Chapter 1

Introduction

It has been seen countless time 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 into the processing of natural language by computers for quite some time.

This thesis explores the creation of a dialog based robotic platform 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 these dialog based robotics systems. This thesis first discusses the creation of a hardware and software platform that not only fit the needs for 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 a process of parsing English words into conceptual ideas. Finally the system will be explained whereby images of object 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 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 operation 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 around 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.

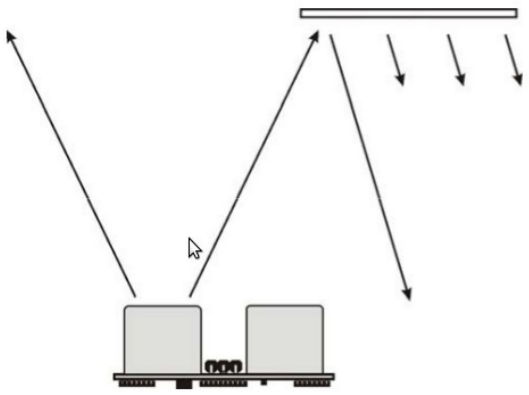
One of the main focuses of this endeavor was to research ways of allowing robots to more easily understand their environment. The most obvious way to do this was through the use of actual images of a robot's environment. The easiest way that was found to get images of the environment was to use web-cams 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 *****. 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

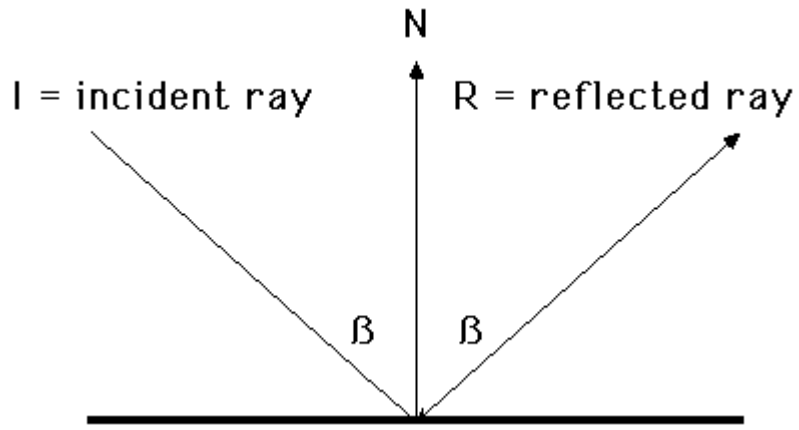
previous camera and allowed for more environmental features to analyzed.

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.

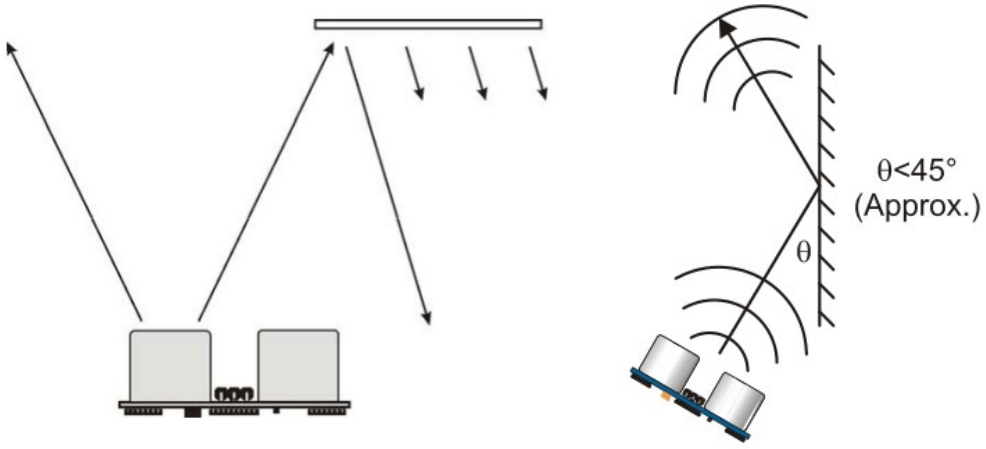
Sonar was first chosen because its operating range is very large. These range-finders will return accurate readings anywhere from 2cm to 3m from an object which is a much larger range than IR rangefinders which will be discussed in more depth later. However, there was a moderate issue encountered while experimenting with sonar on the first iteration. When the sound pulse from a sonar sensor impacts an object, depending how smooth the object is the signal may experience what is called specular reflection. 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 I.I.



This is means that the signal form the sonar will not always reflect directly back to the receiver. If any of the signal does return there is a good chance that it has bounced off of other objects and the distance value obtained is actually much farther than the true distance to the object of interest see figure i.i.i for an example**.**



These figures show instances where the sound pulse from a sonar sensor will never

return to be measured. 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 a beam of infrared light bounces off of an object instead of a pulse of sound. These range-finders however, 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 GP2D02, which has a valid range that is only from xxx to xxx. It was thought initially that this would be too small for the researchers 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 through 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 if they are impacted.

IR range-finders have a definite advantage over sonar as there is much less refraction in the pulsed signal and 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. A different 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.

***In the future 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.

Controllers

The first prototype robot that was built was simply a collection of disparate parts. A device had 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) the processing power of this controller is very minimal, and it is not possible to place a custom program on the Creates 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 it's 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. (http://www.arduino.cc/en/Main/ArduinoBoardDuemilanove)

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. Software was written which

accepts ASCII commands from the computer and carries out whatever command was sent. If the command requires a response from the Arduino it would send that data back to the computer along the 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.

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

Computer vision describes a collection of ideas that express 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, 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 them and the ball. Most importantly, they know for what is essentially 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 that has 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 a image can be represented. The most common method is to use the RGB color space. According to the CIE, a color space is a geometric representation of colors in space, usually of three dimensions [CIE]. 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 are often represented in a string of hexadecimal digits 0x0054FF, the first two values represent red intensity, the middle is green and the last group is blue. This color space can be seen in use in many computer programs such as Adobe Power-point and in computer programming such as defining the colors of various elements in HTML documents.

OpenCV

The first image processing package that was approached in the course of this research was the Open CV package of image processing tools. OpenCV [xx] (Open Source Computer Vision) is “a library of computer vision routines from Intel. First released in 2000, OpenCV code is used in applications such as object, face, and

gesture recognition, lip reading and motion tracking."[xx] 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 and development using OpenCV halted 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 [xxx]. Python initially seemed to be a good choice as there already exist bindings that let python use OpenCV natively. The issue that arose with this implementation was that Python is an interpreted language, with inefficient memory access. This cause image processing operations to take much longer then they had when using OpenCV in C++. At this point a switch could have been made to return to using C++ for the programming. Instead xxx John Spletzer a professor at Lehigh University who has done a large amount of work into robotics and computer vision was contacted. At his recommendation, MATLAB was chosen to control the robot and process images. MATLAB is a high-level language and interactive environment that enables you to perform computationally intensive tasks faster than with traditional programming languages such as C, C++, and FORTRAN [MATLAB website]. MATLAB can be nearly expanded through the addition of what are called toolboxes. MATLAB toolboxes are functions or collections of functions that help a developer to perform certain tasks. MATLAB's

Image Processing Toolbox offers a user the ability to perform operations on images that rival the operations that OpenCV can perform. MATLAB's Image Acquisition Toolbox allowed for the capturing of images from a connected web-cam. In addition to these MATLAB developed toolboxes, a toolbox developed by the robotics students at Lehigh University was used that is designed to control the iRobot Create. The fact still remained that like Python, MATLAB is also an interpreted language. The advantage lies in the purpose behind MATLAB which is the processing of data stored in matrices. This gives MATLAB 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 computers memory as simply a collection of numbers that defines the color values of each pixel in an image. In the earlier example 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 however can't initially distinguish the difference between the pixels that make up the ball from any other pixels in the image. The computer must determine what discrete pixels make up the ball and which pixels can be ignored.

Blob Extraction

Blob extraction is an process by which an image is separated into groups of pixels that share some common features. This is also called connected component extraction because blobs not only are the blobs composed of similar objects but all

objects in a blob are connected to the rest of the blob somehow.

```
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 – 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
```

figure 1.1. Pseudo-code for image blobbing algorithm

The major key to blob extraction is the union/find or disjoint-set data structure that is used to store the pixel labels. Each list of levels 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 a given label belongs to. The “create” function creates a new list containing the label given to the function. Lastly the “union” or “join” function combines two given sets into one, the resulting sets identifier will be the lower of the two initial set identifiers.

figure 1.2. Sample image consisting of 25 pixels arranged in a 5 by 5 matrix

The colors that appear in the above image are generated by the image editing software that was used to enlarge the image so that the separate pixels could be seen. As stated above, when the image is stored digitally in memory it appears as only a string of values. The table below (figure 1.3) shows how these values could be structured should they have been reshaped into a 5 by 5 matrix in which each cell contains the color information for one pixel in the form of [blue, green, red].

	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]

Figure 1.2. Five by five matrix representing the image in figure 1.1

This data is iterated through once, assigning a blob number to each pixel. If the pixel is part of a new blob then a new blob index value is created and the pixel is given that value.

figure 1.3. Blob assignment after the first pass over data in figure 1.2 is finished

	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

	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

Figure 1.4. Blob assignment after the second pass over data in figure 1.2 is finished

The second iteration over the image data joins any blobs that have different blob indexes but are in fact part of the same blob. In the above example only the the blob that pixel (3, 2) belongs to was changed. On the first pass the pixels to the right weren't checked so it was assigned to a new blob. When checking the pixel to the

right of (3, 2) they were found to be in the same blob and where marked for later joining during the second pass.

Initially blobbing seemed as though it would be a valid method for detecting objects in image. Images were blobbed by setting a threshold value for the red, blue and green values for each each pixel. As long as those values weren't to greatly different from the adjacent pixels, the pixel was added to its group. 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 objects shape and the objects average color values. If the value was set low, only portions of the object was detected and although this allowed for better average color calculation, it still distorted the complete shape of the object. When thinking of ways to allow a robot to recognize an object it was deemed necessary to have 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 is derived from statistical analysis of 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.

```
function KMEANS(data_vecs, segments) returns data_idxs, centroids:
        inputs: data_vecs – a one or two dimmensional vector of data to be
                    segmented, see figure 1.3
                segments – the number of segments the data stored in data_vecs
                    should be separated into

        centroids ← segments evenly spaced values inclusivly between min of
            data_vecs and max of data_vecs, see figure 1.4
        prev_centroids ← centroids offset by 1 rem guarantess that prev_centroids
            and centroids are not equal at start of first pass over data_vecs

        data_idxs ← empty one dimensional vector of length LENGTH(data_vecs)
        while prev_centroids ≠ centroids do:
            prev_centroids ← centroids
            dists ← empty two dimensional vector of length of first dimension
                    LENGTH(data_vecs) and length second dimension segments
            for j = 1 to segments 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 segments do:
                    new_centroid[j] ←
                            MEAN(data_vecs[x such that data_idxs[x] = j])
            centroids ← new_centroid

        return data_idxs, centroids

function SUM_OF_SQUARES(vals) returns sum_sq:
        inputs: vals – collection of values to get sum of squares for

        for j = 1 to LENGTH(vals) do:
            sq ← vals[j] ^ 2
```

```
            sum_sq ← sum_sq + sq

        return sum_sq
```

Figure 1.1. Pseudo-code for K-Means algorithm

Figure 1.2. Same 5px by 5px image from the above blobbing example

	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]

Figure 1.2. Five by five matrix representing the image in figure 1.2

	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
…	…	…	…	…	…
23	67	200	100	3	5
34	72	100	203	4	5

Figure 1.3. Two dimensional matrix obtained by transforming the five by five matrix in figure 1.2. Each row in the matrix represents one pixel. The data included for each pixel is the value of its red, green and blue color channels as well as its location in the original image

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

Figure 1.4. The minimum and maximum values of each column from the data in the matrix in figure 1.3, used to create initial centroids

	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

Figure 1.4. The initial centroids for K clusters, K being six in this case. Uses min and max from figure 1.4 as first an last respectively. Determines other values by choosing equally spaced values between min and max

	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

Figure 1.5. Cluster index matrix created after one pass over data. Each row represents the cluster that the pixel in the corresponding row in figure 1.3 belongs to.

K-Means

The first test of using the k-means algorithm showed exceptional results, beyond anything earlier blobbing techniques had shown. The initial blob extraction techniques only cared if adjacent pixels where similar to each other, this often allowed for large gradients in the colors of pixels recognized as being a connected blob, or on portions of an object that were very similar in color would be extracted. The k-means on the other hand considers the similarity between a pixel and the mean of an entire cluster. Through experimentation with using different values for k and using different color spaces for the image that would be segmented, It was shown that applying k-means to the data in an image could reliably return sets of data containing only a specific object in an image.

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, but when it is stored digitally on a computer it appears to a other 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 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 define actions or modify 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 shrunk by using the fact that many words actually represent the same idea and other words just represent different tenses of other words. This leads to the possibility of instead mapping many different words to one central concept or idea. This is known as conceptual parsing. This allows rules to be written for a much smaller and more feasible set of concepts that many different words can be mapped to when a sentence is parsed.

Conceptual Parsing

In creating a conceptual parser the the researcher first constructed a small collection of concepts that words would be able to map to. It became evident that a simple list of concepts would not be ideal. For example the idea of *movement* can refer to many different things. *Movement* can refer to moving an actuator arm or traveling from one location to another. The best way to represent the concepts was to construct a tree. The farther down the branches of the tree are traversed the more concrete the concepts become until the concept is as concrete as can be defined fr the uses of this system. Just having a collection of concepts isn't enough. There still needs to be a way for the robot to map words to these concepts. This is done through the use of a hash map. Each word points to a leaf on the tree that contains the concept the word is referring to. This system only works if each word points to exactly one concept. As was stated above this is not the case. Each word can represent many different concepts. 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 important 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 it requires. Each concept contains a collection of one or more lists. Each one of these lists represents a collection of other concepts that must also be represented in the sentence in order for the concept to be valid. 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. This 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 and the third contains “time” and “direction.” This is when the rest of the sentence must be considered. For this example let the sentence also contain the word “meters.” This word is easier to map as it can only refer to a distance measurement. This lets the robot know that the desired meaning of the word “go” is to command that the robot move a specific distance. In this case the sentence is then search for a direction. In order to determine how many meters the robot is commanded to move, the “distance” concept also contains list of requirements and one of these requirements is the concept of a value, a number in this case. If the requirements of one of a concepts list aren't all there, then the other concept lists are checked. Assume instead that the 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 is able to 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” which is used to classify objects. If on the other hand none of the other requirement lists match then the most complete list is chosen and the robot can then 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 from the work of Jerry A. Fodor and what is written in his book 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 explain various aspects of the human thought process and the way humans choose what concepts relate to each other. Fodor proposes several theories that the 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 27) He is making the statement that the process of thought is by nature one of computation where the brain weighs the validity of concepts as they describe objects or language. His second point states "Computation presupposes a medium of computation: a representational system." 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, 27) One way this can be interpreted is as way to explain a process by which humans are able to learn by inferring plausible meanings for concepts they don't yet understand or haven't learned. 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 inside 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 learning comes to light.

Learning

A major portion of designing a robot that will be able to recognize objects and communicate with a user is the fact that it's 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 robots 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 exists 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” 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 if 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” this represents an item in a reqgrp. The contents of this tag is the name of the required concept. The tag must also contain at least one attribute called “parent” who's 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 are part of different root concepts, such as color being in the descriptor concept and the value concept. The parent attribute ensures that the appropriate required concept is selected. Figure x shows a sample of XML code and figure y shows the graph that is built from that sample.