Rutgers University

Capstone Final Report

Automated Object Locating System

Group 4
Team Members:
Evan Katims
Michael McLaughlin

Advisor:
Prof. Kristin J. Dana

1. Objective

Robust and fast paced object recognition plays a fundamental role in many applications of robotics and computer vision. In our project, we make use of a high definition camera attached to a mobile lego robot in order to recognize and move to desired objects in a room. The main objective of our project is to be able to extract important data from a surrounding scene and through the use of corresponding point matches to templates, determine exactly what objects are in the camera's view. Once this data is obtained the robot can automatically move to objects of our choice, while recording its movement to later output to the user.

2. Procedure

2.1 Objects in the Scene

The first objective for our robot to complete is the recognition of all objects in a specific scene, invariant of the background. This task is accomplished by first taking a picture of the surrounding environment and extracting key points of the image. These images are then related to a database of templates for specific objects, as described in section 2.1.1. Once these key points are found our algorithm compares the amount of key points detected to a set number. It does this in order to determine if the matches are simply coincidental outliers, which then can be discarded. If we require the minimum number of point correspondences for a positive identification to be around fifteen we dramatically reduce error in outputting false positive identifications. After we iterate this process for all of the templates, we output a list of what images are located in both the scene and our database.

2.1.1 Keypoint Detection

A keypoint is defined as a pixel location where there is an extreme change in the gradient. Keypoint detection is essential for our implementation to work. The first step in matching two images is to extract the crucial points in a single image. We use David Lowes SIFT algorithm in order to accomplish this. SIFT takes one input image and recognizes its keypoints, then saves the location and description for use in the point match algorithm. The point match algorithm is used to find the relationship between the objects in a scene and the database of templates for the objects. Point correspondences between the two are matched by computing the dot products between unit vectors. We opted to do this instead of

computing Euclidean distances because the ratio of angles is a close approximation to the ratio of Euclidean distances, and it is computed significantly faster (this only holds true for small angles). We then matched each descriptor in the first image to the appropriate pixel on the second image. Once that is completed, the angle between the vector of dot products is taken via the trigonometric function arccos. If that value is less than distRatio then the two points are considered a match. Once the iteration is complete for all keypoints the robot analyses the number of key points to make an educated guess on what template the object in the scene is best matched to.

2.2 Where is the Object?

The second objective of our robot expands upon our first objective, and solves for the"Where is the object?" question. The first task that needs to be completed is that the robot needs to take a picture in the scene and analyse it. Then it needs to determine if the desired object is in the picture by using the steps that were described in section 2.1. If the object is not located within the scene, the program outputs a message saying that the object was not detected. If the object is detected then the robot moves on to the next step, which is to compute distance. We compute distance using a method called binocular vision. This method uses the fact that parallel images have horizontal epipolar lines, therefore corresponding points only have movement in the horizontal direction. This displacement has an inverse relationship with distance from the object, which is:

$$\frac{(F) \times (B)}{(D)} \tag{1}$$

Where,
F= Focal Length
B= Baseline Distance
D=Disparity

In order for this method to work it requires two parallel images. We originally obtained two parallel images by having our robot take an image from its original position and then move along the baseline so that the second image is parallel to the first. The second method we used was to rectify the images so that the images seemed like they were coming from parallel cameras. The last method we used, and are currently using is having our camera slide along the robot so that the camera is always parallel to the object.

There were two major issues with our first design that significantly decreased the precision of our distance calculations. One of the major design flaws was that we were using a three wheeled robot with only two-wheel drive. This caused imprecise movement of the lego robot, which produced inaccurate results for the measurement of distances. Also, the results varied greatly when our robot was on different terrain (i.e. tile, hardwood, carpet). In order to improve the precision of the robots movements we designed a four-wheeled robot in which the wheels on the same side (right or left) would move in unison. This eliminated the issue of drag on certain wheels that was present in our previous design. Having four wheels on our lego robot significantly reduced the error in the calculation of distance.

The improvement in the movement of the robot lead us to our next issue. The cervos on the lego robot are not very precise. Even with the new design, moving the entire robot to obtain parallel images exactly every time was very difficult and still created some errors. As with most technologies you want to eliminate as much movement as possible to reduce the possibility of errors. As discussed above it is very important to have parallel images to obtain a correct measurement of distance. One of the ways to fix this problem would be to automatically rectify images. We could then take two images from any angle and obtain parallel images. Image rectification is used to project two images onto a common plane that is parallel to the center of the two cameras. After this is complete the pixel motion between the two images lays on a horizontal line, where depth is inversely proportional to disparity. Theoretically, this concept works perfectly and would eliminate any error in movement that was causing an inaccurate measurement of distance. However, due to time constraints we needed to forfeit this idea because we kept receiving errors in the rectification due to outliers that were being picked up in our point match algorithm. For future work we would like to add a rectification toolbox to our implementation in order to make distance measurements completely accurate. With the issue of movement still at hand we redesigned the way that we obtained parallel images. In order to repeatedly acquire the images we designed an arm in front of the robot that held the camera and swung back and forth. This replaced our movement of the entire robot to obtain parallel images and was significantly more precise.

Now that we have two parallel images we can start to compute distance. We will do this by using a point match algorithm as described in 2.2.1. Once the corresponding points were found we were able to use a function that was obtained from experimental results, as described in 2.2.2. The function relates the disparity in the point correspondences to the

actual distance the object is from the robot.

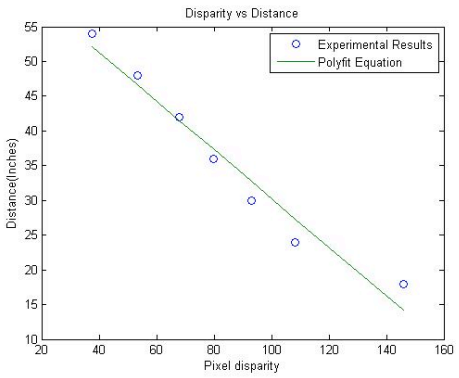
2.2.1 Point Match

When we used our original point match code it matched all similar keypoints in the images. This means since the backgrounds of the two scene pictures are so similar, a significant amount of keypoints were found in the background that were not outliers, but are not useful for our dataset and skewed the result for our template matching. In order to solve this problem we compared both scene images to the templates of the specific image. Each comparison gave us only point correspondences on the object. We then took the mean and standard deviation in the vertical and horizontal directions. With this information we were able to create a box around the desired object in each of the image scenes. Then we were able to compare the two scene images using points inside both boxes, so the points were on the desired object. This algorithm eliminates all background noise and allows us to find point disparity within the desired object.

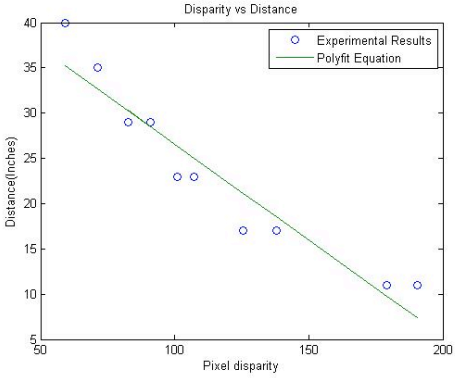
2.2.2 Disparity vs Distance

This experiment was performed in order to create an equation that would be able to directly output distance from two scene images. From the equation above we can see that disparity and distance are related to each other in binocular vision via the focal length of the camera and the baseline movement. Since we are using the same webcam, held on a constant 640x480 image throughout the entire project, the focal length remains constant but still unknown. For the baseline length, we opted to keep this constant in order to eliminate this variable. This allows us to map disparity and distance onto a standard linear representation.

The actual experiment involved taking parallel images of objects at predetermined lengths and finding the disparity in the keypoints. The distances varied from eighteen inches to fifty four inches in increments of six. The experiment was repeated multiple times in order to eliminate any human error that may have occurred at any individual data point. Once the experimental data was vectorized we used MATLABs Polyfit() function in order to give us two constants, A and B.



(a) Method 1



(b) Method 2

Figure 1: This figure shows the plot of the experimental results obtained from the distance vs. disparity tests that were described in section 2.2.2. The circles on the plot above describe the average pixel disparity at each distance ran for several tests. From these points and the polyfit() function we created a line of best fit for the data. The equation for the line is what we use to compute distance in our "Where is the object" section. Since method two provided more accurate results we decided to use that data to plot our distance polynomial. The polynomial is as follows:

$$Y = (A \times x) - B$$

A= -0.3507
B= 65.3498

$$Y = (-0.3507 \times x) - B \tag{2}$$

Where,

This equation now allows us to find any distance given the disparity between two parallel images.

3. Experimental Results

The completed result of our project was the ability of the lego robot to recognize and move to desired objects while recording every move in order to print out directions to the user as to where to find the objects. This result uses all of the algorithms mentioned above to determine the location

and distance to each object. After a user inputs a list of objects, the robot begins to take pictures of the surrounding scene and analyzes them for the desired objects. Once an object is recognized the robot takes parallel images in order to compute distance. Now the robot moves to the object and then iterates the process in looking for the next object. If the object is not within the robots vision, then the robot turns to obtain another view of the surrounding environment until the next image is visible. During this process the robot is recording every move it makes, so at the end it can output directions to all of the objects. Once the last object is found the robot outputs the directions that are needed to find all of the objects.

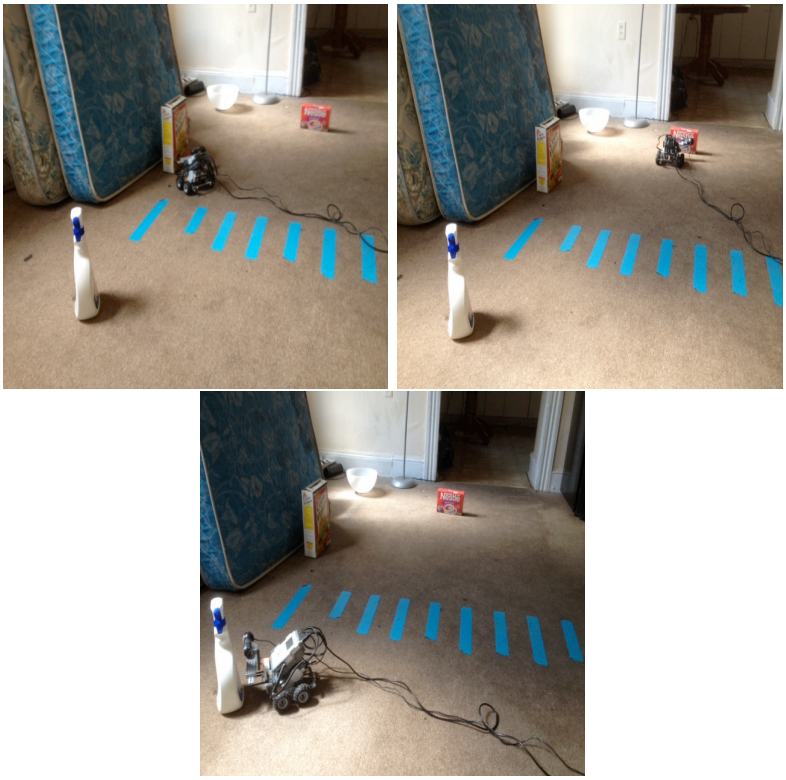


Figure 2: The three pictures above are the lego robot running through the algorithm to find all of the objects. The robot first scans the scene for the flakes. Once found, the robot moves to the 'Flakes' and records the movement. The next object that is located is the 'Nestle'. After the robot moves to the 'Nestle', it searches for the last object which is the 'Lysol'. After all of the objects are found the robot outputs the path that it took to find the objects.

3.1 "Objects in the Scene" Experimental Results

The following pictures are automatically taken by the robot when, "Objects in the Scene" button is pressed.

Figure 3: These images show our first task of the robot, which is to find all of the objects in the scene that match objects in our codebook. As you can see the program cycles through the scene and compares it to each of the images of our objects located in the codebook. If enough point correspondences are found then the program outputs a list of all the objects that it found.

3.2 "Where is the Object?" Experimental Results

The following pictures are automatically taken by the robot.

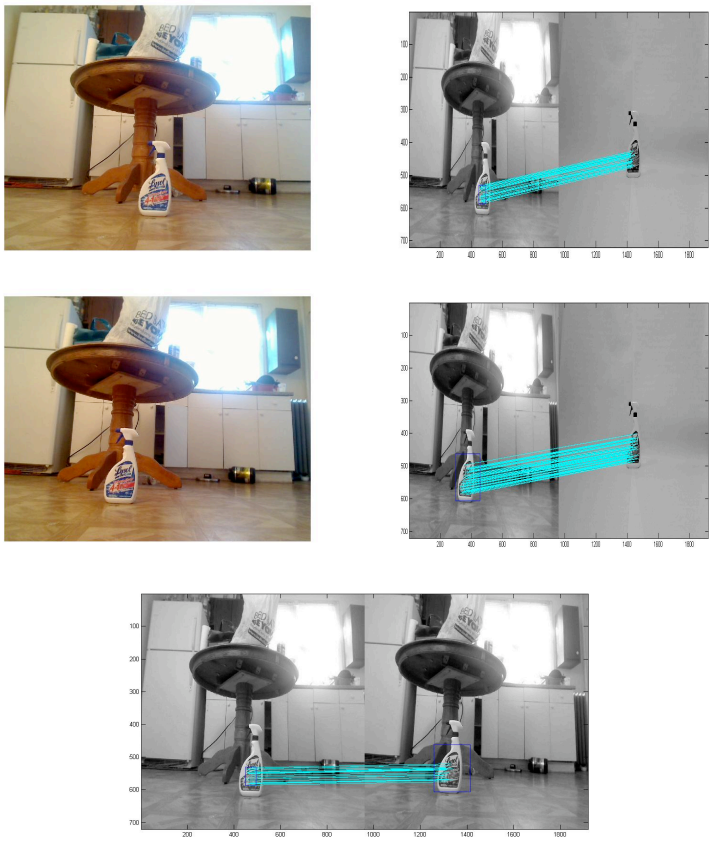
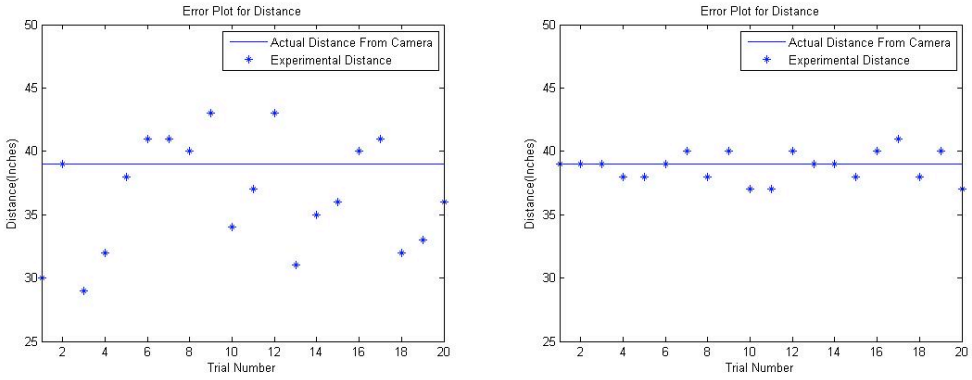


Figure 4: The images above show the progression of steps that our robot cycles through to obtain the distance to a certain object in the scene. The first step is taking an image of the surrounding environment. Once that image is taken it is compared to the template of the object that the user desires. If enough point correspondences are found then the camera moves to its parallel location and takes a second image. Next, the robot finds key points in this image. Through this process the robot is saving the location of the keypoints in order to create a box around the object. Now that we have a box around the object in each scene, we compare the parallel images with each other. The algorithm looks for point correspondences only within both boxes.



(a) Method 1 (b) Method 2

Figure 5: This figure shows the experimental results obtained from several trials of the "Where is the object?" algorithm. We placed an image at thirty nine inches from the robot and recorded each trial's computed distance. In method one, you can see the results were clearly less accurate and unpredictable. The outliers that caused this error are due to inconsistencies in the turning of the robot. If the robot under turns in its movement towards obtaining parallel images, then the outputted distance is closer than the actual distance. If the robot over turns its movement towards obtaining parallel images, then the outputted distance is further than the actual distance. In method two only the camera is moving, and this greatly reduced the error in finding the correct distance. The remaining error is due to the fact that we are not using high quality servos, therefore the spinning rotation is not always exact.

4. Discussion

4.1 Difficulties and Sources of Error

One of the difficulties that was associated with this project was running out of memory in MATLAB. If we used a higher quality image MATLAB ran out of available memory and crashed when we were running our algorithms. Unfortunately, this lead to some sources of error when detecting keypoints in an image. With high quality images the number of keypoints is much greater, which leads to more accuracy in labeling an image. Another benefit of higher quality images is that we would be able to have better object recognition at further distances. At our current picture quality our robot can recognize objects up to approximately six feet from the camera. At any further distance an object viewed from the webcam is too small to

distinguish keypoints, therefore making it unable to be identified. Our objective was to find a reasonable picture quality that would give us an adequate amount of point matches. The code also needed to run efficiently as well as successfully. For us that happy medium was a 640x480 picture. Any lower quality picture spared time at the cost of keypoints and any higher quality image either took too long to run or crashed MATLAB.

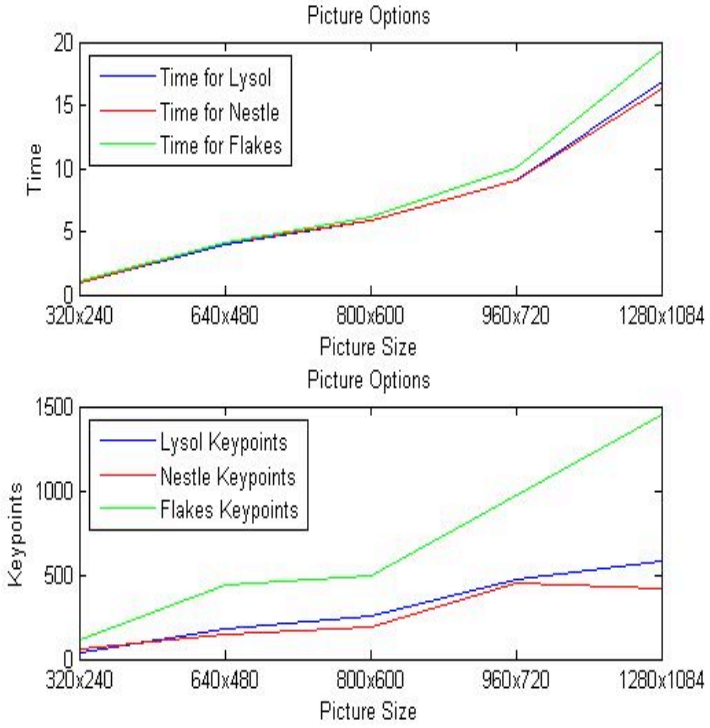


Figure 6: These figures show the experimental results that were outputted from numerous tests involving the picture quality. As per the figure, as the picture quality increases the number of keypoints detected in the object also increases. However, in increasing the quality of the picture, the time the code takes to extract keypoints greatly increases. For the purpose of our project we determined that the best image size that would give us an acceptable amount of keypoints in a reasonable amount of time was 960x720.

Another difficulty in our project was that the servos on the lego robot are not incredibly precise. Since our code is highly dependent on the acquisition of parallel images, any stray from exact parallel images results in an error in the calculation of distance. We found that sometimes our robot would not fully turn the required distance to obtain parallel images due to a jam in the pivot of the third wheel. Through many tests we found that it was very hard to repeat results when the robots movement was completely dependent on the servos and wheel rotation. One of the things that improved our results was the material that the robot was moving on. If the robot was on carpet some of the wheels began to drag, whereas when we tested on tile or hardwood, our results were much more accurate. In order to improve these results we created our four wheeled robot. This robot was much less effected by the terrain and moved significantly more accurately.

Our four wheeled robot showed a great deal of improvement over the three wheeled bot, but still had a large source of error when moving along the baseline to obtain parallel images. Our first idea to remove this error was to use image rectification on two images. With more time we would have been successful in using an image rectification algorithm. However, we would obtain errors every so often due to the fact that we were picking up outliers in the surrounding scene. Improvements could be made into our point matching algorithm that would eliminate all errors using image rectification. To replace image rectification we decided that a motorized arm that held the camera in front of the robot that moves in a parellel plane would accomplish our goal. This addition greatly reduced error from our first design and created near perfect results every time.

Over the past few weeks we were also able to add a mapping system onto our robot. The idea behind this is that the user can choose numerous objects from a list that it would like the robot to map out. Once the robot has the list it will search the surrounding area for the closest object. When the closest object is found the robot will proceed to move to the first object where it will take another image of the surrounding scene and move to the next closest object. This process will iterate until all of the images are found. Throughout the whole process the robot will record all of the steps it takes so it will be able to output the final map of shortest distance to all the objects to the user.

5. Current Trends in Robotics

Computer vision is currently a broad field and is extremely popular in todays research. The ability to have a computer analyze images as a human

would is an extremely useful task, yet it is not a trivial one. The base of many automated technologies relies on the initial task of being able to distinguish the difference between objects in a scene. This is why a large amount of research is currently being done on this topic. Technology is constantly improving and research into computer vision is playing a huge part of this. Computer vision is currently being applied to technology such as navigation, video surveillance, medical analysis, and industrial robots just to name a few.

One of the largest areas of concern around the world, especially in the United States, is the protection of its citizens. Credit card fraud, theft, image tampering, and terrorism are among the largest areas of threats that relate directly to a countrys citizens. It is important as a nation to strive towards improving its monitoring systems in these areas in order to try to limit and prevent the damages theses crimes have against the people of this or any nation. Research into computer vision provides a great way to automatically analyze data faster and more efficiently than most humans can.

In order to develop these complicated algorithms, we need to first look at simpler cases such as object recognition and template matching. In the past five years a lot of research has gone into improving how objects are detected. One of the issues that was originally presented in object recognition with template matching was that there is not a huge difference between certain objects. For example, a tire on a car is very different than a tire on a bus, but when run through basic template matches the computer may believe that they are the same image. This presents a huge error when the precision of recognizing exactly what image is being displayed is important. Nima Razavi, Juergen Gall and Luc Van Gools paper on Scalable Multi-class Object Detection introduces a way to improve the accuracy in determining the difference between similar objects. When looking at an object they not only look at the object in question, but also its location and surrounding features. In setting up a probability function to compare the images to several different templates, the algorithm can vote for the template that has the highest odds of providing a match. Now, this is a very low level objective, but forms a strong base for future work.

Now that we have discussed the improvements that are being made in recognizing images, a new problem presents itself. Being able to detect images with a high precision is very important, but doing this robustly and quickly is just as important. If it takes an algorithm minutes to find a match, then there is little use for this algorithm in a real world scenario. Research into improving template matching for the use of object detection and facial recognition has become a very popular field in computer vision.

Researcher, Alexander Sibiryakov from ASL (Electronics and Vision) Ltd, recently wrote a paper, Fast and High-Performance Template Matching Method, which goes into the importance of a robust and fast paced method of template matching. He was able to offer an improved method to template matching by applying probability to analyze different gradients as well as Lp- and Hamming distances of an image to come up with a prediction at the best match possible. His method not only offers an improvement in speed, but also opens doors into new research into what he describes as, the ability to design a rotation and scale invariant method using multiple geometrically transformed templates.

Both of the processes described above can be combined into a technology that can be implemented within real world applications. Terrorism is unfortunately something that takes place in our world each and every day. The United States have created watch lists in order to keep authorities on high alert of major criminals and terrorists. Often times it is very hard for law enforcement to recognize the people that are on these lists. P.J. Phillips recent research shows that computers often times outperform humans on facial recognition, especially when the humans are observing people in different shades of light and from different angles. If a system can be presented that would effectively analyze a real time camera for criminals or terrorists, many lives could be saved and crimes stopped.

Rama Chellappa of the University of Maryland, Pawan Sinha of Massachusetts Institute of Technology, and P. Jonathon Phillips of National Institute of Standards and Technology have recently discussed the steps that are required to provide successful comparisons to watch lists. In their paper, *Face Recognition By Computers and Humans*, they discuss the importance of being able to verify identities as well as compare people to watch lists via facial recognition sequences. They stress that several steps must be taken in order to successfully create a system that accomplishes the goal of matching a picture of a person to a known identity. One of the more important steps is being able to take one angle of a person and synthesize new images from the reconstructed 3D models. This must be done in order to be able to recognize people from all angles. Another important problem is that people age. If the last known picture of a criminal was from ten years ago, then the current systems would have issues recognizing the face. Some improvements can be made by prediction using the craniofacial growth model. As these technologies are developed even further, there will be more successful algorithms that can perform facial recognition. This system is good for recognizing criminals however, the technology is still not advanced enough to obtain accurate results from a distance further than tens of meters away. Also, the ability to analyze videos quickly enough to

the point where these people may be stopped is still a little bit behind. As we become more advanced in our technologies the field of computer vision will continue to grow and it will be easier to connect these methods that were described above into an efficient algorithm.

One unique aspect of computer vision is that it is always growing and there is always room for improvement. When research into computer vision first started, scientists were trying to figure out certain basic steps such as recognizing the difference between a person and a horse. Once that task was completed, it opened up new doors into now finding the difference between a car tire and a bus tire, which lead into researching faster and more efficient ways to do the comparison. With each problem that is solved in computer vision a new problem is brought to light. This is what makes computer vision such a large area of interest in the world of technological advances. There is always room for improvement and always new doors that are opened.

References

[1] Nima Razavi, Juergen Gall and Luc Van Gools. *Scalable Multi-class Object Detection*. In *Computer Vision Papers*, CVPR 2011.

[2] R. I. Hartley and A. Zisserman.*Multiple View Geometry in Computer* Vision. Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[3] Alexander Sibiryakov. *Fast and High-Performance Template Matching Method*. In *Computer Vision Papers* , CVPR 2011.

[4] E. Trucco and A. Verri.*Introductory Techniques for 3-D Computer Vision* . Prentice Hall, Inc., Upper Saddle River, New Jersey, first edition edition, 1998.

[5] Chellappa, R. and Sinha, P. and Phillips, P.J.*Face Recognition by Computers and Humans* . In Computer Journal , 2010. IEEE Workshop, pages 46-55, 2010.

[6] P.J. Phillips et al.FRVT 2006 and ICE 2006 Large Scale Results, IEEE Trans.*Pattern Analysis and Machine Intelligence* , forthcoming; DOI 10.1109/TPAMI.2009.59

```
function Parallel=BackUp()
% Check Robot
if verLessThan('RWTHMindstormsNXT', '3.00');
    error(strcat('ROBOT NOT CONNECTED OR TOOLBOX NOT SET TO PATH'));
end

%% Constants and so on
TableLength      =200; %620 is a foot    % in degrees of motor rotations
Ports = [MOTOR_A; MOTOR_C];  % motorports for left and right wheel
DrivingSpeed     = -40; %Speed of Motors


%% Open Bluetooth connetion
h = COM_OpenNXT('bluetooth.ini');
COM_SetDefaultNXT(h);


%% Initialize motor-objects:

mStraight                   = NXTMotor(Ports);
mStraight.SpeedRegulation   = false;  % not for sync mode
mStraight.Power             = DrivingSpeed;
mStraight.TachoLimit        = TableLength;
mStraight.ActionAtTachoLimit = 'Brake';

%% Rotate
 mStraight.SendToNXT();
 mStraight.WaitFor();

%% Close Bluetooth connection
COM_CloseNXT(h);
```

```
function Parallel=CamBack()
% Check Robot
if verLessThan('RWTHMindstormsNXT', '3.00');
    error(strcat('ROBOT NOT CONNECTED OR TOOLBOX NOT SET TO PATH'));
end

%% Constants and so on
TableLength      = 135;     % in degrees of motor rotations
Ports = [MOTOR_B];  % motorports for left and right wheel
DrivingSpeed     = -20; %Speed of Motors

%% Open Bluetooth connetion
h = COM_OpenNXT('bluetooth.ini');
COM_SetDefaultNXT(h);


%% Initialize motor-objects:

mCamera                   = NXTMotor(Ports);
mCamera.SpeedRegulation   = false;  % not for sync mode
mCamera.Power             = DrivingSpeed;
mCamera.TachoLimit        = TableLength;
mCamera.ActionAtTachoLimit = 'Brake';

%% Rotate
 mCamera.SendToNXT();
 mCamera.WaitFor();

%% Close Bluetooth connection
COM_CloseNXT(h);
```

```
function Parallel=Camera()
% Check Robot
if verLessThan('RWTHMindstormsNXT', '3.00');
    error(strcat('ROBOT NOT CONNECTED OR TOOLBOX NOT SET TO PATH'));
end

%% Constants and so on
TableLength      = 135;     % in degrees of motor rotations
Ports = [MOTOR_B];  % motorports for left and right wheel
DrivingSpeed     = 20; %Speed of Motors

%% Open Bluetooth connetion
h = COM_OpenNXT('bluetooth.ini');
COM_SetDefaultNXT(h);


%% Initialize motor-objects:

mCamera                   = NXTMotor(Ports);
mCamera.SpeedRegulation   = false;  % not for sync mode
mCamera.Power             = DrivingSpeed;
mCamera.TachoLimit        = TableLength;
mCamera.ActionAtTachoLimit = 'Brake';

%% Rotate
 mCamera.SendToNXT();
 mCamera.WaitFor();

%% Close Bluetooth connection
COM_CloseNXT(h);
```

```
function[First Second Third  counterone countertwo
counterthree ]=ComparePicture(Template, Object, Item, First, Second,
Third , counterone, countertwo, counterthree )

vid = videoinput('winvideo',1, 'RGB24_640x480');
start(vid);
im = getsnapshot(vid);
figure,imshow(im);
imwrite(im,'Center.png')
[box,~,Number, meanPointC]=Pointmatch('Center.png', Template);

if (Item(1)==1&&Item(2)==0&&Item(3)==0)
while(Number>10)
FindCenter(meanPointC)

vid = videoinput('winvideo',1, 'RGB24_640x480');
start(vid);
im = getsnapshot(vid);
figure,imshow(im);
imwrite(im,'Image0.png')

[box1,A1,num1, meanPoint1]=Pointmatch('Image0.png', Template);
fprintf(Object); fprintf('Found \n');

Camera()

vid = videoinput('winvideo',1, 'RGB24_640x480');
start(vid);
im = getsnapshot(vid);
figure,imshow(im);
imwrite(im,'Image1.png')
[box2,A2,num2, meanPoint2]=Pointmatch('Image1.png', Template);
[Amin Bmin]=Pointmatch1('Image0.png', 'Image1.png',box1,box2);
Amean=[mean(Amin(1,:))];
Bmean=[mean(Bmin(1,:))];
Disparity= abs(Amean-Bmean);
value=Disparity(1)
A= -0.2124;   B=47.8232;
DistanceMoving=A*(value)+B
First
First=floor(DistanceMoving)
Straight(First)
CamBack()
BackUp()
break
end
```

```
 if(Number<10)
 TurnRight()
 counterone=counterone +1;
 Item=[1 0 0]
 if ((strcmpi(Template,'LysolClose2.jpg')==1))
 [First Second Third counterone countertwo
counterthree]=ComparePicture('LysolClose2.jpg', 'Lysol', Item,First,
Second, Third, counterone, countertwo, counterthree );
 elseif ((strcmpi(Template,'Flakes1.jpg')==1))
 [First Second Third counterone countertwo
counterthree]=ComparePicture('Flakes1.jpg', 'Flakes',Item,First, Second,
Third, counterone, countertwo, counterthree );
 elseif ((strcmpi(Template,'Nestle1.jpg')==1))
[First Second Third counterone countertwo
counterthree]=ComparePicture('Nestle1.jpg', 'Nestle',Item,First, Second,
Third, counterone, countertwo, counterthree );
 end
 end
end



if (Item(1)==1&&Item(2)==1&&Item(3)==0)
    while(Number>10)
FindCenter(meanPointC)

vid = videoinput('winvideo',1, 'RGB24_640x480');
start(vid);
im = getsnapshot(vid);
figure,imshow(im);
imwrite(im,'Image0.png')

[box1,A1,num1, meanPoint1]=Pointmatch('Image0.png', Template);
fprintf(Object); fprintf('Found \n');

Camera()

vid = videoinput('winvideo',1, 'RGB24_640x480');
start(vid);
im = getsnapshot(vid);
figure,imshow(im);
imwrite(im,'Image1.png')
[box2,A2,num2, meanPoint2]=Pointmatch('Image1.png', Template);
[Amin Bmin]=Pointmatch1('Image0.png', 'Image1.png',box1,box2);
Amean=[mean(Amin(1,:))];
```

```
Bmean=[mean(Bmin(1,:))];
Disparity= abs(Amean-Bmean);
value=Disparity(1)
A= -0.2124;   B=47.8232;
DistanceMoving=A*(value)+B
Second=floor(DistanceMoving)
Straight(Second)
CamBack()
BackUp()

break
end
 if(Number<10)
     TurnRight()
     countertwo=countertwo +1;
     Item=[1 1 0]
 if ((strcmpi(Template,'LysolClose2.jpg')==1))
 [First Second Third counterone countertwo
counterthree]=ComparePicture('LysolClose2.jpg', 'Lysol', Item,First,
Second, Third, counterone, countertwo, counterthree );
 elseif ((strcmpi(Template,'Flakes1.jpg')==1))
 [First Second Third counterone countertwo
counterthree]=ComparePicture('Flakes1.jpg', 'Flakes', Item,First, Second,
Third, counterone, countertwo, counterthree );
 elseif ((strcmpi(Template,'Nestle1.jpg')==1))
[First Second Third counterone countertwo
counterthree]=ComparePicture('Nestle1.jpg', 'Nestle', Item,First, Second,
Third, counterone, countertwo, counterthree );
 end
 end
end

 if (Item(1)==1&&Item(2)==1&&Item(3)==1)
     while(Number>10)
FindCenter(meanPointC)

vid = videoinput('winvideo',1, 'RGB24_640x480');
start(vid);
im = getsnapshot(vid);
figure,imshow(im);
imwrite(im,'Image0.png')

[box1,A1,num1, meanPoint1]=Pointmatch('Image0.png', Template);
fprintf(Object); fprintf('Found \n');

Camera()
```

```
vid = videoinput('winvideo',1, 'RGB24_640x480');
start(vid);
im = getsnapshot(vid);
figure,imshow(im);
imwrite(im,'Image1.png')
[box2,A2,num2, meanPoint2]=Pointmatch('Image1.png', Template);
[Amin Bmin]=Pointmatch1('Image0.png', 'Image1.png',box1,box2);
Amean=[mean(Amin(1,:))];
Bmean=[mean(Bmin(1,:))];
Disparity= abs(Amean-Bmean);
value=Disparity(1)
A= -0.2124;   B=47.8232;
DistanceMoving=A*(value)+B
Third=floor(DistanceMoving)
Straight(Third)
CamBack()
BackUp()
clc
fprintf('The Robot turned %4.2f degrees clockwise in order to find the
first object. \n',counterone*30)
fprintf('The first Movement was %4.2f inches Straight. \n',First)
fprintf('The Robot turned %4.2f degrees clockwise in order to find the
second object. \n',countertwo*30)
 fprintf('The second Movement was %4.2f inches Straight. \n',Second)
 fprintf('The Robot turned %4.2f degrees clockwise in order to find the
third object. \n',counterthree*30)
 fprintf('The third Movement was  %4.2f inches Straight. \n',Third)
 fprintf('The Robot centers the camera on the object once it is found.\n')
 fprintf('At the end of each movement, the robot reverses 4 inches. \n')
break
end
 if(Number<10)
     TurnRight()
     counterthree=counterthree +1;
     Item=[1 1 1]
 if ((strcmpi(Template,'LysolClose2.jpg')==1))
 [First Second Third counterone countertwo
counterthree]=ComparePicture('LysolClose2.jpg', 'Lysol', Item,First,
Second , Third, counterone, countertwo, counterthree  );
 elseif ((strcmpi(Template,'Flakes1.jpg')==1))
[First Second Third counterone countertwo
counterthree]=ComparePicture('Flakes1.jpg', 'Flakes', Item,First ,  Second,
Third, counterone, countertwo, counterthree  );
 elseif ((strcmpi(Template,'Nestle1.jpg')==1))
 [First Second Third counterone countertwo
```

```
counterthree]=ComparePicture('Nestle1.jpg', 'Nestle', Item,First ,  Second,
Third , counterone, countertwo, counterthree );
 end
 end
end
```

```
function []= CreatePath(Object1,Object2,Object3)
Item=[1 0 0]
First=0; Second=0; Third=0;
counterone=0; countertwo=0; counterthree=0;
if ((strcmpi(Object1,'Lysol')==1))
[First Second Third counterone countertwo
counterthree]=ComparePicture('LysolClose2.jpg', 'Lysol',Item,First ,
Second , Third, counterone, countertwo, counterthree );%,CountObject)
elseif ((strcmpi(Object1,'Flakes')==1))
[First Second Third counterone countertwo
counterthree]=ComparePicture('Flakes1.jpg', 'Flakes',Item,First , Second ,
Third,counterone, countertwo, counterthree );%,CountObject)
elseif ((strcmpi(Object1,'Nestle')==1))
[First Second Third counterone countertwo
counterthree]=ComparePicture('Nestle1.jpg', 'Nestle',Item,First , Second ,
Third,counterone, countertwo, counterthree );%,CountObject)
end


Item=[1 1 0]
if ((strcmpi(Object2,'Lysol')==1))
[First Second Third counterone countertwo
counterthree]=ComparePicture('LysolClose2.jpg', 'Lysol',Item,First ,
Second , Third,counterone, countertwo, counterthree );%,CountObject)
elseif ((strcmpi(Object2,'Flakes')==1))
[First Second Third counterone countertwo
counterthree]=ComparePicture('Flakes1.jpg', 'Flakes',Item,First , Second ,
Third,counterone, countertwo, counterthree );%,CountObject)
elseif ((strcmpi(Object2,'Nestle')==1))
[First Second Third counterone countertwo
counterthree]=ComparePicture('Nestle1.jpg', 'Nestle',Item,First , Second ,
Third,counterone, countertwo, counterthree );%,CountObject)
end

Item=[1 1 1]
if ((strcmpi(Object3,'Lysol')==1))
[First Second Third counterone countertwo
counterthree]=ComparePicture('LysolClose2.jpg', 'Lysol',Item,First ,
Second , Third,counterone, countertwo, counterthree );%,CountObject)
elseif ((strcmpi(Object3,'Flakes')==1))
[First Second Third counterone countertwo
counterthree]=ComparePicture('Flakes1.jpg', 'Flakes',Item,First , Second ,
Third,counterone, countertwo, counterthree );%,CountObject)
elseif ((strcmpi(Object3,'Nestle')==1))
[First Second Third counterone countertwo
counterthree]=ComparePicture('Nestle1.jpg', 'Nestle',Item,First , Second ,
```

```
Third,counterone, countertwo, counterthree );%,CountObject)
end
```

```
function [Amin,Bmin] = Pointmatch1(image1, image2,box,box2)

% Find SIFT keypoints for each image
[im1, des1, loc1] = sift(image1);
[im2, des2, loc2] = sift(image2);


distRatio = 0.6;

des2t = des2';                          % Precompute matrix transpose
for i = 1 : size(des1,1)
   dotprods = des1(i,:) * des2t;        % Computes vector of dot products
   [vals,indx] = sort(acos(dotprods));  % Take inverse cosine and sort
results

   % Check if nearest neighbor has angle less than distRatio times 2nd.
   if (vals(1) < distRatio * vals(2))
      match(i) = indx(1);
   else
      match(i) = 0;
   end
end

% Create a new image showing the two images side by side.
im3 = appendimages(im1,im2);

% Show a figure with lines joining the accepted matches.
figure('Position', [100 100 size(im3,2) size(im3,1)]);
colormap('gray');
imagesc(im3);
hold on;
cols1 = size(im1,2);
A=[];B=[];
for i = 1: size(des1,1)
  if (match(i) > 0)
       A=[A [loc1(i,2) ; loc1(i,1)]];
         B=[B [loc2(match(i),2);loc2(match(i),1)]];
  end
end

Amin=[];Bmin=[];
for i = 1: size(des1,1)
  if match(i) > 0 && loc1(i,2)>box(1) && loc1(i,2)<box(2) && loc1(i,
1)>box(3) && loc1(i,1)<box(4) && loc2(match(i),2)>box2(1) && loc2(match(i),
```

```
2)<box2(2) && loc2(match(i),1)>box2(3) && loc2(match(i),1)<box2(4)
    line([loc1(i,2) loc2(match(i),2)+cols1], ...
        [loc1(i,1) loc2(match(i),1)], 'Color', 'c')

       Amin=[Amin [loc1(i,2) ; loc1(i,1)]];
        Bmin=[Bmin [loc2(match(i),2);loc2(match(i),1)]];
  end
end
hold on

num = sum(match > 0);
fprintf('Found %d matches.\n', num);
plot([box(1) box(1) box(2) box(2) box(1)],[box(3) box(4) box(4) box(3)
box(3)]);
hold on
plot([box2(1)+cols1 box2(1)+cols1 box2(2)+cols1 box2(2)+cols1
box2(1)+cols1],[box2(3) box2(4) box2(4) box2(3) box2(3)]);
```

```
function Parallel=Straight(DistanceMoving)
% Check Robot
if verLessThan('RWTHMindstormsNXT', '3.00');
    error(strcat('ROBOT NOT CONNECTED OR TOOLBOX NOT SET TO PATH'));
end

%% Constants and so on
TableLength      = DistanceMoving*52; %620 is a foot    % in degrees of
motor rotations
Ports = [MOTOR_A; MOTOR_C];  % motorports for left and right wheel
DrivingSpeed     = 80; %Speed of Motors


%% Open Bluetooth connetion
h = COM_OpenNXT('bluetooth.ini');
COM_SetDefaultNXT(h);


%% Initialize motor-objects:

mStraight                    = NXTMotor(Ports);
mStraight.SpeedRegulation    = false;  % not for sync mode
mStraight.Power              = DrivingSpeed;
mStraight.TachoLimit         = TableLength;
mStraight.ActionAtTachoLimit = 'Brake';

%% Rotate
 mStraight.SendToNXT();
 mStraight.WaitFor();

%% Close Bluetooth connection
COM_CloseNXT(h);
```

```
function Right=TurnRight()
h = COM_OpenNXT('bluetooth.ini');
COM_SetDefaultNXT(h);

%% Rotate
  l_wheel = NXTMotor( MOTOR_A );
  r_wheel = NXTMotor( MOTOR_C );


  l_wheel.Power =-30;
  r_wheel.Power = 30;

  l_wheel.TachoLimit =90;
  r_wheel.TachoLimit = 90;

  l_wheel.SendToNXT()
  r_wheel.SendToNXT()

  l_wheel.WaitFor()
  r_wheel.WaitFor()
  COM_CloseNXT(h);

end
```