	92%
Up Swipe	90%	42%	70%
Down Swipe	92%	49%	87%
Overall	93%	51%	85%

Figure 2.

ADDITIONAL IMPROVEMENTS

As this is a proof of concept project there are still many improvements that can be made before final implementation. Some of the improvements that would improve the product going into the future are:

IMPROVE HARDWARE

Although Microsoft's Kinect provided the project with a good base sensor for a reasonable cost, it is recommend that an improvement of the current hardware is used in future iterations. The increased resolution of more recent technologies would give a good testing platform for more detailed gestures. However, for a final implementation a custom IR sensor should be created. This would allow the restrictions of the sensor to be customized to the automobile setting to improve the efficiency of the project.

REMOVE OVERHEAD

While libraries such as OpenCV and OpenNI provided some useful and efficient tools, they should be gradually removed and replaced with low level implementations of the functions. Both libraries provide far more tools than this project requires and some of the functions used do more than is necessary in the current application. Reducing these functions would help to increase the speed and size of the project.

ADDITIONAL GESTURES

The current implementation only supports four basic gestures but future iterations should be expanded to include up to 20 gestures. These additional gestures would require changes to be made to the recognition algorithm and an improved sensor to be used. Simple gestures such as static fingers could be an easy way of increasing control over radio selections. As gestures are added, it is recommended that the user's preferences on what is natural be considered, as this helps to increase safety.

CONCLUSION

The objective of this project was to design and implement a software solution that would facilitate the interaction between human made gestures and an automotive infotainment system. The first half of the project consisted of researching and documenting the different algorithms and hardware peripherals that would be used. Since this project consisted mostly of software development this step was extremely important because it made sure the algorithms chosen would be efficient and that the final operating behavior could be approximated. The second half of the project focused on the actual implementation of the product and making sure that it tested correctly in the real world. In the end a C++ solution was created using a Microsoft Kinect sensor, a basic Linux PC, and the OpenNi and OpenCv software libraries. The final product met the goals specified in the project requirements, but this is just the first step in improving safety and user interaction in automobiles.

REFERENCES

[1] Suzuki, Satoshi. "Topological Structural Analysis of Digitized Binary Images by Border following." *COMPUTER VISION, GRAPHICS, AND IMAGE PROCESSING*. 30. (1985): 32-46.

[2] GRAHAM, RONALD. "Finding the Convex Hull of a Simple Polygon." *JOURNAL OF ALGORITHMS*. 4. (1983): 324-331.

[3] Li, Yi. "Hand Gesture Recognition Using Kinect." *J. B. Speed School of Engineering of the University of Louisville*. (2010): Web. <http://digital.library.louisville.edu/utils/getfile/collection/etd/id/2375/filename/5172.pdf>

[4] Ramamoorthy, Aditya, Namrata Vaswani, Santanu Chaudhury, and Subhashis Banerjee. "Recognition of dynamic hand gestures." *PATTERN RECOGNITION*. 36. (2003): 2069-2081. Web. <http://home.engineering.iastate.edu/~namrata/gatech_web/HandGesture.pdf>.

[5] Malima, Asanterabi, Erol Özgür, and Cetin Müjdat. "A Fast Algorithm for Vision-Based Hand Gesture Recognition for Robot Control." Web. <http://webfiles.sabanciuniv.edu/mcetin/publications/malima_SIU06.pdf>.