DO YOU SEE WHAT I'M SAYING:
RELATING LANGUAGE AND VISION TO CREATE INTERACTION BETWEEN HUMANS AND ROBOTS

A Thesis

Presented to the Faculty of the

Department of Computer Science

Kutztown University of Pennsylvania

Kutztown, Pennsylvania

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

---

by

Jeffrey W. Minton

April, 2011

Approved

(Date)	(Adviser)
(Date)	(Reader)
(Date)	(Chair, Department of Computer and Information Science)
(Date)	(Dean, Graduate Studies)

Abstract

It has been seen countless times in science fiction throughout the years, people giving orders to robots without using any special commands or keywords. These people simply talk to machines as they would talk to any other person.

This thesis describes the creation of a system that would allow robots to communicate with human operators more easily. In order to begin a platform had to be developed comprised of a collection of both hardware and software that would allow the robot to perform the actions required of it. In the simplest form this hardware configuration contains an iRobot create, Arduino micro-controller, web-cam and ASUS Eee PC. The Eee PC uses windows XP, MATLAB and C++. Using image processing in MATLAB the robot is able to identify objects that are in its environment. In order to interact with people the robot processes natural language input from the user. Language is parsed by linking words to concepts. A tree is stored in the robot's memory that relates concepts to one another. Each word can represent a number of concepts. To determine which concept a word represents, a list of requirements is kept for each concept. The concepts represented by the rest of the words in the sentence are used to determine the concept the user is referring to. Using concept and the representations of objects in images, words that represent objects can be linked to what those objects look like. This allows the robot to carry out tasks put to it. In the future, this has the possibility of being expanded to using more complex objects that aren't uniformly colored, or with maps of areas and coordinate tracking that would allow for more advanced movement.

Acknowledgements

I would like to express my thanks to all those who supported me while working on this project. The faculty members of the Computer Science Department have continually shown support for all endeavors that I have pursued through my years at Kutztown University, including but by no means limited to this project. I would also like to express sincere gratitude and the most genuine thanks I can give:

To my advisor, Dr. Oskars Rieksts, who renewed my interest in robotics that had waned some in my years following the ending of my high school. Without the robotics club that he and I worked on I do not know if I would have had the drive and interest to go this far.

To my entire family, especially my mother, Adele Minton, who even through the worst times always stood by me and helped me persevere through the hardest times of this project.

To Dave Diehl who teaches technology and engineering at Council Rock High School North. Without the technology club in high school I don’t know if I would have gone into computer science

I would also like to thank Kutztown University for the graduate assistantship that made graduate school possible

Table of Contents

List of Tables and Figures

Chapter 1

Introduction

It has been seen countless times in science fiction throughout the years, people giving orders to robots without using any special commands or keywords. These people simply talk to machines as they would talk to any other person. The technology seen in science fiction movies, books and television shows has often been a strong driving force for the computer industry and robotic communication is no different. Research has been done in the processing of natural language by computers for quite some time.

This thesis explores the creation of a dialog based robotic system through the use of a hardware and software platform and the combination of processing of images and interpreting natural language. There are many issues that are covered in order to explain the attempt to combine multiple systems that together will allow for this dialog based robotics system. What is presented acts both as a blueprint for future work as well as a framework consisting of those features from the blueprint that have been implemented.

This thesis first discusses the creation of a hardware and software platform that not only fits the needs of the dialog system but also could be used for other robotic solutions. Next the image processing techniques that were tested and finally employed will be explained. The third part explains how the author conceived of a

process of parsing English words into conceptual ideas. Finally a system will be explained whereby images of objects can be linked to the concepts defining those objects.

Chapter 2

Robotic Hardware and Software Systems

In order to create software for a robot, there must first be a robot. A fair bit of research and experimentation went into choosing what components the robot would be comprised of. Several prototypes were built and modified until the operation of the robot was satisfactory.

Chassis

The first iteration of a chassis was built using parts from VEX robotics kits. VEX kits allow designers to custom build a robotics chassis to suit their needs. This first robot had a three wheel design. Two drive wheels on the left and right were connected to separate motors and provided for all movement of the robot. A third wheel on the back was used to stabilize the robot. That wheel was what is known as an "omni wheel." These wheels have rollers around the circumference that are perpendicular to the rotation of the wheel which allows it to slide left and right to reduce friction when the robot is turning. The initial build of this chassis was to accommodate the restrictions put forth by the Trinity College Fire Fighting Home Robot Contest. This meant that the robot could not exceed certain dimensional limitations. The need to fit various components onto the chassis caused spacing problems. In addition the drive system was inaccurate and required a great deal of calibration whenever the battery that powered the motors died and needed replacing.

The second chassis iteration came about after the limitations placed on the robot by the competition were no longer a factor. The chassis chosen was the Create made by the iRobot company. This is the same company that produces the well-known Roomba robotic vacuum. The Create is meant specifically to be used as a chassis for robotics experimentation. It is essentially the same design as the Roomba but without the vacuum cleaner. Instead there is a cargo bay in place of the dirt collection area to allow for the storage of additional components. This chassis had many advantages over its predecessor. The Create contains an on board controller that is accessed via a serial port on the robot. The Create reacts to commands which are represented by strings of byte code that have been defined in the iRobot Create Open Interface. This allows an external computer to send commands to the drive system and receive information from the sensors of the Create.

Another advantage that the Create has is that the sensors and actuators are part of the construction. The wheels of the Create are connected to swing arms that let them rotate a short distance below the bottom of the robot. These arms contain sensors that can be used to check if a wheel has dropped, indicating some sort of operational error. Two bump sensors on the left and right front of the robot can let the robot know if it has impacted something. Each bump sensor can be polled independently so that the robot can determine on which side of center the impact occurred. Cliff sensors on the bottom of the robot can be used to ensure that the robot does not travel off any ledges. An IR receiver on the top of the robot can receive signals in 360 degrees. There are charging stations and virtual walls that send out IR

signals that the Create can sense and react to. Lastly there are many mounting points on the Create which allow a developer to secure additional instrumentation to the Create to increase its capabilities.

Sensors

The early stages of this project centered on combining multiple sensing systems in an effort to accurately map a robot's environment. Initially, an attempt was made to combine distance sensing data with visual cues found in a robot's environment to map out the relative locations of points of interest. Web-cams were to be used to gather image data and range finders were to provide the distance data necessary to make the mappings.

One of the main focuses of this endeavor was to research ways of allowing robots to more easily understand their environment. The most obvious way to do this was through the use of actual images of a robot's environment. The easiest way that was found to get images of the environment was to use web-cams connected to a computer via USB. The device controlling the robot could poll the camera for an image and analyze it based on certain rules and restrictions.

The first camera used was “Classic Silver” from the Hercules Corporation. This camera turned out to be ill equipped for the task. The left to right viewing angle of the camera was too small and couldn't take in enough of the environment for the robot to make accurate decisions. To avoid these shortcomings another camera was chosen, the Microsoft Life-cam VX6000. It has a 71 degree left to right viewing angle. This larger view lets the robot capture images that depict a wider area than the

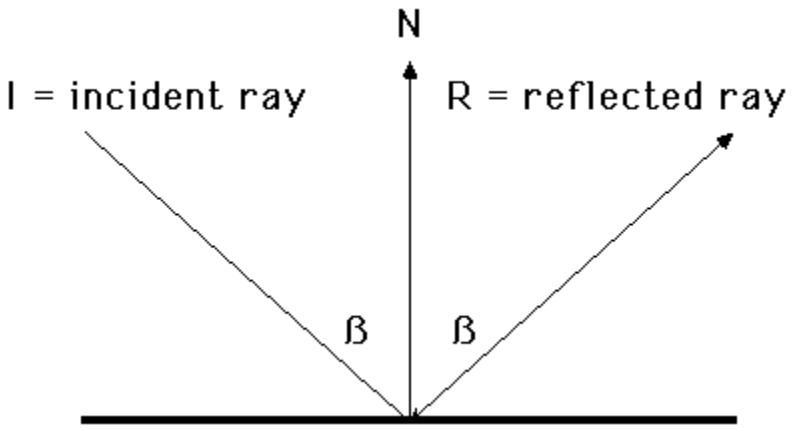
previous camera and allows for more environmental features to analyze.

To perform distance measurements it was necessary to equip the robot with some sort of range-finder. Sonar range-finders use pulses of ultrasonic sound to determine the distance to an object. The controller sends out a pulse of sound which will bounce off solid objects. The elapsed time between sending the pulse and then receiving the reflected signal is measured. The distance to that object can then be calculated since the speed of sound in normal earth atmosphere is known and relatively constant.

Sonar was first chosen because its operating range is very large. These range-finders will return accurate readings anywhere from 2cm (0.7 inches) to 3m (118.1 inches) from an object, affording a much larger range than IR rangefinders which will be discussed in more depth later. However, there was a moderate issue encountered while experimenting with sonar on the first iteration. When the sound pulse from a sonar sensor impacts an object, depending on how smooth the object is, the signal may experience what is called specular reflection (Flynn, 1988). This is the same thing that happens when a beam of light hits a mirror, the light bounces off at an angle equal but opposite the angle of incidence (See figure 2.1.)

Figure 2.1

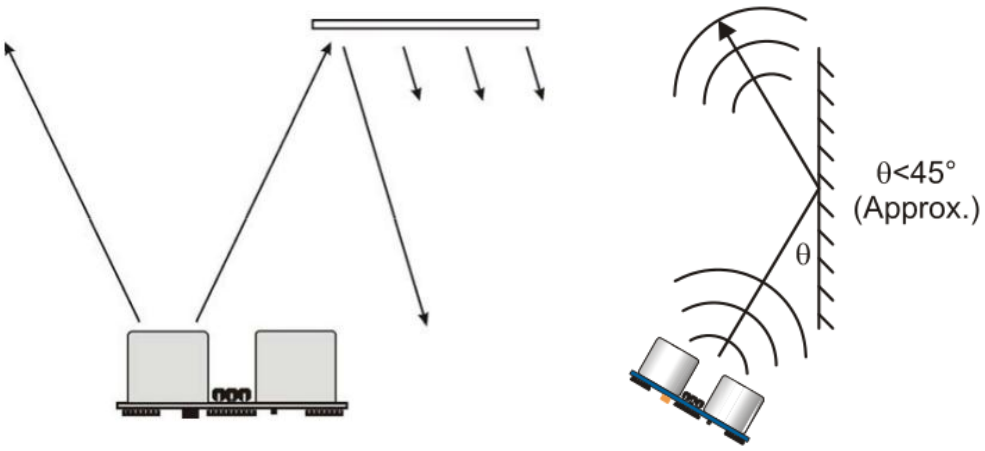
Example of specular reflection



This is means that the signal form the sonar will not always reflect directly back to the receiver. If any portion of the signal returns, there is a good chance that it has bounced off other objects and the distance value obtained is actually much greater than the true distance to the object of interest (See figure 2.2 for an example**.**)

Figure 2.2

Specular Reflection in sonar

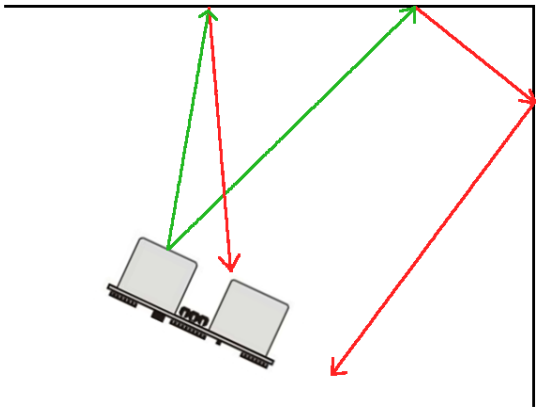


The previous figure shows two examples. One describes what happens when the

object being sensed is not in full view of the sensor. The other describes a situation in which specular reflection can occur.

Figure 2.3

Sonar sensor in a corner



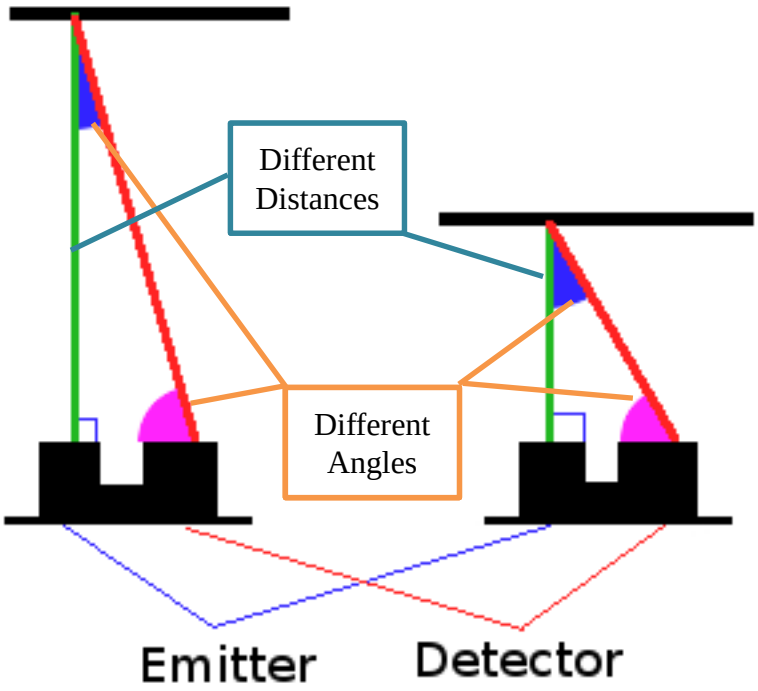
When the sonar sensor is used in a corner, the reflection of the beam causes different parts of the signal to be returned to the sensor at many different times. When this happens the distance readings from one moment to the next become extremely erratic. Due to this limitation it eventually became clear that sonar was not going to be an acceptable way to get distance data. Although sonar offered a very good distance range, there was no way to know that the robot would be facing an object perpendicularly and thus was getting valid readings.

Infrared range-finders use a similar principle to the sonar range-finders except that, instead of a pulse of sound, a beam of infrared light bounces off an object. These range-finders were not initially considered because their operating range is much smaller than that of sonar range-finders. An example of a fairly standard robotics IR range-finder is the Sharp GP2D12. The measurement from the Sharp series of

rangefinders is not based on the time it takes for the light pulse to return. Instead, the receiving side of the rangefinder uses a linear IR detector to determine the angle between the straight line of the emitted IR light beam and the line that follows the path of the light reflected back to the receiver. Knowing this angle and the distance between the emitter and receiver allow the sensor to determine the distance to the object that is reflecting the IR light. According to the Sharp rangefinder specifications this method of distance measurement is called triangulation; an example can be seen bellow.

Figure 2.4

Distance sensing using IR rangefinder and triangulation



Due to the fact that the receiving portion of the sensor must be able to focus on the point of light produced by the emitter, there is a much smaller area in which IR

distance reading are valid. The GP2D12 only has an effective range from 10cm (4 inches) to 80cm (30 inches). It was thought initially that this would be too small for the researcher's purposes. However, after working with the robot for quite a while it was found that distance data was only needed when the robot was fairly close to an object. This is because the web-cam provides sufficient information to navigate among objects so the IR sensor need only be used when the robot determines, using the web-cam, that the objects are very close and can cause a navigation problem.

IR range-finders have a definite advantage over sonar as there is much less full reflection in the pulsed signal. Even when striking an object at an oblique angle, enough of the light beam is reflected back to the sensor for a valid distance reading to be taken. Additionally, the point of light that is produced by the IR emitter is much smaller and more directed than pulse of sound produced by the emitter on the ultrasonic rangefinder. This reduces the scattering that is seen when sonar is used in corners. A version of Sharp range-finder was found which, through the use of different focusing optics, allows for a greater maximum ranging distance which fits the robot's needs very well.

Controllers

The first prototype robot that was built was simply a collection of disparate parts. A device was needed to control the various parts of the robot. The Create does contain a small microprocessor (it must in order to receive and interpret byte code through its serial interface) but the processing power of this controller is very minimal, and it is not possible to place a custom program on the Create's internal

controller. These issues meant that a hardware controller needs to be used to organize the robot's operation and interact with its sensors and actuators

A major part of this experiment is the analysis of images. This requires quite a bit of processing power, far more than any microprocessor contains. For this reason the conclusion was reached that an actual computer would be needed. The term “net-book” is a name for a specific subset of laptop or notebook computers, which generally have a screen size of ten inches or smaller. These computers, although they generally have less powerful processors than their larger counterparts, still are powerful enough to process images. The net-book chosen was the ASUS Eee PC 900HA. This model was chosen specifically to fit with the first version of the robot. The 900HA has a screen measuring 8.9 inches diagonally so it was small enough to fit inside the structure of the chassis.

The Arduino is a micro-controller built around Atmel processors. The Arduino version used is called the Arduino Duemilanove and it can use as its processor either the Atmel Atmega168, which was used in the final implementation of the project, or the Atmel ATmega328. The ATmega168 features a 16 MHz processor, 16 KB of flash memory, 1 KB of static ram and a 512 B EEPROM. These boards are extremely popular among electronic and robotic hobbyists due to their open source hardware interface and low price point of thirty dollars.

The Arduino website (2011) states the following:

> It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz crystal oscillator, a USB connection, a power jack,

> an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started.
>
> (Cuartielles, Igoe & Mellis, 2011)

Additionally there is an internal USB to RS-232 Serial converter so the USB connection can be used as a serial device on the computer.

Putting it Together

To make all of these components work together certain connections had to be made. First, while the Arduino is able to control the individual parts of the robot, it is unable to process the image data from the camera. Second, by itself, the computer cannot control the components of the robot. It is necessary for the Arduino and computer to communicate with one another to coordinate the operation of the robot. To do this the serial interface of the Arduino was utilized. For the first iteration a piece of software was written for the Arduino which accepts ASCII commands. The commands consist of a set of characters. Each character further specifies the desired command. For example consider the command *MF*. *M* tells the Arduino that it must move somewhere, and the next character, *F*, sets the movement direction to forward. Other commands set and get the current speed of each motor individually and also request readings from various sensors. Data from the Arduino is sent back along the same serial connection. This essentially turned the Arduino into a standard peripheral that the computer could use. This made it so that the computer was acting as the sole controller of the entire robot. All data is processed by and all decisions are made by

the net-book.

Figure 2.5

The robot used in the first iteration, viewed from the front

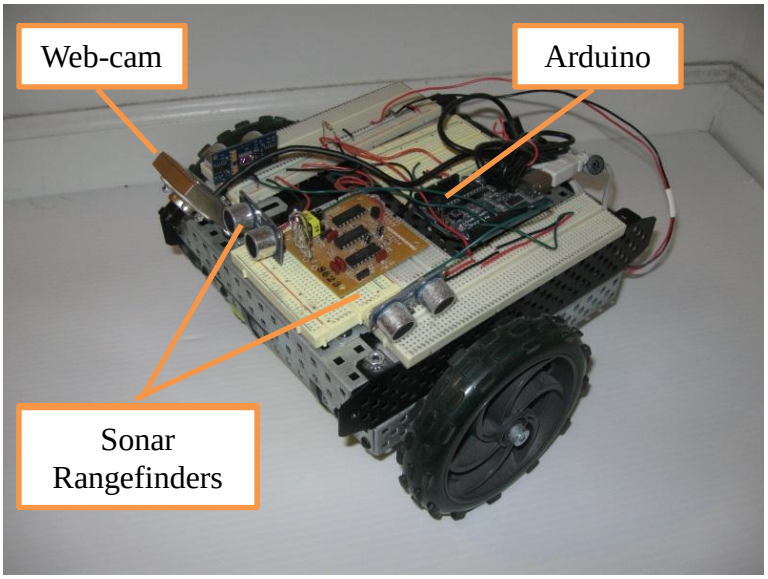


Figure 2.6

The robot used in the first iteration, viewed from behind

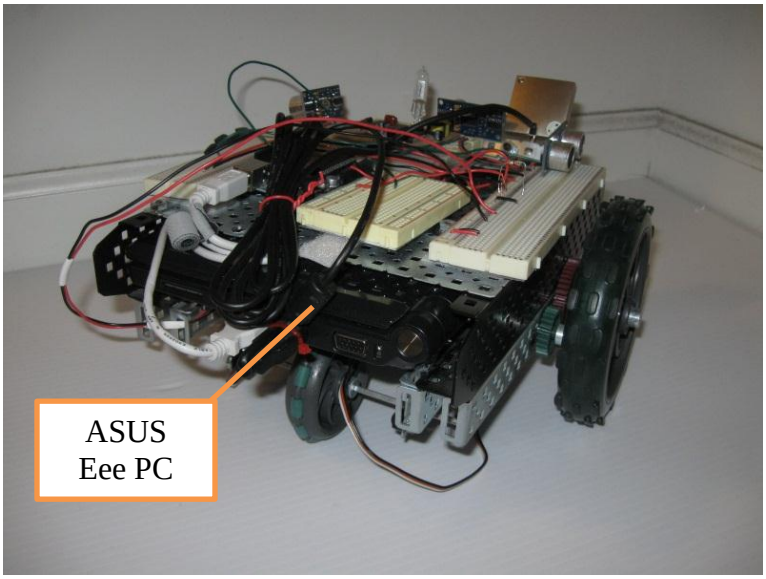
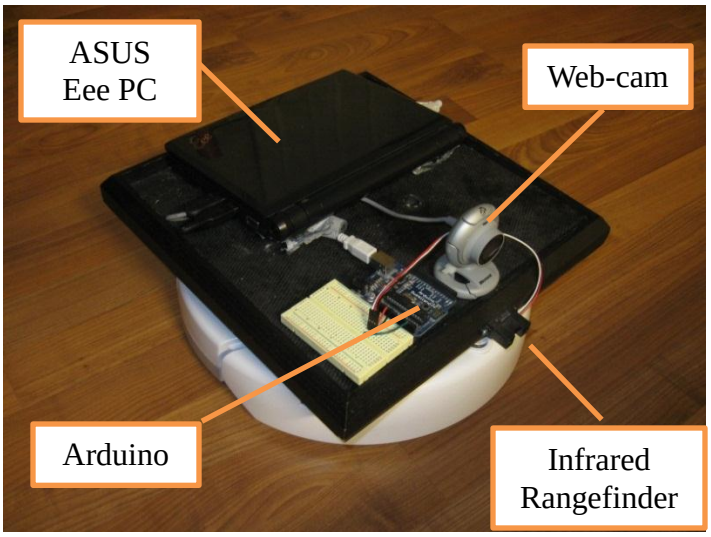


Figure 2.7

The robot used in the second iteration



Figures 2.xx, 2.yy and 2.zz show the layout of the robot's platform in both iterations of its development. In addition to those improvements already mentioned, the second platform greatly reduced what can only be call "clutter" that existed on the first iteration. Since all sensors and actuators on the first robot (except for the camera) had to be driven through the Arduino, this meant that connections had to be made from the Arduino to all motors and sensors on the robot. There also needed to be connections from the battery, which is stored under the chassis, to the motors. As can be seen in figure 2.5, there were many separate wires that had to be used to make all of these connections. The Create contains its motors, battery and all sensing equipment internally. This reduces the number of wires needed to three USB cables to connect the Eee PC to the camera, Arduino and Create. The only additional wiring that would need to be done is to make connections from the Arduino to any additional sensors that are placed on the robot in the future.

Chapter 3

Image Processing

The phrase “a picture is worth a thousand words” actually contains quite a bit of truth to it. People can look at an image and very thoroughly perceive what is in a scene. Quite a bit of work has been done in order to allow computers to use images to gather information about a scene or an environment. This is easier said than done. Significant amounts of processing must be performed in order for the data stored in images to become useful to a computer.

Computer Vision

The phrase “computer vision” describes a collection of ideas that explain the ways in which programmers are attempting to imbue computers with an ability to “see” things in their environment. Computer vision is not by any means a new concept.

Images in Computers

When a person sees a ball in the middle of the floor, they know immediately many facts about the ball. They can see what color it is. They can judge the size of the ball which gives them an idea as to whether or not they could pick it up with one hand. And they can judge the distance between themselves and the ball. Most importantly, they know for a fact that it is indeed a ball. This knowledge comes from years of learning about what objects are and how to interact with them. When a

computer equipped with a camera captures an image of a room with a ball in it, all that is seen is a collection of numbers. Images in computers are simply collections of values that define the colors of the pixels in an image. This means that without further processing, there is no way for a computer to understand what is in an image or what that image may represent. This is the biggest problem that computer vision aims to tackle.

Colors

There are many ways that the color of a pixel in an image can be represented. The most common method is to use the RGB (red, green, blue) color space. According to the CIE (Commission Internationale de l´Eclairage or International Commission on Illumination), a color space is a geometric representation of colors in space, usually of three dimensions (Buckley, Süsstrunk & Swen 1999). In the RGB space, each pixel has three values associated with it. These values represent the intensity of red, green and blue light. The values can range from 0, which means no intensity, to 255, meaning full intensity. Intensities can be and often are represented as a string of hexadecimal digits. For example in the string of hexadecimal digits *0x0054FF* the digits *00*, *54*, and *FF* represent the intensities of red, green, and blue respectively. Figure 3.1 shows the iRobot Create passing between two red cups that were identified using color matching.

Figure 3.1

iRobot Create passing between two red cups

OpenCV

The first image processing package that was approached in the course of this research was the OpenCV package of image processing tools.

The official OpenCV wiki (2011) states the following:

> OpenCV (Open Source Computer Vision) is a library of programming functions for real time computer vision. OpenCV is released under a BSD license, it is free for both academic and commercial use. It was originally written in C but has a full C++ interface and all new development is in C++. There is also a full Python interface. The library has >2000 optimized

algorithms (OpenCVWiki)

OpenCV showed very good potential early on in the research, specifically in the first iteration of the project. However, due to the limitations of the camera that were discovered, further development halted, including development using OpenCV and the camera in the first iteration was largely abandoned.

MATLAB

When work first started on the second iteration of the robot, a need arose to find a way to control the platform that had been developed on the Create. Initially the robot was controlled using a Python library built by Zachary Dodds of Harvey Mudd College. (Dodds, 2007) Python initially seemed to be a good choice as, was mention above, bindings already exist that let Python use OpenCV natively. The issue that arose with this implementation is that Python is an interpreted language, with inefficient memory access. This causes image processing operations to take much longer than they had using OpenCV in C++. At this point a switch could have been made to return to using C++ for programming. In the meantime, however, John Spletzer a professor at Lehigh University who has done a large amount of work in robotics and computer vision was contacted. At his recommendation, MATLAB was chosen to control the robot and to process images. MATLAB is a high-level language and interactive environment that enables one to perform computationally intensive tasks faster than with traditional programming languages such as C, C++, and FORTRAN. (Mathworks Inc., The, 2011) MATLAB can be expanded through the addition of what are called toolboxes. MATLAB toolboxes are functions or

collections of functions that enable a developer to perform certain tasks. MATLAB's Image Processing Toolbox offers the ability to perform operations on images that rival those of OpenCV. MATLAB's Image Acquisition Toolbox allows for the capturing of images from a connected web-cam. Another toolbox used was one designed to control the iRobot Create. The toolbox was developed by John Spletzer, Associate Professor in the Computer Science and Engineering Department at Lehigh University, in conjunction with students working in the VADER Laboratory at Lehigh University. The fact still remains that like Python, MATLAB is an interpreted language. The advantage of MATLAB lies in its purpose, which is the processing of data stored in matrices. This gives MATLAB for image processing a significant edge over Python, as images in computers are nothing more than matrices in which each cell contains the color values for a particular pixel.

Object Recognition

An image itself is useless to a computer. The image is stored in the computer's memory as a collection of numbers which define the color values of each pixel of the image. In the earlier example we considered of a person seeing a ball in a room, the person can see that ball and recognize what it is even though there are countless other objects in the image. A computer program however cannot, without specialized processing, distinguish the difference between the pixels that make up the ball from any other pixels in the image. The program must determine which discrete pixels make up the ball and which pixels can be ignored.

Blob Extraction

Blob extraction, or blobbing, is a process by which an image is separated into groups of pixels that share some common features. (Shapiro & Stockman, 2001) This is also called connected component extraction because blobs are not only composed of similar objects but all objects in a blob are somehow connected to the rest of the blob.

Figure 3.2

Pseudo-code for image blobbing algorithm

```
function BLOB_IMG(img, threshold) returns regions2pix:
        inputs: img – the image to blob, a 2 dimmensional matrix. Top left (0, 0),
                        bottom right (total height, total width)
                threshold – threshold value for use in determining if pixels belong in
                        the same blob
        returns: regions2pix – Hash map data structure that maps a segment
                        representative string to a list of strings representing all the
                        coordinates in the image belonging to that set

        uf ← union/find data structure of strings representing pixel coordinates
        west ← [-1, 0] /*used to find pixel to the north of current pixel of interest*/
        north ← [0, -1] /*used to find pixel to the west of current pixel of interest*/
        width ← WIDTH(img) /*get total number of pixels in x direction*/
        height ← HEIGHT(img) /*get total number of pixels in y direction*/

        for x = 0 to height do:
                for y = 0 to width do:
                        currPix ← [x, y]
                        pixWest ← currPix + west
                        pixNorth ← currPix + north
                        uf.CREATE(MAT2STR(currPix))
                        if (MEAN(pixWest) – threshold < MEAN(currPix) <
                                MEAN(pixWest) + threshold) :
                                if uf.FIND(MAT2STR(pixWest) != emptyMatrix:
                                        uf.UNION(MAT2STR(pixWest),
                                                MAT2STR(currPix))
                        if (MEAN(pixNorth) – threshold < MEAN(currPix) <
                                MEAN(pixNorth) + threshold) :
```

```
                        if uf.FIND(MAT2STR(pixNorth) != emptyMatrix:
                            uf.UNION(MAT2STR(pixNorth),
                                MAT2STR(currPix))
    for x = 0 to height do:
        for y = 0 to width do:
            currPix ← MAT2STR([x, y])
            repStr ← uf.FIND(currPix)
            regions2pix[repStr].ADDELEM(currPix)

    return regions2pix
```

The major key to blob extraction is the disjoint-set data structure. This is used to store lists of the pixels belonging to each blob. The data to be stored must be in a form that can be hashed. To accommodate this each pixel was stored using a string representing its coordinates in the image. Each list is uniquely identified by the lowest value label in the set. This structure contains three functions. The *find* function gives the set identifier for the set to which a given item belongs. The *create* function creates a new list containing the item given to the function. Lastly the *union* or *join* function combines two given sets into one. The identifier of the resulting set will be the lower of the two initial set identifiers. The *union* and *find* functions are designed to be very efficient as there will be many operations performed on the structure where data is retrieved, searched and moved while the algorithm is running.

Figure 3.3

Sample image consisting of 25 pixels arranged in a 5 by 5 matrix

Let us consider an example consisting of a 5 pixel by 5 pixel image which can be seen above in figure 3.2. The table below shows a 5 by 5 table in which each cell contains the color values [blue, green, red] of the corresponding pixel in figure 3.2.

Table 3.1

Five by five matrix representing the image in figure 3.2

(x, y)	**1**	**2**	**3**	**4**	**5**
1	[120, 111, 82]	[118, 109, 80]	[117, 108, 79]	[50, 70, 79]	[51, 71, 81]
2	[121, 112, 83]	[120, 111, 82]	[51, 70, 80]	[50, 72, 81]	[49, 69, 82]
3	[121, 112, 83]	[50, 111, 201]	[50, 109, 200]	[51, 115, 204]	[67, 200, 100]
4	[200, 50, 90]	[225, 104, 67]	[50, 115, 202]	[73, 112, 201]	[72, 100, 203]
5	[201, 70, 100]	[100, 234, 80]	[103, 231, 79]	[99, 229, 78]	[98, 228, 77]

This data is iterated through once, assigning each pixel to a blob. The algorithm described in figure 3.1 compares each pixel with the values of its northern and western neighbors. If a the pixel matches either of these then its coordinate string is

added to the list that the matching neighbor belongs to and the pixel is given a label representing the set it was placed in. If both neighbors match but they belong to different sets, the sets are joined using the *union* function and the label given to the pixel is the minimum of the labels of each neighbor.

Table 3.2

Blob assignment after the first pass over data in table 3.1

(x, y)	1	2	3	4	5
1	0	0	0	1	1
2	0	0	2	1	1
3	0	3	3	3	4
4	5	6	3	7	7
5	8	9	9	9	9

Table 3.3

Blob assignment after the second pass over data in table 3.1

(x, y)	1	2	3	4	5
1	0	0	0	1	1
2	0	0	1	1	1
3	0	3	3	3	4
4	5	6	3	7	7
5	8	9	9	9	9

The second iteration over the image data changes the labels of any pixel if the set they belong to was joined with any other set on the first iteration. In the above example only the label for pixel (3, 2) was changed. On the first pass this pixel did not belong to the blob that its northern and western neighbors belonged to so it was assigned to a new blob. When pixel (4, 2) was checked it was seen to match both the northern neighbor and pixel (3, 2). The sets of these two neighbors were then joined and pixel

(4, 2) was given the label 1. During the second pass the label for pixel (3, 2) is changed to match the label representing the set it now belongs to.

Initially blobbing seemed as though it would be a valid method for detecting objects in an image. Images were blobbed by setting a threshold value for the red, blue and green (RGB) values for each pixel. As long as those values were not too greatly different from those of the adjacent pixels, the pixel was added to the same group as the similar adjacent pixels.

The first problem discovered with the blobbing approach was in the shapes of the blob regions. This method relied heavily on the premise that an object would be fairly uniform in color. This is not the case. Even a ball that is red has many different shades across its surface due to the way light interacts with it. Since the threshold was simply a number, there was no way to represent this complexity. If the value was set higher so that an entire object would be detected, the detection area would invariably encompass areas around the object as well, distorting the object's shape and the object's average color values. If the value was set low, only portions of the object were detected. Although this allowed for better average color calculation, it still distorted the overall shape of the object. In order to allow a robot to recognize an object it is necessary to have in hand the shape and average color across the object in order to identify it and store its signature for later recognition.

Clustering

Clustering is quite similar to blobbing. Using a processing algorithm, a set of data can be split into some number of clusters. Unlike blobbing, clustering consists of

a statistical analysis process which can be applied to any kind of data.

K-Means

In searching the MATLAB knowledge base for reliable ways to segment images and recognize objects, a method was found that uses a statistical method known as K-Means clustering to split images into sets of related pixels. The K-Means algorithm splits a set of data into K clusters. (Image Processing Toolbox: Color-Based Segmentation Using K-Means Clustering)

Figure 3.4

Pseudo-code for K-Means algorithm (Ahmed, 2010)

```
function KMEANS(data_vecs, numSegments) returns data_idxs, centroids:
        inputs: data_vecs – vector of data to be segmented /*must be less then 3
                                dimensions*/
                numSegments – the number of segments the data stored in data_vecs
                    should be separated into

        centroids ← numSegments values evenly spaced between and including min
            of data_vecs and max of data_vecs
        prev_centroids ← centroids offset by 1 /*guarantess that prev_centroids
            and centroids are not equal at start of first pass over data_vecs*/
        data_idxs ← empty 1 dimensional vector of length LENGTH(data_vecs)
        while prev_centroids ≠ centroids do:
            prev_centroids ← centroids
            dists ← empty vector with dimensions
                                LENGTH(data_vecs) by numSegments
            for j = 1 to numSegments do:
                for k = 1 to LENGTH(data_vecs) do:
                                temp ← data_vecs[k] – centroids[j]
                                dists[j, k] ← SUM_OF_SQUARES(temp)
            for j = 1 to LENGTH(data_idxs) do:
                data_idxs[j] ← IDX_OF_MIN(dists[j])
            for j = 1 to numSegments do:
                centroids[j] ←
                                MEAN(data_vecs[x] for all x where data_idxs[x] == j)
        return data_idxs, centroids
```

```
function SUM_OF_SQUARES(vals) returns sum_sq:
        inputs: vals – collection of values to get sum of squares for
        returns: sum_sq – the sum of the squares of values in vals
        for j = 1 to LENGTH(vals) do:
            sq ← vals[j] ^ 2
            sum_sq ← sum_sq + sq
        return sum_sq
```

Table 3.4

Represents table 3.1 reshaped into a 2 dimensional array

Pixel Num	Red Channel	Green Channel	Blue Channel	Row	Column
1	120	111	82	1	1
2	121	112	83	2	1
3	121	112	83	3	1
4	200	50	90	4	1
5	201	70	100	5	1
6	118	109	80	1	2
7	120	111	82	2	2
8	50	111	201	3	2
9	225	104	67	4	2
10	100	234	80	5	2
11	117	108	79	1	3
12	51	70	80	2	3
…..	……	…..	….	….	….
23	67	200	100	3	5
34	72	100	203	4	5
25	98	228	77	5	5

In order to process the image using the k-means algorithm it needs to be reshaped into a 2 dimensional matrix. Table 3.4 still contains all the information from table 3.3 but is in 2 dimensions instead of 3. Each row in table 3.4 contains the information for one pixel in the image. The first three fields contain the color data for the pixel and the last two contain the coordinates of where the pixel was in table 3.3.

Table 3.5

The minimum and maximum values of each column from the data in the matrix in table 3.4

Value	Red Channel	Green Channel	Blue Channel	Row	Column
MIN	49	50	67	1	1
MAX	225	234	204	5	5

Table 3.6

The initial centroids for the clusters when 6 is given as the number of clusters to find

Centroid	Red Channel	Green Channel	Blue Channel	Row	Column
1	49	50	67	1	1
2	84.2	86.8	94.4	1.8	1.8
3	119.4	123.6	121.8	2.6	2.6
4	154.6	160.4	149.2	3.4	3.4
5	189.8	197.2	176.6	4.2	4.2
6	225	234	204	5	5

Since clusters are built around central mean values, there needs to be an initial set of centroids to use for the first pass over the data set. The min and max values from table 3.5 are used for two of the centroids. The remaining k-2 centroids are generated to be evenly spaced between the min and max.

Table 3.7.

Cluster index matrix created after one pass over data. Each row represents the cluster that the pixel in the corresponding row in table 3.4 belongs to.

Pixel	Cluster Number
1	3
2	3
3	3
….	….
10	4
11	2
12	1
13	3
14	3
15	4
16	1
….	….
23	3
24	3
25	4

The first test of using the k-means algorithm showed exceptional results, beyond anything earlier blobbing techniques had shown. The blob extraction techniques initially used only cared if adjacent pixels where similar to each other. This often allowed for blobs that were either too large, which expanded beyond objects, or were too small, only including a portion of an object. The threshold chosen would determine which of these two outcomes would result. The k-means on the other hand considers the similarity between a pixel and the mean of an entire cluster. Experimentation was done where different values for k were used in addition to using different color spaces to represent images. Two of the color spaces that were tested were Lab which actually refers to the L*a*b* color space defined by CIE and YCbCr.

In both spaces, L and Y represent the luminosity or brightness and Cb, Cr, a*, and b* represent intensities of colors. Examples of these color spaces can be seen in figures 3.5 and 3.6. These experiments showed that applying k-means to the data in an image could reliably return sets of data containing only a specific object in an image. The figures bellow show results from k-means on an image when using different *k* values

Figure 3.5

Image that is to be segmented

Figure 3.6

Segment that includes ball when figure 3.5 is segmented using k = 5

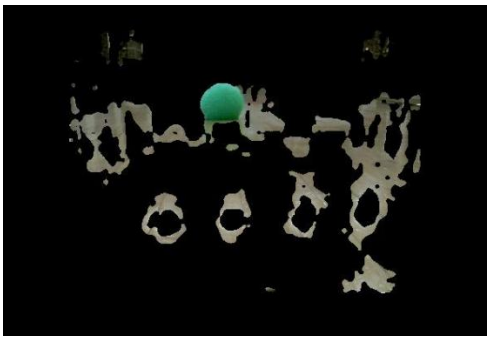
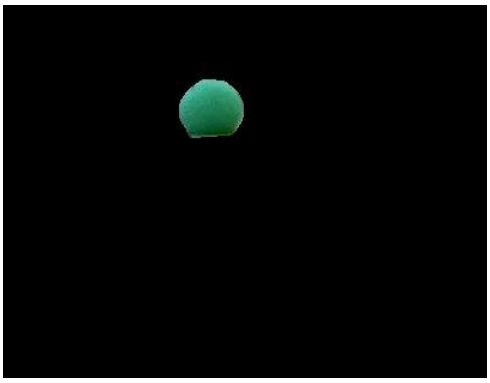


Figure 3.7

Segment that includes ball when figure 3.5 is segmented using k = 6



Chapter 4

Natural Language Processing

"Go to the red ball." This is a simple English sentence that can probably be understood by any two year old child, but when it is stored digitally on a computer it appears to the software to be a collection of seemingly random bits. In order for humans to properly communicate with robots it is necessary to create software that is able to not only read the sentence those bits represent, but also to discern the meaning the user is conveying with the sentence.

Interpret Language

In order to determine the meaning behind a sentence, the meaning behind each word in that sentence must be ascertained. This could be done by creating sets of rules for each word in a sentence that defines actions or modifies those actions. The English language however contains an immense number of different words. There are enough that it wouldn't be feasible to create a rule for each word. This large problem space can be reduced by using the fact that many words actually represent the same idea and some words just represent different tenses of other words. This leads to the possibility of mapping many different words to one central concept or idea. This process is widely referred to as conceptual parsing. This allows rules to be written for a much smaller and more feasible set of concepts and many different words can be mapped to them when a sentence is parsed.

Conceptual Parsing

In creating a conceptual parser the researcher first constructed a small collection of concepts that words would map to. It became evident that more than a simple list of concepts was required. For example, the idea of *movement* can refer to many different things. For a robot, *movement* can refer to moving an actuator arm or traveling from one location to another. The best way to represent such interrelated concepts was to construct a tree. (Gurevych, Malaka, Porzel & Zorn, 2003) The farther down the branches of the tree one travels, the more concrete the concepts become, until the concept is as concrete as can be defined for the purpose of this system.

Having such a tree of concepts is not sufficient for understanding. There still needs to be a way for the robot to map words to these concepts. This could be done through the use of a hash map where each word would point to a leaf on the tree that contains the concept the word is referring to. However, that would only work if each word pointed to exactly one concept. As stated above this is not the case. Each word can represent many different concepts. (Roos, Vogt & Wiesman, 2002) In order to accommodate this fact it is necessary to create a list of concepts that each word points to in the hash map. This leads to the central issue relating to conceptual parsing. How can the desired concept referred to by the use of a word in a sentence be discerned? This is accomplished by pointing each concept leaf to the leaves of other concepts that are required for conceptual completeness. Each concept contains a collection of one or more such lists. Each one of these lists represents a collection of other

concepts that must also be present in the sentence in order for the concept to be valid. (Simmons, 1970)

This works as follows. The robot is given a command that contains the word "go." The hash map is checked and the first concept in the list that "go" points to is a reference to the *travel* concept, which is a child of the *movement* concept. The *travel* concept contains three separate collections of required concepts. The first contains just the *place* concept. The second contains both the *distance* and *direction* concept. The third contains *time* and *direction*. At this point the rest of the sentence must be considered.

Continuing with this example, let the sentence also contain the word "meters." This word is easily mapped as, in this context, it can only refer to a distance measurement. This lets the robot know that the desired meaning of the word "go" is a command to move a specified distance. That being the case, the sentence is then searched for a *direction*. In order to determine how many meters the robot is commanded to move, the *distance* concept also contains a list of requirements. One of these requirements is the concept of a *value*, a *number* in this case. If any requirement of one of the lists is missing, then the other concept lists are checked.

Consider now the example sentence presented at the start of this chapter. "Go to the red ball." This sentence contains the word "to." Being a preposition, this word works to connect two parts of a sentence. When this kind of word is encountered, the robot should then look at what comes before the preposition, "go," and what comes after it which in this example is "the red ball." The word "the" can often be ignored

from a conceptual parse as it does nothing to further define any objects or places. Through further processing "ball" parses to *object* and "red" parses to *color* this is used to classify *object*. If, on the other hand, none of the other requirement lists match, then the most complete list is chosen and the robot can query the user explaining that there is a problem with the sentence and the user may then make any adjustments necessary.

The Process of Thought

The method of parsing concepts used in the course of this research was influenced largely by the work of Jerry A. Fodor as presented in The Language of Thought. In the beginning of the first chapter Fodor sets out to show the validity of a set of arguments. These arguments deal with theories of his that together explain various aspects of the human thought process and the way humans choose what concepts relate to each other. Fodor proposes several theories that this researcher used as a basis for constructing a language to knowledge mapping.

Fodor first states that, "The only psychological models of cognitive processes that seem remotely plausible represent such processes as computational." (Fodor, 1978, p.27) He is making the statement that the process of thought is by nature one of computation. The brain weighs the accuracy of concepts as they describe external objects or language. His second point is that "Computation presupposes a medium of computation: a representational system." (Fodor, 1978, p.27) Here he is making the point that if cognition is a computational process then there must be a medium in which to carry out these computations. This medium, he says, is a representational

system. A representational system is a common theme in cognitive processes. It explains that no matter what the situation, people use internal conceptual representations for the real world objects they are interacting with. Lastly he states that “remotely plausible theories are better than no theories at all.” (Fodor, 1978, p.27) One way this can be interpreted is that having a general idea of what concept is being presented is better than having no idea at all. It is also possible that this can lead to a process in which assumptions that are made can be later proven and thus new concepts are learned using assumed relations to known concepts.

These statements made by Fodor led the researcher to the thought of creating a web of concepts as a form of representational system. This web would allow external objects to be represented in software and their relationships computed. Also it was thought that it is not necessary to understand exactly the concept that a word represents. Instead, all that is needed is an assumption of the best fitting concept as it would provide more information than no concept at all. When all of these ideas fall into place, a method of not only thinking but also learning comes to light.

Chapter 5

Objects and Concepts

Until this point, computer vision and natural language processing have been discussed as two separate systems. In order for a dialog based interaction system to work these two separate concepts must be connected so that the objects and concepts in one system are able to refer to those in the other.

Language Referents

In order to properly communicate, two entities must share a system of referents. This need was explored in the work of Amichi Kronfeld, more specifically his essay entitled Reference and Computation: An Essay in Applied Philosophy of Language. This essay discusses the relating of objects that people refer to in language (Kronfeld, 1990). When people are speaking to each other there are quite often a number of ambiguities present. The task of filtering these ambiguities and determining what they are referring to does not cause concern for the people in the conversation. Healthy human minds are able to be presented with a referent in language and determine the concept or object in reality that is being referred to without any confusion. Take the following example. Bob and his wife Margaret are at a party.

Bob: “Harry looks like he and his wife are having a good time at the party.

Margaret: “Sally is not Harry’s wife!”

Bob: “Oh, interesting.”

Most likely this interchange will seem perfectly normal to anyone that reads this thesis. However the only reason for that is because humans have an ability to understand the sharing of referenced objects. Most people reading that interchange will infer that the situation resembles the following description. Bob and Margaret are together in one part of the room. Harry is some distance away and he is standing with a woman. Margaret knows that the woman near Harry is not his wife. However, Margaret is able to infer that the referent in Bob’s statement is the woman standing near Harry, even though she is not his wife. She does not need to ask Bob to clarify to whom he is referring; she immediately knows that Bob has mistaken this woman to be the wife of Harry. Additionally, when Margaret refers to the woman as Sally, even though it is possible that Bob never knew the woman’s name, Bob also is able to immediately understand that the woman he was internalizing as Harry’s wife is actually named Sally and also that she is in fact not Harry’s wife. He has no need to ask who Sally is; he is able to understand that he and his wife are referring to the same person. In this situation the objects and people being referred to are given different names but no problem arises with the communication as each person is able to infer who or what the referent is.

One way that this human interaction can be explained is through a concept called the KLO triad. (Minton & Rieksts, 2011) KLO refers to the interaction between language (L), the object being referred to (O), and the knowledge base (K) that an agent is using. This concept can be explained using the above example of Bob and

Margaret's conversation. When Bob speaks to Margaret he is referring to the object that is now known to be called *Sally*. Bob needed to use some form of language to express his ideas to Margaret. This is where the knowledge base comes into the process. The knowledge base is a conceptual system where agents can internalize what external objects mean to them. This includes their understanding of what objects are and the different ideas that parts of language can describe.

When Bob attempts to describe the relationship between Sally (whose identity he does not truly know) and Harry, the most realistic and accurate way his brain can determine to represent this woman who is interacting with Harry is by reasoning that she must be his wife. This causes the connection between the object that is Sally and the language that describes her as wife. This connection that is made is simply a result of the stimulus that Bob has encountered up to this point.

Similarly when Margaret hears what her husband has said she must then relate the language from Bob to an object in reality using her own knowledge base. When she hears Bob refer to Harry's wife, an examination of her knowledge is able to produce that object that she has related to the concept *Harry's wife*, whom she in fact does know. Unlike Bob, Margaret already has a distinct idea of the object that the concept *Harry's wife* relates to. This then causes her to try to locate this object in her surroundings.

When she is unable to locate the object that she knows to be Harry's wife she must reevaluate the statement from Bob. She is able to look again at the idea *Harry's wife*. Now she can focus on the idea *wife* and fill that concept with what a wife is.

She knows that it would be a woman who is most likely married to a man and that it is probable that the man and woman would be near each other in strictly special terms and that these two people would also most likely be interacting in some way with each other. She then is able to observe the interaction between Harry and Sally. This finally allows her to reassign the concept *wife* in Bob's statement to actually be referring to Sally. Margaret knows that Sally is not Harry's wife and is then able to respond to Bob and in doing so promptly corrects his error.

This entire process of interpretation, relation, and expression is carried out seemingly instantly. As human beings, Bob and Margaret are using knowledge bases that have been added to and refined for many years. The fact that Bob and Margaret don't share the same knowledge base does not affect their ability to communicate. The human mind is able to infer new relationships inside the knowledge base and learn from its surroundings. The KLO triad itself acts to normalize a person's interaction with their world and allows all part of the triad grow and evolve as the person learns and interacts with new things.

Establishing Referents

In this system of KLO triads and referent matching, there are at least two different methods by which the mind can attain new information that can be used to expand its knowledge base - intensional learning and extensional learning. Intensional learning is the process used by Bob and Margaret in the example above. A concept is learned by attempting to understand the internal meaning of the information portrayed to a person. This requires the learner to already have some data that relates to the

concept they are trying to learn. In the case of Bob he was able to use the relation to the woman that was with Harry and correct his misunderstanding that she was Harry's wife by adding the fact that her name is Sally and that she is in fact not Harry's wife at all. This he does without Margaret explicitly telling him that he was wrong. Instead simply the intensional import of her statement allowed him to affect a change in his knowledge base.

In contrast to intensional learning, extensional learning involves a much more explicit interaction between the learner and their surrounding environment. If Margaret were attempting to extensionally inform Bob of the identity of Sally and her relation to Harry she would have had to point her out among the other people in the room and explicitly say that the woman she was pointing to was named Sally and that she was not Harry's wife. This type of learning is often seen when someone is attempting to explain what an object is to a young child. They pick up the object and show it to the child or point at the object and at the same time they explain what the object is called and possibly try to describe what the object is used for and how it works, if applicable. Throughout life people continue to learn through a combination of both of these methods. Both methods are also necessary as some complex ideas and concepts cannot be easily inferred from relations with known concepts. Also it would be difficult for people if they had to be explained everything in an extensional way throughout their life. If this was the case then knowledge would never be able to expand and the ability for someone to discover a concept that has not yet been discovered would be impossible.

Artificial Communication

The concept of the KLO triad is an excellent way to create a system where humans could communicate with computing systems. The language and object in the triad are the simplest items to create analogs for. The input of external objects is simply that data which is taken in through the robot's camera. The external language is simply what information is relayed to the robot from an external presenter or produced from the robot for an interacting agent to take in. The knowledge base can then be seen as being the most complex of the pieces that must be created. This is where all the processes discussed earlier such as computer vision, object detection in images and natural language processing come to the fore. These systems must come together within the robot's software and work together to allow the robot to accurately carry out any tasks it is set to.

The tasks involved with mapping words to the concepts they represent have been discussed earlier. However, only the mapping of language to concepts was covered, not relating those concepts to objects in images. There were processes described that can allow software to extract discrete objects in images. Again, though, this only deals with objects that are in images. In order to be of use in creating a KLO triad and facilitating communication there must be some way for the software to understand what the word ball refers to. Likewise, the software must be able to understand that when a ball is extracted from an image that there are concepts that can describe it. In order to attain these internal relations, both of the previous systems that were discussed must be expanded upon and given more interoperability with each

other.

Classifying Objects

Earlier, the ability to attain discrete objects from images was explained to be desirable. The reason that this was deemed necessary involves the enhancements that were discussed which must be applied to image processing. As it stands there is already a way for words to map to concepts. If this mapping can be said to be one part of the final knowledge base then there needs to be a connection to the other part, where concepts are able to map to objects obtained from images. For the purposes of this research, the objects that a robot is concerned with are uniform in color and are either simple shapes or composed of only a few simple shapes. In that case, the knowledge map for the robotic system has two important pieces of information that must exist in order to connect concepts with objects - the color of the object and the object's shape.

When the robot is given the word "red", this word is able to refer to two different concepts, both of which are *color*. One of these concepts is a child of the *descriptor* concept group in which concepts that are all related to describing and refining other concepts are contained. The other *color* concept *is* a child of the *value* concept. This concept deals with things that give discrete values to other concepts, the most common being the *number* concept. Now "red" plays two roles: be something that is describing another object and also declare that there is a value that this word represents. If the term "red ball" is analyzed, red is defining a color descriptor for the ball but also defining what the value is for that color descriptor.

This brings the discussion to the first addition that must be made to the current map of concepts. When *color* is encountered in the space of it being a *value* then some value must be stored internally so that software can access it. One way that has been explored to create a database of values for colors that can be referred to is to use what will be referred to here on as *color signatures*. (Sridharan & Stone, 2009) These signatures are obtained when extracting objects from images. The color values across all pixels that make up the object can be serialized where the matrix of color values is reshaped into a one dimensional list of values that contains all the color data present in the object of interest. This data can then be stored directly onto the robot's knowledge base and can then be retrieved for further comparison with other colors in the future.

Now that the color of an object can be identified there must be a way for the robot to recognize the shape of common objects. There are many words that represent specifications of the parent concept *object*. These words aren't part of *action* or *value* and are further described by *descriptor*. When the software encounters the word "ball," there will be a mapping to the concept *ball* under the *object* heading. The *object* concepts in the map differ from other concepts in that they need to be supplied with a set of attributes internally and cannot rely on other words in a sentence to describe their characteristics. For the purposes of identifying simple objects, an important attribute would be the shape of the object. Is the object a circle, a square, or a combination of shapes that together define an object? The software can then search for objects in the knowledge base that match the shapes associated with the object

identified in the sentence being examined. In addition to this, it is possible to save images of known objects so that in the future, when that word is again encountered, all that is necessary to identify an object is to compare it to known examples of that object.

Learning

A system has been described where objects in images and the concepts represented by language can be related to one another. However, this system only allows for a fixed set of known object-to-concept-to-language mappings. A major hurdle for designing a robot that will be able to recognize objects and communicate with a user is the fact that it is not possible to give the robot a collection of every object or word that can be encountered. To overcome this, the robot needs to have the ability to learn. While a piece of software is running it is trivial to store new information in a data structure. Issues arise when a program is stopped or a computer is turned off. The answer to this is to store the data representing the robot's knowledge on some type of more permanent storage. A system had to be found that would allow the data to be stored in a way that was fairly portable.

Very early on in the search for a knowledge storage solution, XML showed itself to be a very appropriate choice. The main benefit of XML is the ability of the programmer to define their own tags. Where the tags that exist in HTML have been defined by the W3C, the tags that XML uses are completely dependent on what type of data is to be stored in the XML document or what the XML document is supposed to represent. This made XML an obvious choice for storing a collection of related

material that could represent the knowledge a system has. When the decision to use XML was made, it was necessary to find a parser that could take data in an XML document and build an internal data structure that could represent the knowledge stored in it.

There are many parsers that exist for both MATLAB and for C++, so this first part was not a problem. The harder part of this is that in order for the robot to learn, there has to be a way to take any additional knowledge attained while running and store it in the XML file. This created the need for an XML generator. This was harder to find. The issue was that the parser had to convert data into the same structure that the generator would use to build the new XML file. There were implementations of parsers that could also generate new XML documents from the data that was parsed but, while robust, they included unnecessary functionality that made the surprisingly simple interaction required much more complicated. This prompted the creation of a custom parser and generator. This custom solution allowed for streamlining of the process to include only the functionality that was required by the knowledge representation system.

The parser functions by reading through an XML file that represents a collection of concepts and building a graph from the file. For the most part, the tags in the XML file represent concepts and sub-concepts. There are two special tags that represent other properties of the objects in the file; specifically they represent references to other objects that an object needs to have access to. The first of these is the *reqgrp* (required group) tag. This tag lets the parser know that the data contained

in this tag represents a list of other objects that are required for this object to be valid. This is a way to check whether a word is representing a given concept by ensuring that the sentence the word is in contains the other required concepts. The second tag is *req* (required, or required concept) which represents an item in a *reqgrp*. The content of this tag is the name of the required concept. The tag must also contain at least one attribute called *parent* whose value is the name representing the parent of the required concept. This is necessary as there are some concepts that have the same name but can be placed under many root concepts. For example color is in the *descriptor* concept and the *value* concept. The parent attribute ensures that the appropriate required concept is selected. Figure 5.1 shows a sample of XML code and figure 5.2 shows the graph that is built from that sample.

Figure 5.1

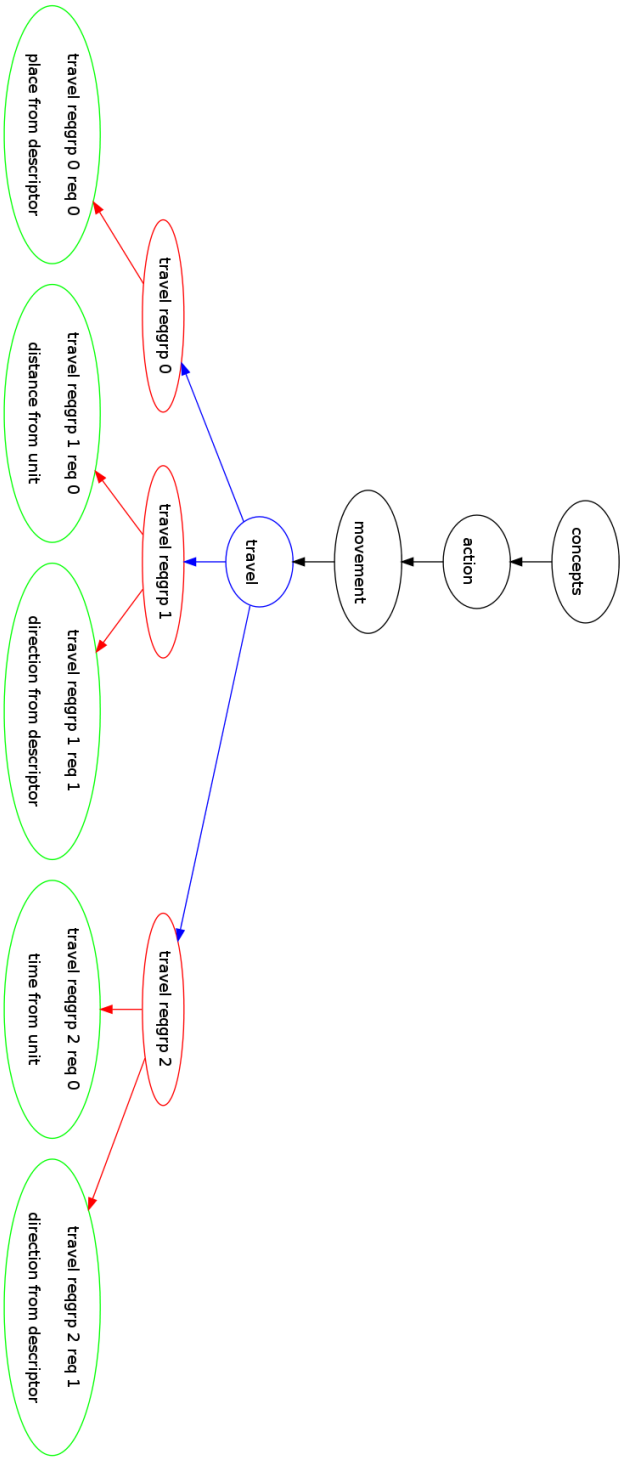
XML code that is used to create a graph of concepts

```
<concepts id="concepts">
    <action id="action">
        <movement id="movement">
            <travel id="travel">
                <reqgrp id="reqgrp">
                    <req parent="descriptor">place</req>
                </reqgrp>
                <reqgrp id="reqgrp">
                    <req parent="unit">distance</req>
                    <req parent="descriptor">direction</req>
                </reqgrp>
                <reqgrp id="reqgrp">
                    <req parent="unit">time</req>
                    <req parent="descriptor">direction</req>
                </reqgrp>
            </travel>
        </movement>
    </action>
</concepts>

```

Figure 5.2

The graph that is created by processing the XML code in figure 5.1



Concepts rule all

At this point it is time to revisit the sentence that has been used to describe many of the workings of this system: "Go to the red ball." The processes described in this thesis have made a case for designing a way for humans to interact with robots which is much more intuitive for the humans. When the KLO triad for robotic interaction is used to consider this sentence the process now goes as such. "Go" still points to the *travel* concept. "Red" still points to the *color* concepts that reside in both *descriptor* and *value*. "Ball" will refer to *ball* under the concept of *object*. When vision and image processing is brought into the system, a connection can be made. The concept of the color that is read from the sentence can refer to any number of signatures that describe what the color red can be when seen in the robots environment. The concept *ball* can now refer to known representations of balls and those representations can be compared against objects in the environment to find similar items. This has turned the sentence into a set of actions and descriptions that can call routines for the robot. First the robot would need to find a red ball by examining the area it is in. Once the ball is found the robot need simply move in the direction of the ball. (Lauria, Bugmann, Kyriacou & Klein, 2002)

Chapter 6

Future Work

Despite all the work that has been done for this research and all that has been said, there are definitely areas where improvements can be made that the researcher believes will increase a robot's abilities beyond what has been discussed

Robotics

In terms of robotics hardware, the use of laser range-finders would be desirable. Lasers have very little if any attenuation of the beam for very long distances and would be able to provide much better ranging data than either sonar or infrared. Additionally, it would be of use to implement a more powerful computer to act as the controller for the robot. The processor in the Eee PC that was used is only a 1.4 GHz single core CPU and MATLAB's image processing functions still did not process as fast on the Eee PC as they had on the researcher's personal computer. In order to obtain more environmental data, supplemental cameras could also be installed on the robot. This would also introduce the possibility of stereo vision which would allow for depth information to be gathered using the cameras.

Image Processing

The object extraction that was represented in this thesis dealt only with the recognition of objects that were of uniform color and composed of simple shapes. It is possible that algorithms and processes could be used that would allow for more

complex objects to be identified.

NLP and Learning

The act of learning for the robot that is presented here is actually a somewhat cumbersome approach. It is possible that through further research and experimentation a more efficient map of concepts can be constructed that would allow for the robot to make more inferences on its own and streamline the process of learning. Work is currently being done on other projects to further the goal of natural language processing. Projects such as WordNet and FrameNet are currently under development. These are lexical databases of word definitions and semantic maps. Projects such as the Natural Language Toolkit (NLTK) are being developed to assist in parsing language to grammatical meanings of words. Incorporation of work done using neural networks and genetic algorithms can make it easier to learn by seeing similarities between data.

Chapter 7

Conclusion

It is likely that in the future, people will indeed be communicating with robots as they would another person. The development of this interface system requires a firm foundation involving simple, versatile systems that are able to be expanded upon. The creation of an expandable robotics platform that combined a robust set of hardware and software went through multiple iterations. The iRobot Create provides the system with a sturdy base upon which to build. Using a web-cam to gather image data gives the programmer a large amount of data that can be used for navigation and identification. A netbook provides the ability to use nearly any programming language for software creation. Lastly the Arduino allows for the addition of many other sensing devices. Using C++ that can execute MATLAB functions gives the versatility and speed of C++ combined with the optimizations that MATLAB provides when working with images and other matrices. In order to interact with objects in the world a robot would need to extract them from images of the world. K-Means clustering provides a valid way to extract discrete objects from images. In order to communicate with a robot, the robot needs to be able to understand the meaning behind the words that are said. Mapping words to the concepts they represent not only gives the meaning but also reduces the number of items that would define actions for the robot to be much more manageable than the set of all English word. Finally, combining

concepts that represent colors and objects with representations of those concepts from images will provide a way to connect language to real world objects. These systems can provide a baseline for a future where people talk to autonomous robotic systems the same way they talk to their friends.

Bibliography

Ahmad. (2010). *KMeans Segmentation – MEX*. Retrieved from http://www.mathworks.com/matlabcentral/fileexchange/27969-kmeans-segmentation-mex

Buckley, R., & Süsstrunk, S., & Swen, S. (1999). Standard RGB color spaces. Proceedings from Seventh Color Imaging Conference: *Color Science, Systems and Applications*. Scottsdale, Arizona

Cuartielles, D, & Igoe, T, & Mellis, D. A. (2011, January 26). *Arduino Duemilanove*. Retrieved from http://www.arduino.cc/en/Main/ArduinoBoardDuemilanove.

Dodds, Z. (2007). pyCreate [software]. Available from http://www.cs.hmc.edu/~dodds/erdos/old.html

Flynn, A. M. (1988). Combining Sonar and Infrared Sensors for Mobile Robot Navigation. *The International Journal of Robotics Research, 7*(6), 5-14

Fodor, J.A. (1975). *The language of thought.* New York, NY: Thomas Y. Crowell Company, Inc.

Gurevych, I., & Malaka, R., & Porzel, R., & Zorn, H. (2003). Semantic coherence scoring using an ontology. Proceedings of HLT-NAACL 2003. Edmonton

Image Processing Toolbox: Color-Based Segmentation Using K-Means Clustering. Retrieved from http://www.mathworks.com/products/image/demos.html?file=/products/demos/shipping/images/ipexhistology.html

Kronfeld, A. (1990). *Reference and computation: An essay in applied philosophy of language*. New York, NY: Press Syndicate of the University of Cambridge

Lauria, S., & Bugmann, G., & Kyriacou, T., & Klein, E. (2002). Mobile robot programming using natural language. *Robotics and Autonomous Systems. 38*, 171-181

MathWorks, Inc., The. (2001). *MATLAB - The Language Of Technical Computing*. Retrieved from http://www.mathworks.com/products/matlab/

Minton, J., & Rieksts, O. (2011). Steps towards developing an intelligent robotics course. Proceedings from PACISE 2011, Shippensburg PA

OpenCVWiki. (n.d.). Retrieved fom the OpenCV Wiki: http://opencv.willowgarage.com/wiki/

Roos, N., & Vogt, P., & Wiesman, F. (2002). Automatic ontology mapping for agent communication. Proceedings from AAMAS'02. Bologna, Italy

Shapiro, L. G., & Stockman, G. C. (2001). Computer Visión. Prentice Hall. Retrieved from http://www.cse.msu.edu/~stockman/Book/.

Simmons, R. (1970). Natural language question answering systems. *Communications of the ACM*, 13(1), 15-30

Sridharan, M., & Stone, P. (2009). Color learning and illumination invariance on mobile robots: A survey. *Robotics and Autonomous Systems. 57*, 629-644