Robot Recognition of Military Gestures
CS 4758 Term Project

Garrett Bernstein (gsb29) - CS M.Eng '11
Nyk Lotocky (njl36) - CS M.Eng '11
Dan Gallagher (drg86) - CS '12

***Abstract*— The goal of this project is to have the telepresence robot recognize and follow military hand signals. These signals consist of commands such as Halt, Crawl Forward, Run Forward, Retreat, Flank Left/Right, etc. Upon recognizing a gesture the robot will execute a hardcoded response. The robot will be equipped with a Kinect sensor that detects the skeleton of a human. We have also coded a baseline classifier that treats each gesture as a stationary pose and then uses a geometrical KNN algorithm to predict the gesture. The baseline does very well at classifying gestures that are actually stationary and does passably well at classifying gestures that actually involve motion. Our actual system uses the fact that all gestures are relatively sinusoidal and extracts relevant features such as frequency and amplitude that are then used in an SVM to classify the gesture. We are able to consistently classify gestures with over 96% accuracy in randomized offline testing and the robot can recognize all 30 gestures on both arms with our live algorithm.**

I. Introduction

By definition war is very dangerous for those involved. Even on routine patrols troops are at risk of attack from adversaries. Incorporating capable robots to work with and instead of troops in as many tasks as possible would greatly reduce the number of human casualties suffered. Troops operate in very dynamic environments, however, and operating and interacting with a robot companion via some sort of controller is unwieldy and cumbersome. Troops themselves interact with a set of defined hand gestures that each convey a specified meaning or command. Ideally any mobile robot companion would also be able to recognize and respond to those hand gestures. We explore this goal by attaching a Kinect to a telepresence robot and implementing hand gesture learning algorithms.

As the Kinect is a relatively recent product, few, if any, academic papers have been published on the topic of gesture recognition with the Kinect. There have been many "hackers" who have tackled projects of interest with the Kinect and posted them to the Web. One example is a Toronto hospital using the Kinect to control the viewing of MRI images. [1] Another example is a hacker using a Kinect to control basic functions of a television such as changing channels and volume. [2] Google just recently released Gmail Motion, a gesture-controller for email, but that is out of the scope of this course. [5] We found these examples lacking, however, in that the gestures performed and recognized are very rudimentary. The MRI-controller only detect linear movements, e.g. To zoom in on an image a doctor would move their hand towards the screen. The TV-controller only detects the user's hand extending to a general location, e.g. To increase the volume the user raises their hand above their head. Troops must interact with robots in a much more dynamic setting requiring much more communication so we wish to implement gesture recognition for much more complex gestures.

II. Approach

A. *Pose Representation*

After examining multiple documents on military gestures we decided to limit our scope to just analyzing the motion of the right elbow and hand as the rest of the body is not usually involved in the motion. [3] [4] The recorded Kinect data gives global (x, y, z) coordinates for joints, as shown in Figure 1. People's bodies all have different proportions, so to negate that effect as much as possible we decided to use the orientation of the hand and elbow with respect to the shoulder.

Our first task was to define a representation for the pose of an arm. Because the elbow is constrained by the upper-arm and the hand is constrained by the forearm, the location of each joint only has two degrees of freedom. Therefore we can use the spherical angles of the elbow with respect to the shoulder and of the hand with respect to the elbow to fully describe a pose. The tuple would then be $(\theta_{elbow}, \phi_{elbow}, \theta_{hand}, \phi_{hand})$.

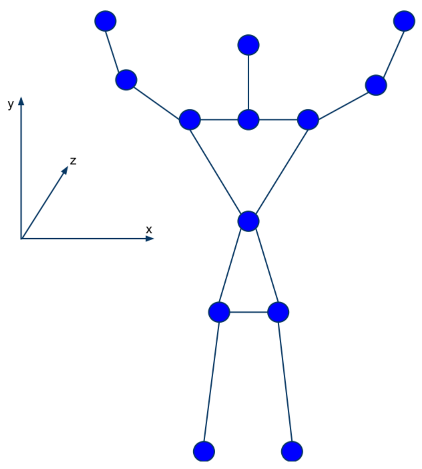

Fig. 1. Kinect Skeleton

B. Stationary Gestures Baseline

We began by creating a baseline algorithm that treated everything as a stationary gestures. This meant that even if the gesture involved motion we pretended it was stationary. We recorded data of a gesture being executed over a 5-second period at 50 frames per second. Because we treat all gestures for this baseline as stationary, instead of using every frame we throw out the first and last second and then pick three frames at random from each of the middle three seconds. We use $\frac{2}{3}$ of the data files of each gesture for training data and withhold the rest for testing. The training data is just a conglomeration of all the randomly chosen frames, i.e. there is no concern for which data file the frame came from. The test data does keep frames from each data file grouped so that they can be used together to predict the gesture of that file.

Given the three feature-tuples for a datafile, we run 3-KNN based on Euclidean distance for each tuple with all of the training data. We use majority-rule with arbitrary tie breaking, so that if at least two of the nearest neighbors agree on a gesture then the tuple adopts that gesture, and if all three neighbors disagree one is chosen at random. Once all three tuples for a data file have adopted a gesture we execute a second round of voting. This is the same majority-rules voting used in KNN but if no consensus is reached among the three tuples we say the datafile does not adopt a gesture and automatically fails classification.

C. First Aproach: KNN

Feature extraction begins a list of the angle 4-tuple discussed in Section II-A over a span of some number of seconds. Training files are 5 seconds long and live data is parsed every time the buffer fills. This leads to a lot of extra data points that do not provide any useful information about the gesture. To avoid this we iterate through the data and extract points that are relevant in the sense that they are some threshold distance away from the previously recorded point. In practice this throws away approximately half of the data points and nicely sketches out the gesture. We call this cleaned up data the *splitDP* of the gesture. To account for data points in static gestures being well under the distance threshold from each other we added in a time threshold, such that if we have not extracted a point within the time threshold we extract a point. In practice this gives us approximately 6 data points for a stationary training gesture, which in essence gives us the same data for stationary gestures we were using for our baseline.

Next, this series of data points is treated as a function in four dimensions. To determine the "distance" between training and test data, the test function was cross-correlated with each training example, and divided by the product of the magnitude of the two functions. This similarity value was between 0 and 1, and increased as functions became more and more similar, so when subtracted from 1, served as a monotonic distance function. Using this distance function, the same voting scheme as from the baseline KNN system was applied to determine the resulting classification.

D. Actual System

1) Feature Extraction: Feature extraction begins with a list of the angle 4-tuple discussed in Section II-A over a span of some number of seconds. Training files are 5 seconds long and live data is parsed every time the buffer fills. Once we have the *splitDP* we can extract our actual features to describe the gesture. We take advantage of the fact that the dynamic gestures are all sinusoidal in nature. For each of the angles in the list of 4-tuples we extract the same 4 features: frequency, amplitude, mean, standard deviation. The static gestures are clearly not sinusoidal, and not every theta of each dynamic gesture is sinusoidal, but these four features prove to be quite effective at describing all types of gestures.

- *Frequency*
  This is an estimate of the sinusoidal frequency of the angle. First we find the local maxes of the function by finding every data point that is greater than its closest neighbors on either side. We then take the average of the distances of consecutive local maxes to be the approximate frequency.
- *Amplitude*
  This is an estimate of the amplitude of the angle. We find the local maxes and mins as described in the item above. Then we find difference of the average of local maxes and average of local mins as our estimated amplitude.
- *Mean*
  This is simply the mean of the theta value. We include this to ensure that the position of the gesture is not lost.
- *Standard Deviation*
  This is a modified calculation of the standard deviation. We use the formula

  $$\sqrt{(\theta_1 - \mu)^2 + (\theta_2 - \mu)^2 + \cdots + (\theta_n - \mu)^2}$$

  Notice that the division by $\sqrt{N}$ is omitted. Doing so allows this feature to encode two pieces of information. Remembering that the *splitDP* has under 10 data points for static gestures and over 200 for dynamic, omitting the division by $\sqrt{N}$ causes this feature to easily discriminate between static and dynamic. Then, within each of the two groupings, this feature encodes how far from the rough mean of that cluster the gesture travels. In other words, this tells us whether a dynamic gesture has small or large movements of the hand.

2) Classification: For the actual system we employ the SVM-Multiclass toolkit by Thorsten Joachims [6]. We have the four features discussed in Section II-D.1 for each of the angles in the 4-tuple gesture descriptor, giving a total of 16 features. Training SVM-Multiclass outputs a model file. We use a Radial Basis Function kernel with a c value of 0.3. We also experimented with using a Linear kernel but that proved to be much slower for training and less accurate for testing. Testing a gesture instance on that model then outputs a predicted gesture name. When testing on stored gesture files the predicted gesture can easily be compared to the known gesture to determine accuracy.

3) Left/Right Arm Classification: Taking advantage of the symmetry of the human body we have added the capability to execute gestures with both the right and left arm. Instead of training a whole new dataset with the left arm we simply take the left arm coordinates recorded by the Kinect and mirror them across the chest. This way the left arm can be used to gesture and our classification algorithm can simply compare it to the right arm-based database.

4) Running Live On Robot: To run our classifier live on the robot we pull data directly from the Kinect. We let the buffer fill with 4 seconds worth of data and then classify that for both arms. After every classification we then pause for 2 seconds before classifying again, allowing for a 2 second overlap in the data. Pausing allows us to classify separate cycles of our “sinusoidal” functions. Because this method entails a person standing in front of the robot for an indefinite period of time, and the person may not always be executing a desired gesture, we need to make sure our classification algorithm is very confident it has observed a known gesture. Therefore, while running live we ensure we have observed a known gesture by not outputting a final classification immediately. Instead, we wait until we have observed 3 consecutive intermediate classifications of the same gesture before outputting that gesture as our final prediction. We classify each arm separately and the three consecutive intermediate gesture classifications must come from the same arm. We decided to be extra conservative with classifying live gestures because in a military setting it is worse to issue a wrong command than no command at all. Therefore, it is worth taking slightly longer to ensure a correct gesture prediction.

5) Robot Commands: Given a final gesture prediction we then need to tell the robot to execute the desired response. For each gesture we hard-coded a set of (x, y) waypoints relative to the current position for the robot to visit that would create a motion indicative of an appropriate response to the gesture. Given a sequence of waypoints we execute feedback linearization to determine the necessary velocity and angular velocity to reach the next waypoint. The robot then integrates its odometry measurements to determine how close it is to the target waypoint, stopping motion when it is close enough to the goal. A small sample of our actions include:

- **Come -** The robot drives fowards a meter.
- **Sneak -** The robot slowly weaves forwards in a jagged motion, simulating a stealthy approach.
- **Listen -** The robot stops moving and plays a sonar noise, to simulate using its sensors to listen.
- **Shotgun -** The robot jerks back and forth, as if it had just fired a shotgun.
- **Backup -** The robot slowly drives backwards.
- **Cover -** The robot patrols in a square around the starting location, to ensure that the area is secure.

III. Data

Our database consists of 30 separate gestures executed on the right arm, including 15 static and 15 dynamic. These gestures were chosen from already established military and flight controller handbooks. [3] [4] For each gesture we have 3 people recording 10 examples of 5 seconds each, for a total of 30 examples. The picture guide for the static gesture "Abreast" (Figure 2) and the dynamic gesture "Cover" (Figure 3) are provided.

We also include a static "Anti-Gesture," which entails the person standing at ease with their arms by their sides. This proves to also pick up if the person is resting their hands on their hips. If this gesture is observed we do not send any command. This allows us to stand at ease in front of the robot without worry of accidentally sending a command.

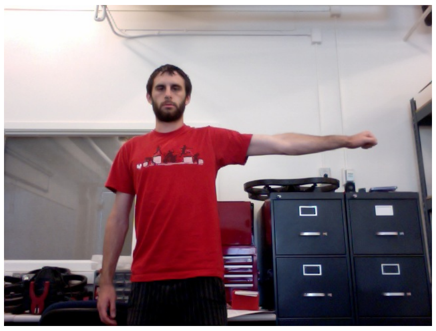
Fig. 2. Static Gesture "Abreast"

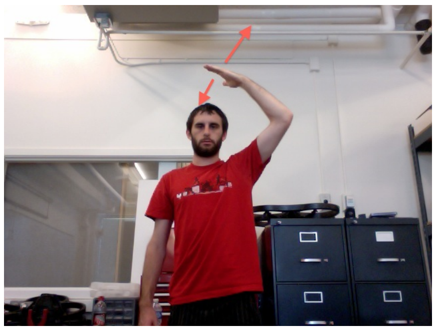
Fig. 3. Dynamic Gesture "Cover"

We have uploaded this database to Professor Ashutosh Saxena's site for public use. [7] We include pictures of each gesture and videos entailing the motion of each gesture for reference. We also provide two Python scripts to process the data into desirable formats. *extractFeaturesFromFile.py* parses the data received from the Kinect into (x, y, z) coordinates of the right shoulder, elbow, and hand. Then, *extractSVMFeaturesFromList.py* parses the (x, y, z) data into the theta 4-tuple describing the angles of the elbow in relation to shoulder and hand in relation to elbow.

TABLE I
STATIC GESTURE ACCURACIES FOR OFFLINE TESTING

Gesture	Accuracy
Abreast	1.0
Enemy	1.0
Freeze	1.0
Hide	1.0
Injury	1.0
Pistol	1.0
Rifle	1.0
Stop	1.0
Unknown	1.0
Antigesture	0.99
Backup	0.95
Land	0.93
Gas	0.90
Watch	0.85
Listen	0.74

TABLE II
DYNAMIC GESTURE ACCURACIES FOR OFFLINE TESTING

Gesture	Accuracy
Action	1.0
Advance	1.0
Attention	1.0
Charge	1.0
Cover	1.0
Crouch	1.0
Rally	1.0
Shift Fire	1.0
Point of Entry	0.99
Confused	0.98
Hurry	0.98
Sneak	0.98
Confused	0.98
Out of Action	0.96
Come	0.91

IV. EXPERIMENTS

A. *Stationary Gestures Baseline*

Our baseline approach that treats all gestures as static classifies with overall accuracy of 96.75%, including 99.9% accuracy for static gestures and 93.6% accuracy for dynamic.

B. *First Approach: KNN*

Our initial approach of using KNN on *splitDP*s classifies with overall accuracy of 90.1%, including 82.67% accuracy for static gestures and 97.5% accuracy for dynamic.

C. *Offline Testing*

First we tested our offline system on the database of 900 gesture example files, constituting 30 different gestures. We split this into 600 training and 300 test examples, 20 training and 10 test examples per gesture. The accuracy results are presented in two tables. Table I and Table II show the accuracy for the 15 static gestures and 15 dynamic gestures respectively.

We achieve an overall accuracy of 96.9%, with 95.7% accuracy for static gestures and 98.5% for dynamic gestures.

D. *Live Testing*

For live testing we had 3 people execute all 30 commands in succession in front of the robot. We defined a successful classification if the correct gesture command was sent to the robot in a reasonable amount of time and if no incorrect gesture command was sent. While we are only testing 3 examples per gesture, we have already shown that our classifier works very well offline and this is to show that it can translate to running live. Each person was able to execute all 30 gestures successfully. The robot also successfully responded for the 6 gestures for which we have coded movements. Unfortunately the the Kinect must recalibrate the person executing gestures in front of the robot after every movement.

V. ANALYSIS

The Stationary Gesture baseline performs almost perfectly on static gestures, as it was designed to do. It also does decently well for dynamic gestures. Our actual approach achieves roughly the same overall accuracy as the baseline, though it does 1.4% worse for static gestures and 4.9%. Therefore a logical next step for this project is to implement a cascading classifier that first decides if the gesture is static or dynamic and then uses the stationary or actual classifier accordingly. The first step could be another SVM, or it may even be sufficient enough to just look at the length of the *splitDP*.

Our initial approach with KNN does significantly worse on static gestures and comparable on dynamic. The biggest downsides is that it takes much longer to train and test because KNN is so slow.

The system works very well running live on the robot, which is the goal of the project. The major limitations come from using a Kinect to track the skeleton as the person must stand in front of the robot with their arms in the “Psi” pose to calibrate. The Kinect then must be able to see the person’s skeleton to collect data so challenges such as occlusion are impossible to tackle.

VI. Future Work

As mentioned in Section V, creating a cascaded classifier to first decide if the gesture is stationary or dynamic and then using the Stationary Baseline or Actual System accordingly would most likely improve the performance of the system.

Improving the features extracted from the gesture and then plugged into SVM-Multiclass would be the biggest aspect that would improve the accuracy of this project. Making the features more relative to the body would help distinguish between gestures of the same shape in different locations. Also using features other than frequency and amplitude would allow us to try gestures that aren’t as sinusoidal, though strictly sinusoidal gestures appear to be desired in military settings.

An interesting extension to this project would be implementing two-arm gestures. With our current system classifying the two arms separately we envision that two-arm gestures would entail determining if certain one-arm gestures occur in the correct combination for both arms. For example, if both arms are in the rifle position, instead of classifying as rifle it would be the “surrender” gesture.

Another interesting extension would be to incorporate motion of fingers into the gestures. Unfortunately the Kinect skeleton tracker does not currently find fingers, so another vision tracker would have to be used.

Extracting extra information from the gesture would be very useful for military personnel. For example, when a robot is told to move abreast it could discern which arm is giving the motion and then move the correct side accordingly.

References

[1] http://www.businessinsider.com/kinect-gesture-recognition-technology-helping-surgeons-2011-3

[2] http://code42tiger.blogspot.com/2011/02/controlling-tv-and-set-top-box-with.html

[3] http://www.lefande.com/hands.html

[4] http://www.fcrc.org/hand/

[5] http://mail.google.com/mail/help/motion.html

[6] http://svmlight.joachims.org/svm_multiclass.html

[7] http://pr.cs.cornell.edu/