Real-time Computer Perception of Human Nonverbal Behavior

Thomas Weng
Yale University
April 30, 2015
Advisor: Brian Scassellati
Graduate Advisor: Henny Admoni

Abstract

Humans intuitively perform nonverbal behaviors in social interactions, using movements and gestures to both add emphasis and convey additional information that may not be expressed in speech alone. Robots that are able to recognize and mimic these behaviors can be more effective agents in social settings. Reliable nonverbal behavior recognition, however, is a difficult task, as many components of the problem are largely not mathematically well defined. Some of the technical challenges include tracking humans in a scene, geometrically inferring the projection of a point gesture or head pose, and accounting for ambiguous human movements.

My project makes use of recent breakthroughs in hardware and software technology to build a perception system capable of detecting human nonverbal behavior in real-time. Designed for robots engaging in object-based collaborative tasks, the system detects behaviors such as pointing, head orientation, and vocal input, determining the most likely target object for a given behavior. The Microsoft Kinect v2 acts as the primary source of input to the system, which interfaces with robots through Robot Operating System, the standard communication protocol for robot platforms. This system provides researchers at the Yale Social Robotics Lab and elsewhere the ability to incorporate real-time nonverbal behavior detection into experiments on human-robot interaction. Our evaluations of the system demonstrate that it recognizes human nonverbal gestures and their intended target objects with a high level of accuracy in real-time. The evaluations also demonstrate that the system's performance is most constrained by the number of objects of interest it must recognize.

Introduction

Recent breakthroughs in hardware and software technology have enabled the development of powerful perception systems for robotics research. The Microsoft Kinect, first released for development in 2012, provided researchers with an inexpensive sensor bar that outperformed existing sensing solutions. With body tracking, depth sensing, and other features coming built in with the device, the Kinect solved several hard problems for the research community and lowered the barrier of entry for performing experiments requiring human sensing (Han, 2013). The second version of the Kinect, released for development less than a year ago, improved upon the original

by both providing more precise data streams and adding additional features such as face tracking.[i] Another recent innovation accelerating the pace of robotics research is the development of Robot Operating System (ROS), a flexible set of software libraries and tools for programming robots. ROS development started in 2007 and has since become the standard tool for programming robot platforms.

We would like to take advantage of the capabilities of both the Kinect and ROS for experiments on human-robot interaction. The sensing technology provided by the Kinect allows us to easily track humans and recognize their movements and gestures, and ROS provides the framework through which we can program robots to recognize and respond to these cues. However, compatibility issues prevent these two technologies from pairing easily: ROS is only compatible with Unix systems, whereas the Kinect and its Software Development Kit are designed for Windows only. While there have been efforts to reverse-engineer the drivers of the original Kinect for Linux, this method loses the SDK and all built-in software such as body tracking, significantly reducing the feature set of the device. Furthermore, no such driver solution exists for the Kinect v2. As a result, there has been no way to integrate the Kinect v2 with ROS.

This paper describes a perception system that combines the Kinect and ROS technologies and demonstrates the detection of human nonverbal behaviors in real-time. The system takes data from the Kinect sensor running on Windows and sends it over a wireless network to a ROS instance on a Linux machine, which controls robot motions and behaviors. This implementation preserves the software features that reverse-engineering the proprietary drivers does not, and is also the first such project we know of that successfully bridges the two technologies. Originally a collaborative project between Yale and MIT researchers, the system described in this paper branches from the original and focuses on the domain of spatial collaborative tasks and recognition of human nonverbal behavior. The resulting system can detect objects in a scene, track human nonverbal behavior, and determine the target objects of the behaviors.

The perception system described here opens new possibilities for incorporating real-time behavior detection in human-robot interaction experiments. While we focus in this paper on detecting nonverbal behaviors relevant to object-based collaboration tasks, the sensing capabilities of the Kinect can certainly be leveraged in other areas of HRI research, such as robot tutoring systems. This system can be taken as a proof-of-concept to be modified and adjusted as required for future experiments.

Design and Implementation

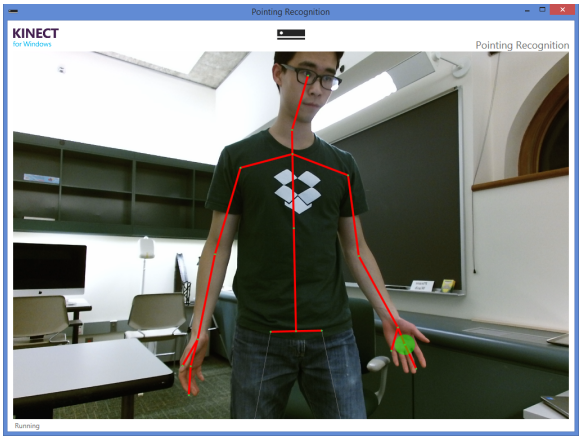
Requirements

The system must be able to: (1) locate objects of interest in 3D coordinate space, (2) detect humans to track key cues such as pointing, head orientation, and simple vocal input, and (3) rank objects by the likelihood that the object is the target of a particular gesture. We use head orientation as a proxy for eye gaze, which the Kinect is unable track. These three tasks must be performed

in real-time, meaning that processing must keep pace with the input of 30 fps, with minimal dropping of frames.

Input

The Kinect provides sensing data on independent streams at 30 frames per second. The system primarily uses the Depth, Color, Body, and Face streams. The Depth and Color streams simply provide image data arriving on the IR and color cameras, whereas the Body and Face streams are a little more complex; these streams provide the results of preprocessing through Microsoft's proprietary machine learning algorithms, which include both coordinates (i.e. positions of body joints and face points) and features (i.e. whether an individual is engaged with the device). See Fig. 1 for visualizations of the Body and Face streams.



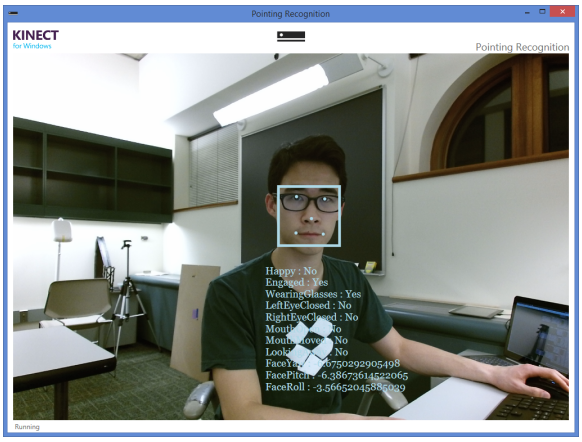


Figure 1. Visualizations of data from the Body and Face streams.

System Architecture

See Fig. 2 for a diagram of high-level architecture of the system.

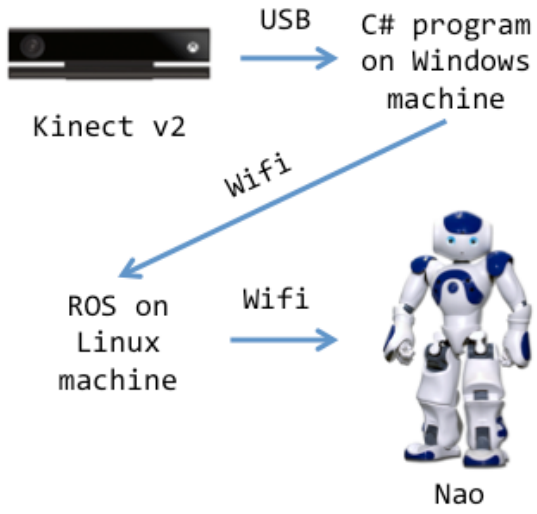


Figure 2. Overview of system architecture.

The Kinect provides sensing data to the Windows machine over USB. A C# program runs on the Windows machine to receive incoming data and process it further. Then, the C# program serializes processed data into JSON and sends messages to the Linux machine over a web socket connection.[ii]

When messages arrive on the Linux machine, they are received and published by

ROS onto a *rostopic*, or a named communication bus over which different ROS nodes can publish and read messages, named `/windows_rosbridge`. A ROS node parses messages on `/windows_rosbridge` before re-publishing the result onto different rostopics for each data type, e.g. `/face_info` and `/skeleton_info`. From there, other nodes can subscribe to these rostopics and use the data to perform further calculations and direct robots.[iii] See the technical documentation for further details.

Object Detection

The system uses color-based object detection to locate objects of interest in the Kinect's color stream. The Windows machine performs this step of processing using the color subtraction methods provided by the OpenCV image-processing library.[iv] Users must provide a color range in RGB for each object they would like to track; to aid users with this step, we developed the *RGB Thresholder* utility which is included in the codebase. To locate an object, the program subtracts all colors outside of the specified range from a color frame and finds the center of the remaining pixels. To obtain the coordinates of the object in 3D space, the program can then map this center pixel from the color frame to its corresponding pixel in the depth frame. This coordinate pair can then be sent to ROS as the position of the object.

It is important to note that the detection method described here requires a full color frame subtraction for each object, a step that is computationally expensive as the number of objects increases. The processing time required for this step exceeds the Kinect's frame rate when there is more than one object, requiring us to skip frames and update the object positions at a lower rate than 30 fps; see the evaluations section for more details. Optimization techniques exist that may mitigate this problem and are discussed in later sections as well.

Detecting Nonverbal Behaviors

Pointing

Human pointing gestures are often approximations of the true position of the target objects (Butterworth, 2000), so our perception system must be tolerant of these approximations and infer the most likely target object of a given pointing gesture. We can quantify referential ambiguity by way of two metrics: (1) the angle between the vector v_1 of a point gesture and the vector v_2 of the hand to an object, and (2) the Euclidean distance of the vector from hand to object. The angle describes how inaccurate a point is in referring to an object, and the distance describes how far the object is from the point gesture. The vectors and points described here are all three-dimensional; see Fig. 3 for a simplified diagram.

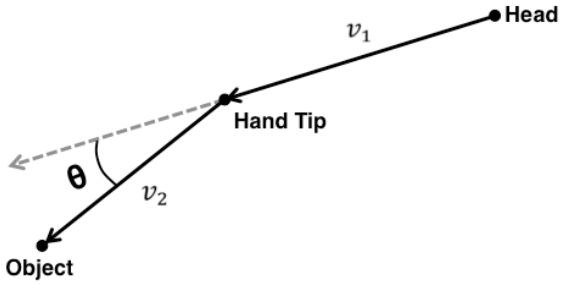


Figure 3. Vectors for computing point gestures and the angle θ between them.

We calculate the v_1 by component-wise subtraction of the head coordinate from the

hand tip coordinate. Both coordinates are 3D points provided by Kinect body tracking data. To calculate v_2, we also do component-wise subtraction, using the hand tip point and the position coordinate of an object.

Now, to compute the metrics, note that the distance metric is simply the magnitude of v_2. Computing the angle θ between v_1 and v_2 the dot product:

$$v_1 \cdot v_2 = x_1x_2 + y_1y_1 + z_1z_2$$
$$= |v_1||v_2| \cos\theta$$

$$\theta = \cos^{-1}\frac{v_1 \cdot v_2}{|v_1||v_2|}$$

Therefore, we can compute the angle by taking the arccosine of the dot product divided by the magnitudes of the two vectors.

Head Pose

The Kinect provides data on the yaw, pitch, and roll angles of the head through the Face stream. In the Kinect's coordinate system, pitch, roll, and yaw are rotations about the x, y, and z axes, respectively.[v] The visualization of rotation angle is provided in Fig. 4.

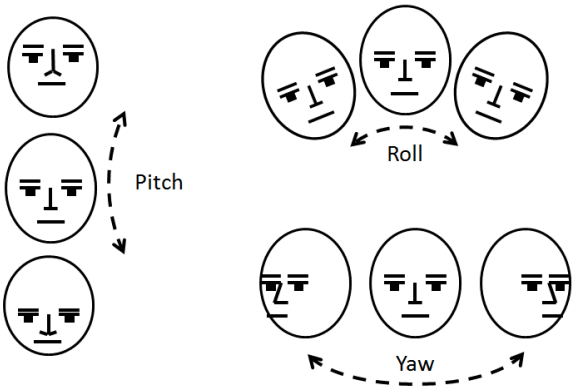


Figure 4. Pitch, roll, and yaw for head poses.

We apply trigonometric functions to these angles to calculate a vector describing head orientation.

$$x = \sin(yaw)$$
$$y = \sin(pitch)$$
$$z = \cos(yaw)$$

$$v_{head} = [x, y, z]$$

We can then get vector v_3 of the object to the head to compute the angle and distance metrics using the same calculations that we used for pointing gestures. See Fig. 5.

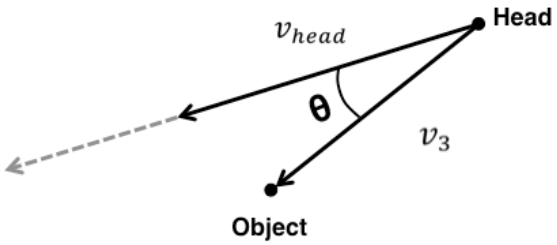


Figure 5. Vectors for computing head orientation vs. an object and the angle θ between them.

To mitigate false positives for both pointing gestures and head orientation towards objects, we set thresholds on the maximum acceptable angle for objects; any object whose θ falls outside of the accepted range is not considered a possible target of a

gesture. These thresholds are set on a case-by-case basis, since they depend in part on the scale and context of the interaction.

Note that this method of tracking pointing and head pose relies on knowing beforehand what the objects of interest are in a scene. In a more generalized version of this task where objects are not known in advance, more sophisticated techniques such as ray-tracing and point cloud analysis would be required to identify the targets of these behaviors.

Vocal Input

While vocal input does not fall under the category of nonverbal behavior, it is a common and intuitive paradigm for robot interaction so we implemented it in our system. We used the `Microsoft.Speech` library to listen for preset keywords using the Kinect's microphone array. As with other sensing tasks, the Windows machine sends a message over Wifi to Linux upon keyword recognition. While we did not execute rigorous tests on the performance and accuracy of vocal input, we found no issues from the addition of this feature and the system worked smoothly. Our implementation of vocal input indicates that detection of human nonverbal behavior does not preclude the incorporation of other communication paradigms.

Evaluations

Demo Video

We recorded a demo video to be included in our evaluations. The evaluative portion of the video is a minute and a half long, and it presents an integrated test of the system in which the Nao robot directs a user to point to and look at different colored blocks on a table. The goals of the test are as follows: (1) gauging the system's speed in detecting nonverbal behaviors in real-time, and (2) measuring the system's accuracy in identifying the target of a behavior.
Fig. 6 shows the setup of the test.

Figure 6. Setup of video evaluation. The Kinect and Nao sit on the large table, and the blocks rest on the smaller one. Note that the brown platform on the smaller table was removed before testing.

In the test, the Nao instructs the human to point to or look at an object, providing a six second window to perform each task. If the system detects that the human performed the correct action, it tells the Nao to continue to the next task; otherwise, it waits until the time limit was reached to do so.

The blocks are spaced evenly apart, except for the red and blue blocks, which were contiguous. The blocks were color segmented using the *RGB Thresholder* utility in preparation for testing. For the ranking of target objects, we only used the θ metric in this evaluation, so the object with the smallest θ was deemed to be the target of

a gesture. The thresholds of maximum acceptable θ were set to 0.35 radians and 1.5 radians for pointing and head pose, respectively.

Of the ten tests performed in the evaluation, one case was a false positive and one was a delayed response. Both of these cases occurred in the tests for head orientation, which is a more ambiguous movement than pointing. The false positive was triggered by the fact that the user was already looking in the direction of the block the Nao requested. The perception system consequently determined that the requested block had the smallest θ, even though the user was not actively looking at the block. The delayed response, on the other hand, was a misdetection that the system corrected in a few seconds. Otherwise, the gestures and their target objects were detected correctly and instantly.

Note that the video evaluation fails to demonstrate cases where the user points to an object not requested by the Nao. Such cases did not cause a problem in less formal tests we performed. Let it also be clear that we take these tests to be basic evaluations, so any application of the system in more specific settings requires further testing and tailoring of the system to the intended use. However, this simple evaluation does demonstrate that this perception system is feasible and is worth researching or developing further.

Processing Objects Test

In addition to the tests performed in the demo, we also tested the system for its tolerance regarding number of objects. As mentioned previously, the amount of processing time between frames limits the number of objects the system can support, as each object requires an expensive call to the OpenCV color subtraction method on the entire 1920x1080 frame. To mitigate this problem and provide the color subtraction routines with more processing time, we perform object detection on every fourth frame. To evaluate this mitigation method, we measured the performance of the system with different numbers of objects: all trials used the same test video in which eight distinct colors were present, but in each trial the program was configured to recognize a different number of the colors. We used the "`rostopic hz`" command to measure the publishing rate for the trials. See Fig. 7 for the results.

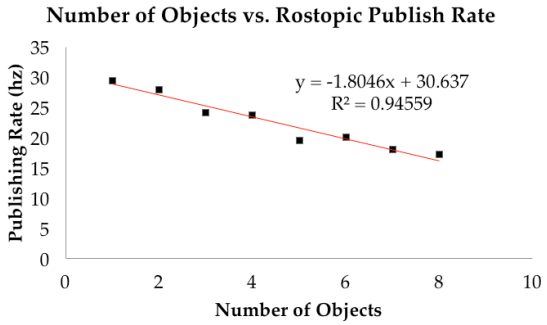


Figure 7. Graph of objects vs. `/objects_info` publish rate for a system segmenting objects every four frames.

The graph shows a linear decrease in publishing rate as the number of objects increases, so that by eight objects we drop ten of thirty frames on average. Better optimizations may be able to address the issue more effectively. For example, writing the image processing code in C++ instead of

managed C# code or using more efficient algorithms that make use of techniques such as template matching are both promising possibilities.

Conclusion

In this paper, we have described a perception system that can recognize the nonverbal behaviors of pointing and looking at objects, through the proxy of head orientation. The system uses the recently released Kinect v2 for sensing, and ROS for message processing and robot control.

The evaluations of the system demonstrate that it is able to detect gestures and their targets with good accuracy and in real time. The evaluations also revealed that the system's primary performance limitation is its color-based object detection routine.

Future Work

The perception system described here is a proof-of-concept system, so there is a fair amount of extensibility for the project. One possible extension would be to develop a version of the system for sensing in tutoring applications, or other areas of research in HRI.

The Yale Social Robotics Lab is exploring the possibility of using the system in experiments on the computational modeling of object references. Henny Admoni's recent work in submission pertains to this subject. The current implementation of the system can only detect individual gestures and identify their targets—aggregating these individual behaviors into a higher-level model is a logical next step.

Acknowledgements

I would like to thank Professor Scassellati and Henny Admoni for advising me on this project. Thanks to Sam Spaulding and Alex Litoiu for their help navigating existing codebases. Thanks to Rachel Protacio as well for this document template.

References

[1] Jungong Han; Ling Shao; Dong Xu; Shotton, J., "Enhanced Computer Vision With Microsoft Kinect Sensor: A Review," *Cybernetics, IEEE Transactions on* , vol.43, no.5, pp.1318,1334, Oct. 2013

[2] Butterworth, G. and Itakura, S. (2000), How the eyes, head and hand serve definite reference. British Journal of Developmental Psychology, 18: 25–50. doi: 10.1348/026151000165553

[3] Admoni, Henny. Modeling communicative behaviors for object references in human-robot interaction. *Work in submission*.

---

[i] For more on the differences between the original Kinect and the Kinect v2, see http://blogs.msdn.com/b/kinectforwindows/archive/2014/07/15/the-kinect-for-windows-v2-sensor-and-free-sdk-preview-are-here.aspx

[ii] Latency over the wifi network does present a small concern for real-time systems, but under stable, closed conditions, this should not pose a problem.

[iii] All ROS nodes in our implementation are written in Python. For more information on rostopics, see http://wiki.ros.org/Topics

[iv] The OpenCV library we used was a port for C# called OpenCvSharp.

[v] For more on the Kinect's coordinate system and head pose angles, see

https://msdn.microsoft.com/en-us/library/jj130970.aspx. Also the source of the image.