# Comparing Deep Neural Networks and Traditional Vision Algorithms in Mobile Robotics

**Andy Lee**
Swarthmore College
alee3@swarthmore.edu

## ABSTRACT

We consider the problem of object detection on a mobile robot by comparing and contrasting two types of algorithms for computer vision. The first approach is coined "traditional computer vision" and refers to using commonly known feature descriptors (SIFT, SURF, BRIEF, etc.) for object detection. The second approach uses Deep Neural Networks for object detection. We show that deep neural networks perform better than traditional algorithms, but discuss major trade offs surrounding performance and training time.

### Author Keywords

Computer Vision; Deep Neural Network; SIFT,SURF; Caffe; ROS; CS81; Adaptive Robotics

## INTRODUCTION

Computer vision is an integral part of many robotic applications. In the 90s, we saw the rise of feature descriptors (SIFT, SURF) as the primary technique used to solve a host of computer vision problems (image classification, object detection, face recognition). Often these feature descriptors are combined with traditional machine learning classification algorithms such as Support Vector Machines and K-Nearest Neighbors to solve the aforementioned computer vision problems.

Recently there has been an explosion in hype for deep-neural networks. In fact, there has been a wide-spread adoption of various deep-neural network architectures for computer vision because of the apparent empirical success these architectures have seen in various image classification tasks. In fact, the seminal paper *ImageNet Classification with Deep Convolutional Neural Networks* has been cited over 3000 times.[2] This is also the model used as a standard out-of-the-box model for a popular deep neural network framework called Caffe.



In this paper, we consider the problem of object detection on a mobile robot. Specifically, we compare and contrast two types of algorithms for object detection. The first approach is coined "traditional computer vision" and refers to using commonly known feature descriptors (SIFT, SURF, BRIEF, etc.) for object detection alongside common machine learning algorithms (Support Vector Machine, K-Nearest Neighbor) for prediction. In contrast, the second approach uses Deep Neural Networks architectures.

Our emphasis will be on analyzing the performance of two approaches in a mobile-robotic setting characterized by a real-time environment that constantly changes (lighting, nearby objects are constantly moving). Specifically, we wish to detect objects while the mobile agent is moving, differing from the performance on still-image tasks – the focus of object detection papers.

Our paper is broken up as follows.

- Background knowledge on traditional vs deep neural network approaches.
- Survey of relevant litearture in computer vision.
- Discussion of experiments
- Appendix: Implementation details for mobile robotics (navigation, ROS, hardware)

## COMPUTER VISION BACKGROUND

Computer vision can be succinctly described as finding telling features from images to help discriminate objects and/or classes of objects. While humans are innately endowed with incredible vision faculties; it is not clear which features humans use to get such strong performance on vision tasks. Computer vision algorithms in general work by extracting feature vectors from images and using these feature vectors to classify images.

Our experiment considers two different approaches to computer vision. On the one hand are traditional approaches to computer vision. These approaches date back to the past 10-20 years and are characterized by extracting human-engineered features (edges, corners, color) deemed to be relevant in vision tasks. One can say these techniques lean towards a *human-driven* approach.

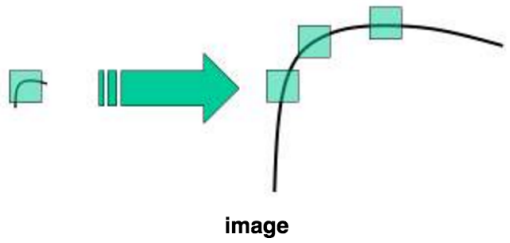


**Figure 1. Example showing scaling changes detection of a corner.**

On the other end of the spectrum are techniques stemming from deep neural networks, which are quickly gaining popularity for their success in various computer vision tasks. Once again the goal is to extract features that help discriminate objects; however, this time these features are learned via an automated procedure using non-linear statistical models (deep nets). The general idea of deep-neural networks is to learn a denser and denser (more abstract) representation of the training image as you proceed up the architecture. In this section, we provide a brief but more detailed introduction to these techniques.

### Traditional Vision

We now describe some techniques used in traditional vision. The idea behind each of these techniques is to formulate some way of representing the image by encoding the existence of various features. These features can be corners, color-schemes, texture of image, etc.

#### *SIFT (Scale-Invariant Feature Transform)*

The SIFT algorithm deals with the problem that certain image features like edges and corners are not scale-invariant. In other words, there are times when a corner looks like a corner, but looks like a completely different item when the image is blown up by a few factors. The SIFT algorithm uses a series of mathematical approximations to learn a representation of the image that is scale-invariant. In effect, it tries to standardize all images (if the image is blown up, SIFT shrinks it; if the image is shrunk, SIFT enlarges it). This corresponds to the idea that if some feature (say a corner) can be detected in an image using some square-window of dimension $\sigma$ across the pixels, then we would if the image was scaled to be larger, we would need a larger dimension $k\sigma$ to capture the same corner (see figure 1). The mathematical ideas of SIFT are skipped, but the general idea is that SIFT standardizes the scale of the image then detects important key features. The existence of these features are subsequently encoded into a vector used to represent the image. Exactly what constitutes an *important feature* is beyond the scope of this paper.

#### *SURF (Speeded-Up Robust Features)*

The problem with SIFT is that the algorithm itself uses a series of approximations using difference of Gaussians for standardizing the scale. Unfortunately, this approximation scheme is slow. SURF is simply a speeded-up version of SIFT. SURF works by finding a quick and dirty approximation to the difference of Gaussians using a technique called box blur. A box blur is the average value of all the images values in a given rectangle and it can be computed efficiently. [1, 5]

**Figure 2. A picture of SIFT feature keypoints from OpenCV.**

#### *BRIEF (Binary Robust Independent Elementary Features)*

In practice, SIFT uses a 128 dimension vector for its feature descriptors and SURF uses a minimum of 64 dimension vector. Traditionally, these vectors are reduced using some dimensionality reduction method like PCA (Principal Components Analysis) or ICA (Independent Components Analysis). BRIEF takes a shortcut by avoiding the computation of the feature descriptors that both SIFT and SURF rely on. Instead, it uses a procedure to select $n$ patches of pixels and computes a pixel intensity metric. These $n$ patches and their corresponding intensities are used to represent the image. In practice this is faster than computing a feature descriptor and trying to find features.

### Deep Neural Networks

The deep networks we examine in this paper are convoultional neural networks. For those familiar with artificial neural networks, these are simply multi-level neural networks with a few special properties in place (pooling, convolution, etc.). The basic idea is that we will take a raw RGB image and perform a series of transformations on the image. On each transformation, we learn a denser representation of the image. We then take this denser representation, apply the transformation and learn an even denser representation. It turns out by using this procedure, we learn more and more abstract features with which to represent the original image. At the end, we can use these abstract features to predict using some traditional classification method (and it works surprisingly well). We now discuss the architecture of the convolutional neural network. Notice that each of the layers we describe below,

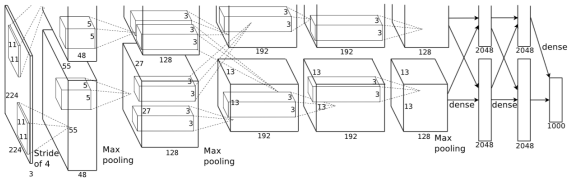


**Figure 3. CNN training process.**

transform the input in some fashion and allow us to reach for more abstract features.

The implementation of the CNN and some of its techniques are discussed in the below section. Because of space constraints, we also skip a detailed discussion on the various layers (convolution, rectified linear units, max-pooling) because this was thoroughly explained in class already.

## RELEVANT LITERATURE

In this section, we survey 2 related pieces of literature. We provide a brief summary of the techniques used and discuss the implications and connections with our project.

### ImageNet Classification with Deep Convolutional Neural Networks

Arguably one of the most influential papers in applying deep learning to computer vision, this paper discusses a neural network containing over $60$ million parameters and $60$ million parameters that significantly beat previous state-of-the-art approaches to image recognition in a popular computer vision competition: ISVRC-2012 [2]. For those familiar with neural networks, the key innovation fo Convolutional Neural Networks is that the layers are not fully connected; this is to improve learning time and prevent overfitting.

The architecture of the featured convolutional neural network is given by 5 convolutional layers that are followed by max-pooling layers and 3 full-connected layers with a final 1000-way softmax. Important innovations of this paper include an alternative to the traditional sigmoid activation function and methods for reducing over-fitting. Traditionally, activation functions take the form

$$f(x) = (1 + e^{-x})^{-1} \text{ or } \tanh(x)$$

However, the paper advocates the use of *Rectified Linear Units (ReLUs)*, which refers to the activation function

$$f(x) = \max(0, x)$$

The max function significantly accelerates learning time. To reduce over-fitting, the authors also artificially enlarged the data set by generating image translations and horizontal reflections on each image, while preserving their labels. This increased the training set by a factor of 2048. While the new training examples are highly correlated with the original examples, this data augmentation forces the network to learn multiple representations of the same image, thereby encouraging generalization. While the exact details are given in the paper, we simply state that the CNN described in this paper achieves an error rate of $37.5$ as compared to a more traditional approach of using SIFT + FV features, which achieves an error rate of $45.7$.

### CNN Features off-the-shelf: an Astounding Baseline for Recognition

The authors take a different approach to deep-learning.[4] Rather than use a deep neural network for prediction, the authors extract features from the first fully-connected layer of the *OverFeat* network which was trained to perform object classification on ILSVRC13. In other words, the deep net step tells us *which features* are important to consider when doing learning on image data; think of this step as dimensionality reduction to learn condensed representations of the image. The next step is to determine how effective this condensed representation is for prediction. Hence, with these features in mind, the authors train simple classifiers (Support Vector Machine with linear kernel) on images specific to a target data set. Here's a summary of what we discussed.

1. Train a deep-neural network on ILSVRC13
2. Record the features $F$ learned from the first fully connected layer from step (1).
3. Consider a brand-new image data set (Pascal VOC, Oxford 102 Flowers).
4. Extract features $F$ from each image of data set.
5. Train SVM (linear kernel) on this new data set.
6. Report results.

The incredible and perhaps counter-intuitive result is that this approach performed similarly to networks trained specifically for the object classification on ILSVRC13. In this way, the authors have demonstrated an alternative to meticulously tuning a copious amount of hyper-parameters that is commonly employed in computer vision. Put another way, the authors provide a compelling claim that the features obtained from deep-convolutional nets should be the primary features used in most visual recognition tasks. This SVM + CNN approach significantly outperformed tradition techniques using SIFT features.

One final note. The authors tested this approach on a variety of visual recognition tasks, including attribute detection, image classification and object detection – all of which performed considerably well. In each task, the authors also experimented with data-augmentation procedures similar to the techniques described in the previous paper. Data-augmentation performed slightly better than no data-augmentation.

### Implications

While we did not have a chance to fully implement and explore the techniques presented in these two papers because of time constraints, we take this time to point out a few observations and thoughts. Without a doubt, the first paper demonstrated the efficacy of deep-neural networks in computer vision tasks. Moreover, the results of both papers demonstrate that traditional vision techniques are quickly being outmuscled by deep neural networks. However, that does not mean traditional vision techniques are obsolete; many of the image processing tasks for cleaning a data set remain the same. For example, we saw in both papers that underlying principles (augmenting a data set by shifting, translating and rotating) are being used in conjunction with deep neural networks.

Often a criticism of deep neural networks is that they require a massive data set in order to train effectively, otherwise they are prone to over-fitting. Moreover, there is a long tedious process of tuning parameters. The second paper, however, demonstrated that a combined approach of using learned features from deep nets on traditional linear classifiers can be extremely effective. We believe this represents a huge breakthrough for robotics as it allows us to leverage the power of deep neural networks in a way that makes the simple computer vision tasks of our experiment very tractable. In fact, the authors showed that even a generic model from Caffe is good enough to begin using this approach.

The papers discussed highlight two computational innovations using machine learning for computer vision. There has also been recent literature exploring how to use *context* to better detect objects. For instance Torralba, Murphy and Freeman learn feature global feature descriptors to learn a scene and use such scene information as a type of prior when performing object detection [3]. Their techniques heavily rely on performing inference on graphical models. We skip the mathematics behind this approach, but it represents yet another approach to computer vision – translating intuitive principles of human vision to computers.

## EXPERIMENT

We are interested in how traditional computer vision algorithms perform against deep neural networks in a mobile robotic setting. In our case, we focused specifically on object detection on a moving turtlebot in the hopes of identifying factors that determined robustness (or lack thereof) in object detection. Furthermore, once those factors had been identified, we pushed to achieve strong object detection for both algorithms. Admittedly, our experiments are less theoretical than those covered in our survey of relevant literature; therefore our results focus mainly on the practical considerations of vision in robotics. We were motivated by the following questions.

1. Would deep neural networks outperform traditional vision algorithms and vice-versa? If so, by how much? If not, why not?
2. How robust would object detection in general be? Would we be able to detect objects all the time, some of the time, seldomly or none of the time? What factors contribute to this?
3. What are the practical issues that make it difficult to perform vision in the mobile robotic setting (changing lighting conditions, dynamic environment, stochastic nature of sensors)?

### Experiment Hypothesis and Design

Given the recent popularity of deep neural networks, we hypothesized that CNNs would significantly outperform traditional feature-descriptor algorithms (SIFT, SURF). By how much, we were not sure. The first step was to design an experiment to benchmark these two approaches. We wanted our experiment to benchmark computer vision in the mobile-robotic setting. To accomplish this, we placed our mobile-agent (turtlebot) in a closed room with a target object to detect. We then start the robot at a fixed position and let it wander using a simple autonomous exploration algorithm for a set period of time (45 seconds). This interval was chosen to be long enough so as to ensure the object would appear in the turtlebot's camera feed thereby giving our computer vision algorithms a chance to classify the object. If the vision correctly identifies the object, then it stops and we record the iteration as a success. Otherwise, if time runs out we record the iteration as a failure. The same procedure is run identically for CNNs and our traditional vision algorithms.

Our experiment took place in the G.R.A.S.P (General Robotics Automation, Sensing and Perception) laboratory at the University of Pennsylvania. The lab contains a semi-closed area to run robotic experiments characterized by a green-turf carpet. Around are chairs and desks that act as office space for members of the lab.We set the robot's autonomous explorer to stay within the confines of the open-space and not venture to the office areas. Hence, one can think of the robot exploring the area of the green-turf carpet until it has found its desired object (see figure 4).

Note there is some stochasticity in the way the robot explores (sometimes it explores closer to the object, other times farther away). Therefore, on some iterations the robot may be biased to be in positions that allow it to better detect the object. However, the autonomous exploration procedure does not change when we use CNNs vs traditional algorithms; the two components are independent. In other words, one type of algorithm does not have an inherent advantage over the other in terms of exploration. That means if we run the experiment a sufficient number of times then we would have data to report statistically rigorous results that indicated whether or not one algorithm performed better than the other. For our experiment, we run each setup 10 times for each algorithm and

record the results. In this context, setups are characterized by the object the robot is required to detect.

Finally, we make one last comment regarding our experiments. Others have proposed other benchmarks to compare the two algorithms. For example, we could have recorded (in addition to successes and failures) on average how quickly each algorithm identified an object corectly. This is a subject for future work, but for the time being we were simply interested in knowing which algorithm detected better. We felt this setup was enough to get some interesting results, but also easy enough to convey our results. We summarize each run of our experiment in a setup below and provide a picture that captures our environment.

*Step by step procedure for one run-through*

1. Place robot in a fixed position. (same procedure for CNN and traditional vision)
2. Place the desired object for the robot to find in the controlled area and ensure that the object is in the robot's line of vision if robot faces object.
3. Initiate autonomous exploration on the robot and watch it map the environment.
4. Set a timer for 45 seconds.
5. Record whether or not the robot stopped during this process, which would indicate the robot has found a desired object.

### Experiment Results and Discussion

We now discuss the results of our experiments. Firstly, we decided on two objects for our setups: a soccer ball and a computer keyboard. Because our turtlebot used a webcam that was elevated on a mast, we needed to choose objects that could be elevated enough to be the the turtlebot's line of vision. Both those objects could be detected by the built-in caffe network. In order to test these objects on our traditional algorithms, we used the find-2d-object package to extract SIFT,SURF and BRIEF features from those objects (discussed below). So that we didn't bias the results, we trained the find-2d-object algorithm on different but similar keyboards in an environment outside of the actual experiment environment.

In earlier experiments, we realized there was an inherent flaw in our experiment setup. In the original setup, the robot explores autonomously while the vision algorithms try to detect

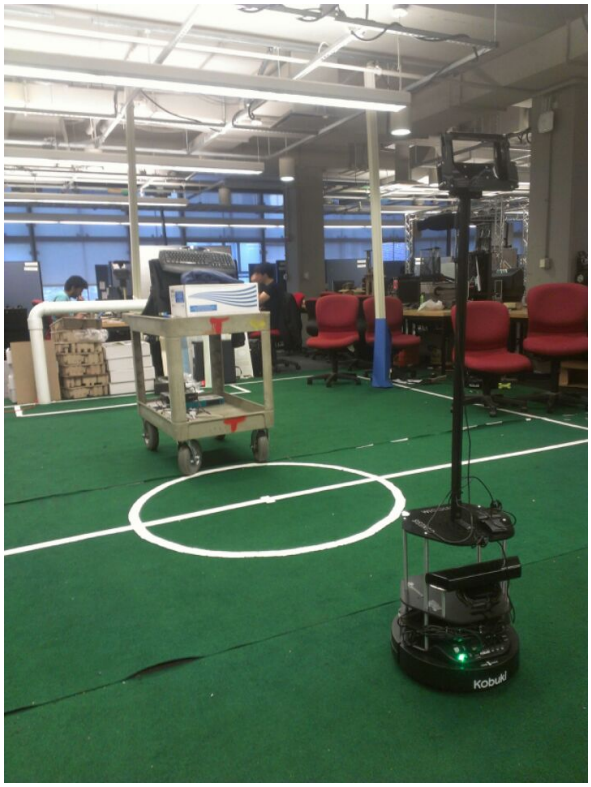


**Figure 4. The robot directly faces the target object, a keyboard on an elevated platform.**

the objects coming in from the camera-feed. The problem is while the robot explores, there is significant movement of the camera, especially given that our camera is mounted on a thin mast. It turns out that this movement induces blurry images, which negatively affects the prediction accuracy. In fact, an accurate prediction is simply not possible with a blurry image. Therefore, with the original setup it was impossible for the turtlebot to detect an image. Essentially, the problem was that there was never enough of a window for the camera to get an accurate read on its surroundings. To overcome this difficulty, we needed to modify our exploration behavior. Rather than explore on each iteration, we set up the robot to turn in place very slowly after each exploration step. This accomplished two things. Firstly, by turning in place a small degree each step, the camera was able to get an accurate read of its environment. Secondly, we paused on each turn so that each algorithm could make 5 predictions on each turn. This allowed each algorithm to localize the object and make an informed prediction.

| Modified Explore (Keyboard) | Number of successful trials (out of total trials) | Duration of each run |
|---|---|---|
| Caffe | 6/10 | 45 |
| Find Object 2d | 4/10 | 45 |

**Table 1. Modified exploration with keyboard.**

| Modified Explore (Ball) | Number of successful trials (out of total trials) | Duration of each run |
|---|---|---|
| Caffe | 4/10 | 45 |
| Find Object 2d | 3/10 | 45 |

**Table 2. Modified exploration with ball.**

**Figure 5. A picture of the keyboard (target object) on an elevated platform that is in the robot's line of vision.**

### Discussion and Conclusions

We notice that the performance of the algorithms improved dramatically using the modified exploration procedure. That said, we take this time to note a few more aspects of the experiment that the data does not illustrate. It turns out that both algorithms worked amazingly well when they were extremely close to the object and could get a good reading. The neural network always recognized the keyboard and ball when it was within a few inches of it and find-object-2D recognized the objects about 70 percent of the time. The problem is that our autonomous exploration procedure does not always guarantee that we will get close enough to the object to achieve this result; this follows from the stochastic nature of our exploration scheme.

Therefore, to extend this project we would have liked to implement one additional feature of object localization that would encourage the robot to move closer to objects when it sees one. This would guarantee that the robot would achieve the necessary distance to get a precise reading on an object and allow us to strictly consider the vision algorithms rather than have the autonomous exploration scheme potentially skew our results in the wrong way. Another thing that was not well illustrated from the data was the fact that find-object-2D only worked well with objects with prominent features. Therefore, objects like paper plates (one solid color, no edges) fared very poorly. On the other hand, the paper plates did not fare as poorly with caffe. Hence, deep neural networks can detect a much wider range of objects, at least based on our observations.

In addition, it seemed that there was difficulty detecting an smaller objects such as keyboards and balls. One natural extension to achieve a more refined comparison between two computer vision approaches is to run our experiments on larger objects such as chairs and doors.

To conclude, we note a particularly important tradeoff between CNNs and traditional vision algorithms. To train our traditional vision algorithms (extracting feature descriptors), we only needed to process an object in the camera feed for about 30 seconds to 1 minute and this allowed us to get 70 percent accuracy. In essence, traditional vision algorithms represent a quick and dirty (though surprisingly effective) method for doing computer vision, but they are limited not just by accuracy but also the objects that they can detect. On the other hand, had we decided to train a CNN from scratch, we would have needed a few hundred thousand training examples of our objects in order to train effectively. This because deep networks take many many training examples before they are effective. As such , we can see an immediate tradeoff between accuracy and an investment of time, energy and resources. Therefore, we can conclude that for on-the-fly mobile robotic applications, traditional vision algorithms still have its place, despite the wide-spread success of deep networks.

## CONCLUSION AND FUTURE DIRECTIONS

In this paper, we discussed a simple computer vision experiment that compared traditional computer vision algorithms and deep-neural networks. We showed that the deep neural network performed better, but explained the difficulties that we faced with running object detection on a moving robot. We also surveyed a few pieces of literature representing research directions in computer vision. With more time, we would have particularly liked to explore using CNN features and running simple classifiers (SVM) to see how such an approach would fare in the mobile robotic setting. In addition, we would have also liked to explore scene recognition and determine how much an improvement we would gain with object detection given scene knowledge.


## ACKNOWLEDGMENTS

We would like to thank Dr. Eric Eaton for letting us audit his Integrated Intelligence for Robotics class and for providing the turtlebot resources, without which this experiment would not have been possible. We would also like to thank the following Upenn students Yifan Wang, Guan Sun, Christopher Clingerman, Pedro Tecchio for providing assistance in installing Caffe and OpenCV on the turtlebot as well as providing a set of helpful eyes when debugging integration issues. Finally, we would like to thank Lisa Meeden for providing a wonderful classroom experience in her CS81 (Adaptive

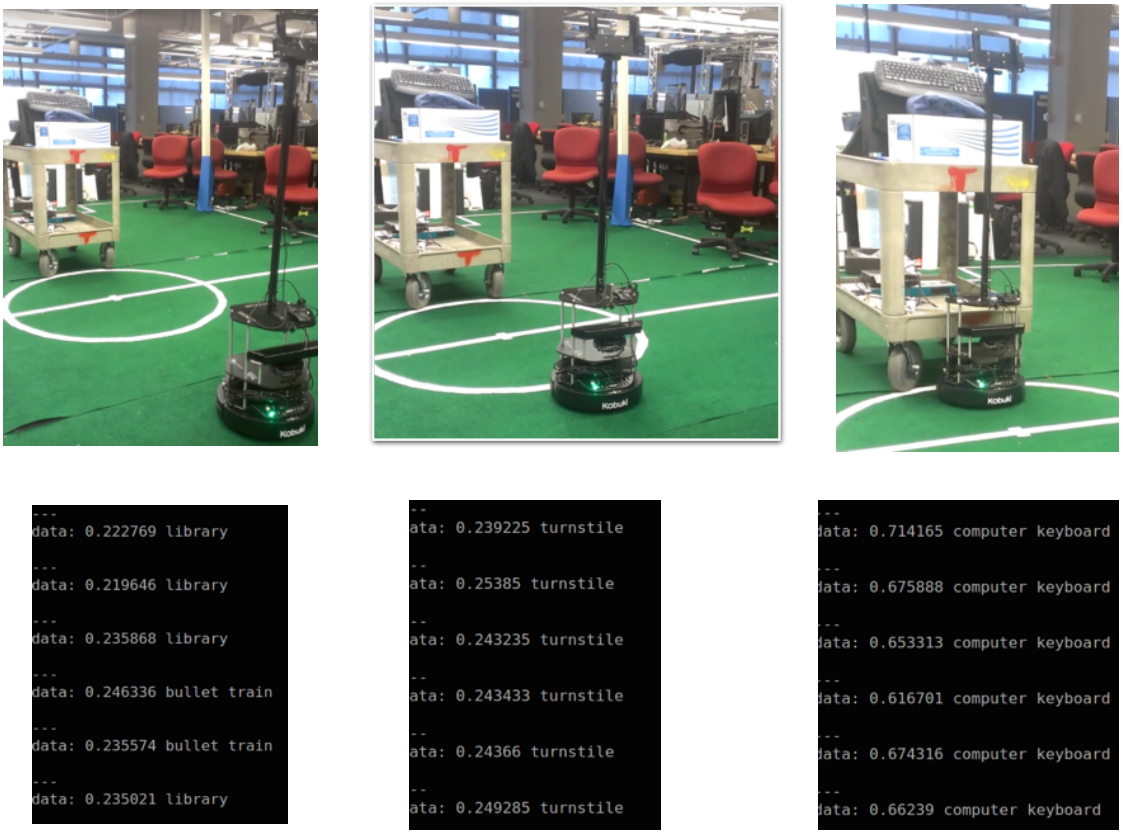


**Figure 6. Caffe predictions based on distance from keyboard.**

Robotics) class.

## REFERENCES


1. Dsp.stackexchange.com. Understanding surf features calculation process, 2015.
2. Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. 2012.
3. Murphy, K. P., Torralba, A., and Freeman, W. T. Using the forest to see the trees: A graphical model relating features, objects, and scenes. In *Advances in Neural Information Processing Systems 16*, S. Thrun, L. Saul, and B. Schölkopf, Eds. MIT Press, 2004, 1499–1506.
4. Razavian, A. S., Azizpour, H., Sullivan, J., and Carlsson, S. Cnn features off-the-shelf: an astounding baseline for recognition, 2014.
5. Stackoverflow.com. How to apply box filter on integral image? (surf), 2015.


## APPENDIX

For the interested reader, we discuss the major components that powered our turtlebot, including turtlebot hardware, ROS and navigation software. Note that this section can be safely skipped because the paper's focus is on computer vision. We have simply included this for those that may way to reproduce our experiments.

### Vision Software

In this section, we provide details on the actual vision algorithms used, including their corresponding software packages.

#### *OpenCV*

OpenCV is perhaps the most widely used open-source software for computer vision. It supports a variety of functions like converting RGB images to gray-scale, re-sizing images and extracting feature descriptors. In our case, the two algorithms deep neural networks and traditional feature descriptors (described below) both depended on many tools found in openCV. In that sense, OpenCV acts as a fundamental software package that a host of computer vision algorithms rely

on.

We make a brief digression to talk about the version of OpenCV that was used. The most current long-term support version if OpenCV 3.0, however, we experienced a large number of integration issues when trying to use OpenCV. The reason is many of our applications (Caffe) has been configured to use openCV 2.4.10. We quickly realized that openCV 3.0 was not backwards compatible and so we decided to stick to using version 2.4.10, which worked fairly well.

*Caffe*

Caffe is an open-source deep learning framework developed by BVLC ( Berkeley Vision and Learning Center). There were many very good, very well-maintained open frameworks for deep learning, but we ultimately chose Caffe because it was easiest to integrate with ROS. A few things to note.

The out-of-the-box model given by Caffe is a slight variation on the network featured in *ImageNet Classification with Deep Convolutional Neural Networks*. We did not make any changes to this neural network because it was able to detect the objects in our experiment straight out of the box. In addition to Caffe, we used a plugin called *ROS Caffe* which provides a bridge between ROS and Caffe to facilitate integration. *ROS Caffe* allows us to subscribe the the camera-feed topic and directly publish a prediction on a ROS topic named "caffe-ret".

### Find Object 2D

We used a ROS package called *Find Object 2d* for traditional computer vision. This package is a simple Qt interface to try OpenCV implementations of SIFT, SURF, FAST, BRIEF and other feature detectors and descriptors for objects recognition. To use Find Object 2D, we simply needed to place an object in front of the camera. The package had a very easy-to-use training procedure to extract features by recording a series of camera feeds. Using *Find Object 2d* we can subscribe to the camera-feed topic and directly publish position, rotation, scale and shear of an object. As an aside, notice that this package allows us to localize objects, a feature that *ROS Caffe* does not support.

### Hardware

Our mobile agent is a standard turtlebot manufactured by Willow Garage. The turtlebot comes with a standard Microsoft Kinect for Xbox 360, which 3d-sensing in the form of a depth-sensor and RGB capabilities; the Kinect is used strictly for localization and obstacle avoidance and not for any computer vision tasks. We decided on a few additions to the turtlebot in order enable the use of a webcam for computer vision tasks. These additions include a mast, use of the NUCi7 for onboard computing and a Microsoft Lifecam for the webcam.

### ROS

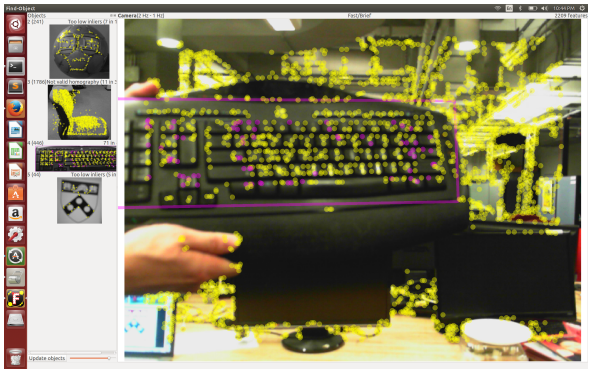

**Figure 7. Image of using Find-Object-2D to extract keyboard features.**

We use the ROS (Robotic Operating System) framework to program our turtlebot. ROS imposes a standard event-based programming paradigm primarily involving function-callbacks for robotics that streamlines the development process. In addition, it provides uniform guidelines for the development of software packages in an effort to encourage seamless integration.

*ROS Implementation*

"The fundamental concepts of the ROS implementation are nodes, messages, topics, and services.

**Nodes** are processes that perform computation. ROS is designed to be modular at a fine-grained scale: a system is typically comprised of many nodes. In this context, the term node is interchangeable with software module. Our use of the term node arises from visualizations of ROS-based systems at runtime: when many nodes are running, it is convenient to render the peer-to-peer communications as a graph, with processes as graph nodes and the peer-to-peer links as arcs.

Nodes communicate with each other by passing messages. A **message** is a a strictly typed data structure. Standard primitive types (integer, floating point, boolean, etc.) are supported, as are arrays of primitive types and constants. Messages can be composed of other messages, and arrays of other messages, nested arbitrarily deep.

A node sends a message by **publishing** it to a given topic, which is simply a string such as odometry or map. A node that is interested in a certain kind of data will **subscribe** to the appropriate **topic**. There may be multiple concurrent publishers and subscribers for a single topic, and a single node may publish and/or subscribe to multiple topics. In general, publishers and subscribers are not aware of each others existence. " [1]

### Navigation Software

[1] Taken from http://www.robotics.stanford.edu/ ang/papers/icraoss09-ROS.pdf

We use a variety of software packages to both execute simple functionality on our turtlebot (navigation) and carry out object detection algorithms. We defer the software for computer vision for a later section. This section focuses strictly on the packages we used to do navigation on the turtlebot.

### *Move base*

*Move base* is a package that comes standard with turtlebot. Given a map, it allows the user to issue goal-based navigation directives to the turtlebot. Subsequently, *move base* executes this goal and abstracts away the path planning process. Because most of our applications provides a location-based navigation goal (object position, autonomous mapping), *move base* is called for nearly every navigation task.

### *Gmapping*

"The gmapping package provides laser-based SLAM (Simultaneous Localization and Mapping), as a ROS node called slam gmapping. Using slam gmapping, you can create a 2-D occupancy grid map (like a building floor plan) from laser and pose data collected by a mobile robot." [2]

Gmapping is currently the most popular algorithm used for SLAM. It uses Rao-Blackwellized particle filters to map the environment and is a critical component to any form of autonomous exploration (described below). To the novice robotics programmer, gmapping can be thought of as primarily a package for using kinect camera feeds to map the environment. This map is subsequently used for path-planning, obstacle avoidance and autonomous exploration.

### *Hector Navigation*

It will be clear later, but our experiment relies on the ability for a robot to autonomously explore its environment. Since gmapping only provides a method for localizing and mapping, we need an additional component that allows the robot to explore unknown areas (points that have not been mapped).

To accomplish this task, we use a sub-module of the Hector Navigation package called Hector Explore. Hector Explore works by looking at unmapped regions and generating a host of possible routes. The algorithm then selects one of these routes and calls *move base* to execute the route.

A very popular alternative to Hector Navigation for autonomous exploration is the Frontier Exploration package. In fact, the Frontier Exploration package is well-documented and maintained; it works for not just the turtlebot but for a series of other mobile robotics. We tried Frontier Exploration, but quickly ran into multiple issues that was very difficult to debug. For some reason, Frontier Explore generated a series of routes that were deemed impossible to execute by *move base*, despite working in simulations. For this reason, we decided it was simply easier to switch to Hector Explore rather than try debugging the problem.

[2] Taken from http://wiki.ros.org/gmapping