**Outline of Research Goals**

We envision the final setup consisting of a fixed sensor overlooking a table-sized interaction space. This sensor will aid in recognizing a set of predefined gestures, from which we can continue processing accordingly, whether this is to identify the object being pointed to, the point to move to, or some other domain-relevant task. Ideally, this gesture set can begin small and expand as needed to make collaboration simpler, building on previous research in this field to expedite the actual work needed to implement gesture recognition.

On that note, we outlined a few intermediary research goals that could serve as targets to move towards as we explore the technologies and techniques involved in this domain.

1. Identify other gestures of need.
2. Recognize a single pointing gesture.
3. Process a single pointing gesture to identify the area referenced.
4. Recognize an object that is pointed to.
5. Research and implement other gestures identified in (1).
6. Integrate into the existing UIMA pipeline.

These 6 objectives ramp up in difficulty quickly and may possibly expand beyond the work that can be completed this year. We do note that the 2nd through 5th objectives are closer to the domain of image processing than machine learning and thus may allow us to circumvent the associated volumes of data needed for training. Regardless, these objectives are entirely arbitrary and may be adjusted as stakeholders identify other, more promising directions to take our research. In particular, if possible grants necessitate the pursuit of other work then doing so seems more fruitful than pursuing the above outline. In either case, some form of gesture processing must be able to address the above 6 objectives in some fashion, making them ideal targets to segment our work moving forward.

**Camera**

In previous research ventures, those performing gesture recognition considered two primary factors of camera quality in efficacy and cost. These two factors are fairly obvious qualifiers of camera performance, but it is efficacy that naturally lends itself to further discussion. In particular, while we could find research efforts that used pretty much any device imaginable, the vast majority seemed to rule that depth sensors are the technology of choice in this domain. To consider why depth sensors appeared so prominently in various literary sources, it is worth considering the options that exist in this domain.

When imagining a robot configuration in a shared space, one’s natural inclination is to imagine a single camera focused on a static collaboration space. Such a setup is fairly trivial to configure, as there is minimal need for configuration. Rather, it is as simple as setting up the camera, orienting or calibrating in some fashion to obtain a sense of direction, then processing as programmed. However, the obvious drawback of this system is the lack of angled views. This loss may not seem like much of an issue, but it is difficult to construct depth information from a single camera angle. As alluded to in the processes used for gesture recognition, depth plays an integral part in many of the existing gesture recognition techniques, making it difficult to justify the usage of a single camera usage for gesture recognition alone.

The natural evolution from one camera is to use multiple cameras, in particular stereo cameras, to create a more “human” field of vision. What this approach gains over single cameras is the ability to utilize the multiple views to compute depth information. Naturally, obtaining this depth information requires merging the images from each of the camera into once “usable” result, a process that is not computationally trivial. Furthermore, since these cameras operate independent of one another and have feeds that are merged into one view through processing, minute adjustments to one camera can disorient the entire system. Hence, calibration is particularly important with this setup; consequently, it is difficult to utilize this setup anywhere other than a well-lit, static place. While these are constraints that can certainly be imposed to our setup, they are not ones that should be assumed unnecessarily.

At a surface level, depth sensors combine the ease of use for single cameras with the depth information provided by stereo cameras. Unlike standard cameras, a depth sensor forgoes color information in favor of a 3D depth map, one whose accuracy tends to beat out the computed values provided by multiple stereo cameras. Many depth sensors use some form of infrared emission followed by infrared detection, which is combined with various tricks to obtain depth information. One existing methodology is simply to project a grid of dots with the emitter, then to use the differences in spacing of the dots to determine depth information. More recently, emergent technology uses time-of-flight technology, which instead considers the time delay between emission and detection, as opposed to geometric distancing. These variations in depth sensors have various tradeoffs that are tied more to the individual devices using these technologies, as opposed to the techniques themselves.

Across depth sensors, the reliance on the “emission-detection loop” means that most depth sensors have fairly short effective ranges, falling prey to noise beyond several meters of operation. This tradeoff is one that may limit many practical uses, but is not one that is too concerning for shared space collaboration. A defined, static working space would likely be no larger than a table’s worth of space, which is easily in the range of detection for a depth sensor. In addition, while cost was not a factor considered much with the other cameras, it is of note that many depth sensors are relatively cheap (in the range of hundreds compared to thousands of dollars for industrial stereo cameras), making them especially practical for research.

**Camera Verdict**

Certainly, regardless of model, a depth sensor makes logical sense for our use case. When it comes to identifying blocks, we are not sure of the limitations that we will find using a depth sensor in isolation. Perhaps it will be necessary to use both a depth sensor and a stereoscopic camera configuration to identify both blocks and gestures, but for gesture recognition alone it is hard to justify the use of any alternative in lieu of a depth sensor.

When looking at individual models of depth sensors, a surprisingly large number of research papers utilized the Xbox Kinect. While this device is not particularly robust, it is uniquely cheap and is sufficient for effective, albeit not optimal, gesture recognition. Indeed, one review paper from 2012 reports that the Kinect “is used by 22 of the [37] papers reviewed” (Suarez and Murphy). While the Kinect was designed for full body recognition for gaming, it is paired with an SDK that affords the flexibility to further analyze depth information to isolate gestures. The usage of depth sensors and the Kinect in particular has spurred collaborative efforts to make interfacing with these devices easier, providing alternatives to the stock SDK such as OpenNI that may make it even easier to consider hand gestures.

The most noteworthy alternative comes at Dr. Shibberu’s suggestion with the Intel RealSense Depth Camera D432. This camera has its own SDK and efforts to make this camera interoperable with prominent open source alternatives like OpenNI are ongoing. However, the camera itself has definite advantages with respect to hardware (see<https://click.intel.com/intelr-realsensetm-depth-camera-d435.html>), which makes balancing the hardware advantages with the potential software losses a definitive tradeoff between the two models.

**Resources**

1. Coupeté, Eva, et al. “Gesture Recognition Using a Depth Camera for Human Robot Collaboration on Assembly Line.” *Procedia Manufacturing*, vol. 3, July 2015, pp. 518–525.
   1. <https://www.sciencedirect.com/science/article/pii/S2351978915002176>
2. Suarez, Jesus, and Robin R. Murphy. “Hand Gesture Recognition with Depth Images: A Review.” *2012 IEEE RO-MAN: The 21st IEEE International Symposium on Robot and Human Interactive Communication*, 2012.
   1. <https://www.researchgate.net/publication/236160250_Hand_gesture_recognition_with_depth_images_A_review>
3. Liu, Hongyi, and Lihui Wang. “Gesture Recognition for Human-Robot Collaboration: A Review.” *International Journal of Industrial Ergonomics*, vol. 1, no. 13, Mar. 2017
   1. <https://www.sciencedirect.com/science/article/pii/S0169814117300690>

**Generating Simulated Camera Data**

If we were to take a machine learning approach to gesture recognition, one of the primary problems we would face would be acquiring data to train and test our algorithm. Since the data would very likely be videos of a person, or potentially just their arm, gesturing at objects on a table, it seems infeasible to generate this data manually. Discussed below are two potential alternatives to manual data generation.

One video of a gesture could be used to create multiple pieces of training data by creating a point cloud out of the footage, and using [Point Cloud Library](http://docs.pointclouds.org/trunk/index.html) to simulate video of the same gesture from multiple angles. The main page on this software states that “PCL supports natively the [OpenNI](http://www.openni.org) 3D interfaces, and can thus acquire and process data from devices such as the [PrimeSensor](http://www.primesense.com/?p=514) 3D cameras, the [Microsoft Kinect](http://www.xbox.com/kinect), or the [Asus XTionPRO](http://us.estore.asus.com/index.php?l=product_detail&p=3397)” (<http://www.pointclouds.org/about/>).

One major pro of this method is that that many technologies which PCL supports align with those we have begun investigating for our own project. Some cons are that it still requires at least some manual recording of camera data and that potentially the additional simulated angle would not be robust enough to account for another person than the one who generated the training data moving in a significantly different manner.

For the second approach, if a virtual simulator is built for gesturing at a set of objects, in some 3D framework such as Unity, some randomization can be applied to the simulated motion to expand the size of the training set greatly. Transfer learning can then be used to train a neural network to recognize gestures at objects in the real world based on this trained network. (<https://arxiv.org/abs/1703.06907>)

Use of Unity3D specifically for this has been tried in the paper [here](https://projekter.aau.dk/projekter/files/281077760/THESIS_Transferring_Deep_Reinforcement_Learning_from_a_Game_Engine_Simulation_for_Robots.pdf) and while it was effective as reinforcement learning in the simulation, it proved to be insufficient for the robot to actually perform the task. This may mean that this method is not completely sufficient for our task, but gesture recognition requires slightly less precision than controlling a full robot arm, due to the absence of variables such as grip angle and grip pressure. While this method seems promising, the major con is that a lot of development time would have to be devoted to the simulation, which would not produce any useful artifacts for the project in its intermediate stages.

**Existing Gesture Recognition Techniques**

For the research on gesture recognition, I tried to mainly focus on papers that specifically dealt with pointing, since pointing is a little unique in gesture recognition, and is recognized as a challenge. For one, we not only need to interpret the hand gesture, but also determine where it is pointing. One of the articles I was going through (though I could only read the abstract for some), utilized a projection from the 3d camera view to a 2d construction of the planar surface that the user would point at. This would enable a construction of a top-down view. Furthermore, that paper utilized information about the placement of joints to construct vectors, and used vector calculations to obtain the final information as to where the user is pointing. [1]

Another paper used binocular stereo vision to determine how the user was pointing. Unlike the previous paper, the planar surface would not have to be in view for the program to extrapolate where the user was pointing. Unfortunately, I could not get any information past this point as I could not get access to the full article. [2]

Another approach is to utilize Hidden Markov Models to determine the pointing direction. In particular, I think the paper dealing with this is very relevant to our project, since a robot utilizing this technique would be able to identify the object that the user is pointing at, and I think determining an object through gesture recognition might be more useful than generating coordinates or some other type of output. This approach also utilizes a stereo camera. Essentially, the authors of this paper tracked the hand and face of the user with 3-D particle filters, and a HMM would take the position of the hand and try determining a set of coordinates that the hand is pointing at. Then, a pointing direction is estimated from these coordinates. A problem this approach suffers from is that it requires at least a view of the human from the waist up. [3]

Visage Technologies also provided information about some Latent-Dynamic Conditional Random Fields that it utilized in conjunction with Microsoft Kinect to get accurate pointing information. I’m not too familiar with this approach, though it seems that it is similar to a neural network in that variables are inferred rather than supplied. [4]

Another paper provides more insight into this technique. The approach utilized essentially uses Microsoft Kinect with OpenNI to get a skeleton of the user. This information was then passed through a LDCRF to obtain the pointing direction. What I found from this paper is that we still need to train a LDCRF. The authors of this paper recorded 990 gestures as an evaluation dataset. [5]

All of the approaches above require a large view of the human to determine the pointing direction. Another paper describes how we might determine pointing direction if only given the image of the hand. Initially, the approach utilizes background subtraction to filter the pointing object. Then, the pointing tip is found. The authors utilize several heuristics to determine the pointing tip, such as distance from the border of the image, being on the edge of the pointing object, etc… It is possible that we could also use a neural network to determine this. Unfortunately, this paper does not go into detail about how to extrapolate a pointing direction from this information, but given the pointing tip and the pointing object, we should be able to compile some heuristics to determine the direction of the pointing vector. [6]

From the research papers online, the approaches utilized make me think that deep learning is probably not the route we should take when constructing an algorithm to determine pointing direction/coordinates. Instead, if we want to utilize a machine learning model, looking into Hidden Markov Models, and Latent-Dynamic Conditional Random Fields would be best. (Note that the LDCRFs seem to perform better, but I don’t know nearly as much about them). Furthermore, we could avoid machine learning altogether by utilizing background subtraction and some heuristics (that we will obviously need to work on) to determine pointing direction. This will involve vector calculation. For a machine learning model, we will need to obtain training data. Most of the papers I read seemed to depend on the coordinates of the hands and face to determine pointing direction. However, I think that we can simplify this to utilize position of finger joints, so that we don’t need a full view of the person to know where they are pointing. Obviously, this will need some work, since joint recognition is not as easy as face recognition.

Links to documents:

1. <https://ieeexplore.ieee.org/document/7531881/>

2. <https://ieeexplore.ieee.org/document/4593304/>

3. <https://www.sciencedirect.com/science/article/pii/S0262885610001149>

4. <http://visagetechnologies.com/point/>

5. <https://arxiv.org/pdf/1510.05879v1.pdf>

6. <https://link.springer.com/content/pdf/10.1007%2F978-3-642-36546-1_37.pdf>

**Vision Libraries**

The most popular vision libraries that I found to be worthy of being considered are: xbox kinect sdk, OpenNI (with NITE as a middleware), and OpenCv. Even then, among these three, I focused my research between the kinect sdk and OpenNI. However, as an aside, I did find OpenCV to be more flexible with finding contours and convex hulls. OpenCV also has a reliable face detector using Violoa-Jones object detection method. With the head location, the candidate hand location can then be estimated for further gesture processing.

Both the Xbox Kinect SDK and OpenNI provide fully articulated 20 point body tracking with nodes for each hand. The pros regarding OpenNI, and con for xbox Kinect, is that NITE body tracking is more popular in the research community. This means a better support in the long run. NITE also provides waist-up body tracking which would be useful for sitting down. However, NITE does come with a con in that unlike Kinect SDK, NITE requires a calibration pose to initialize body tracking. This was the extent of information I found pertaining to any vision libraries.

The first camera that is considered is the Xbox Kinect. One of the big selling point for the Kinect is its low cost. The Kinect also comes with software libraries to perform hand and body tracking. However, in many of the paper, it is not clear whether these libraries are being used or replaced by custom algorithms. The Kinect includes a QVGA (320x240) depth camera and a VGA (640x480) video camera. The QVGA is able to produce image streams at 30 frames per second. It works on the principle of structured light in which an IR light emitter produce a non uniform array of dots to detect depth. Unfortunately, there are downsides: the QVGA is limited by near and far thresholds for depth estimation with a range of 1.2 to 3.5 meters and it also does not function well in bright sunlight since the strong light in the environment easily drowns out the IR projection. The VGA video camera is able to produce image streams at 30 frames per second. One popular combination that many use is using the Kinect as a depth sensor for gesture recognition and OpenNI and NITE middleware for hand tracking.

Other popular depth sensors that are considered are Time of Flight (ToF) cameras, stereoscopic cameras, and single cameras. One example of a ToF camera is the SwissRanger from Mesa Imaging. ToF cameras determine pixel depth in one of two ways. The first way is to measure the round-trip flight time of light projected onto the scene and reflected back to the sensor. The second is the measure the phase-shift of the reflected light. As a result, ToF cameras are able to produce accurate depth images at 50 frames per second. Unfortunately, the trade-off is its low resolution of 144x176.

The third type of camera to consider is stereoscopic cameras. It’s a multi camera setup where two (or more) simultaneous images are captured from a pair of calibrated video cameras. Image registration methods are used to create a disparity map that estimates per-pixel depth. However, this leads to lower-fidelity depth images than ToF cameras. Since multiple images are taken, it also means more processing is required and more computational overhead to solve the image registration problem for each image pair. Nevertheless, stereoscopic cameras work well in bright light and can be built with standard video cameras. An example would be the Bumblebee from Point Grey.

**Miscellaneous Notes and Hardware Specs**

OpenNI & Xbox Kinect -

* Low Cost!
* Most of the papers are on hand tracking/gestures
* Most popular depth sensor used is Microsoft Kinect
  + Comes with software libraries to perform hand and body tracking
  + However, not clear whether these libraries are being used or replaced by custom algorithms
* Kinect includes:
  + QVGA (320x240) depth camera
    - Produce image streams at 30fps
    - Also used in ASUS Xtion Pro (developed by PrimeSense. Same company developed by group lead by PrimeSense)
    - Works on the principle of structured light
      * IR light emitter produce non uniform array of dots to detect depth.
    - Limited by near and far thresholds for depth estimation
      * Range is ~1.2 to 3.5 meters
    - Does not function well in bright sunlight
      * Drowns out the IR projection
  + VGA (640x480) video camera
    - Produce image streams at 30fps
  + PrimeSense also published NITE, a middleware module for OpenNI to perform body tracking and hand tracking
    - CLOSED SOURCE
  + People have used Kinect as depth sensor for gesture recognition and OpenNI and NITE middleware for hand tracking.
* Developed for full-body motion tracking
* OpenNI framework (via NITE middleware) and Kinect SDK provide fully articulated 20 point body tracking with nodes for each hand.
* NITE body tracking is more popular in the research community
* NITE provides waist-up body tracking (useful for sitting down)
* NITE con- unlike Kinect SDK, require a calibration pose to initialize body tracking.

Other popular depth sensor types are:

* Time of Flight (ToF) cameras
  + SwissRanger from Mesa Imaging
    - Determine pixel depth in one of two ways:
      * Measuring round-trip flight-time of light projected onto the scene and reflected back to the sensor.
      * Measuring the phase-shift of the reflected light
    - Produce accurate depth images at 50fps
    - However at low resolution (144x176)
* Stereoscopic cameras
  + Capture two simultaneous images from a pair of calibrated video cameras
  + Use image registration methods to create a disparity map that estimates per-pixel depth
  + However, produce lower-fidelity depth images than ToF cameras
  + Also require computational overhead to solve the image registration problem for each image pair
  + Work well in bright light
  + Can be build with standard video cameras
  + Example of stereo cam for commercial use is the Bumblebee from Point Grey
* Single camera
  + Con- Needs carefully controlled lighting for the scene
  + Can use shadow analysis to infer depth information
  + Need multiple light sources anto detect depth discontinuities

Depth cameras vs Color cameras

* Hand Segmentation
  + Depth Camera
    - Simple to use in a situation where the user is expected to face the depth camera and hold their hands out in from of themselves for gesturing
      * Easy to isolate the hands
  + Color Camera
    - No depth information
    - Available skin-color maps
    - Cascaded classifires on Haar-like features
    - Con- weakness in lighting changes

<https://www.researchgate.net/publication/236160250_Hand_gesture_recognition_with_depth_images_A_review>

OpenCV -

**Easy to detect hand gestures**

* Parameters:
  + Hue (of skin)
  + Lightness
  + Saturation
* Finding Contour and Convex Hull
* Use cv.partition for clustering
* Detect fingertips using defect points
* Available face detector
  + Violoa-Jones object detection method
  + Determine the head location and then can estimate candidate hand locations.

<https://medium.com/@muehler.v/simple-hand-gesture-recognition-using-opencv-and-javascript-eb3d6ced28a0>

Neural Networks and Support Vector Machines are commonly used for pose and gesture recognition.

* Con- requires minimal preprocessing fo data
* SVM are notoriously difficult to implement

Hand segmentation is most commonly accomplished using depth thresholding or region growing techniques. Hand tracking is done using Kalman filters and mean shift. Another alt is to use the NITE body and hand tracking module for OpenNI rather than implementing our own. Gesutre classification can be done with Hidden Markov Models, K-NN, ANN, SVM, and FSMs.

**Notes**

* Normal vision based gesture recognition is composed of four steps of model initialization:
  + Tracking
  + Pose estimation
  + Gesture recognition
  + Classification
* Most feature extraction algos work on images from a single camera, but many setups use stereoscopic vision from two or more cameras
* However stereoscopic vision requires higher processing speed of the hardware
* Gesture recognition tracking consist of two processes of *figure-ground segmentation* and finding *temporal correspondences.*
* Figure-ground segmentation:
  + Background subtraction
  + Motion based
  + Appearance based
  + Shape based
  + Depth based segmentation
* Gestures can be viewed as sequence of postures in time, the most widely used classification method is a Hidden Markov Model.

**This is good stuff for gesture recognition theory:**

<https://diuf.unifr.ch/main/diva/sites/diuf.unifr.ch.main.diva/files/joomla_reportForrer.pdf>