Learning and Adaptivity
Object Recognition

Stepan Pazekha, Santosh Thoduka
Bonn-Rhein-Sieg University of Applied Sciences

June 17, 2014

1 Introduction

The purpose of this project is to enable autonomous recognition of industrial objects. The objects were chosen were the official objects from the Robocup@Work competition and an additional object, *M20_180*, which is a longer version of *M20_100*. See Figure 1 for the @Work objects.

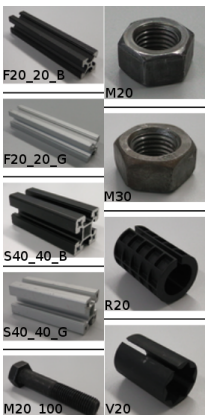

Figure 1: *RoboCup@Work objects [1]*

The existing object recognition package ("ORB-Box") used by the b-it-bots@Work team has a recognition rate of 85.9% (as determined from the test error rate of the neural network used for both laying and standing objects). This project attempts to improve the recognition rate by exploring usage of additional features and new learning methods.

2 Approach

2.1 Data collection

Pointcloud data for each of the objects was collected using an existing state machine which segmented, clustered and accumulated pointclouds of objects placed on a flat surface. The Asus Xtion Pro was used as the RGB-D camera for generating the pointclouds. Fifty samples were collected for each object in different orientations and lighting conditions. Figure 2 shows a sample pointcloud from *M20_100*. An additional 12 samples each were collected as test data.

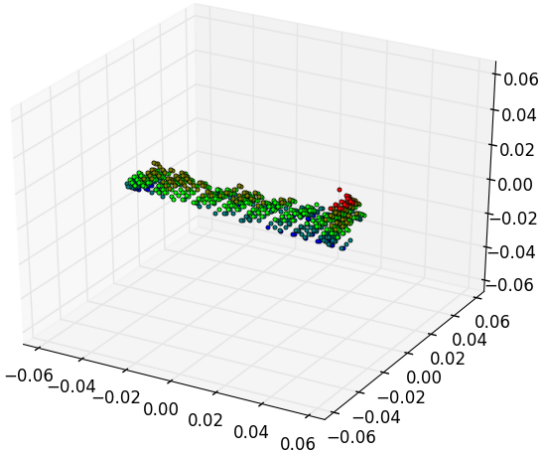

Figure 2: *Pointcloud for M20_100*

2.2 Invariance to Pose

All pointclouds were transformed so that their three principal axes (retrieved by PCA - see code) are aligned along the x,y and z axes respectively. This renders subsequent features calculated on this pointcloud invariant to the original pose of the object.

2.3 Features

The ORBBox approach uses 6 features - x,y,z lengths of the bounding box of the pointcloud, mean colour, median colour and number of points in the pointcloud. This was found to be insufficient to,

for example, distinguish between *F20_20_B* and *M20_100* reliably.

This project uses x,y,z lengths of the bounding box, mean colour and median colour as done previously. It was decided not to use the number of points since that feature is dependent on the distance the object from the RGB-D camera. In addition to the five features mentioned, the following features were also used:

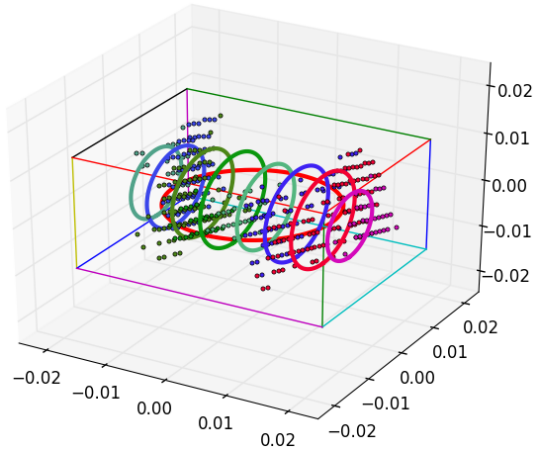

Figure 3: *M30 nut with mean circles and bounding box*

2.3.1 CoG Offset

The offset of the centre of gravity of the object from the midpoint of the minimum and maximum x points is used as a feature. It was used in an attempt to better distinguish between *F20_20_B* and *M20_100* since *F20_20_B* is symmetric about the Y-Z plane whereas *M20_100* is not.

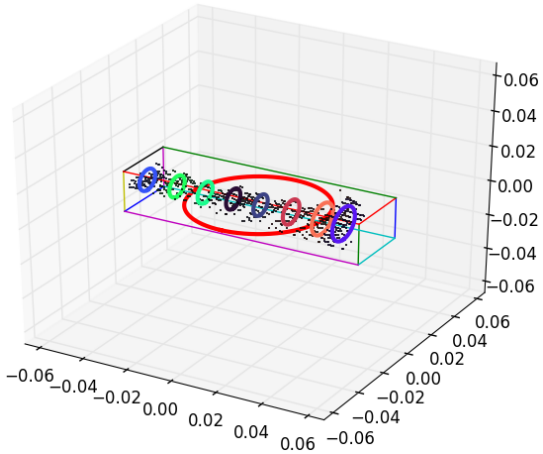

Figure 4: *M20_100 bolt with mean circles and bounding box*

2.3.2 Mean Circles

The radius of the mean circle on the X-Y plane, ratio of the outlier to inlier error w.r.t the mean circle and the radial density (which measures how densely clustered the points are around the mean circle) were used as features. These features describe the size, "roundness" and distribution of density of the object around the mean circle. See code.

In addition to the circle on the X-Y plane, the same features were calculated along the Y-Z plane. Since the object is the longest along the x-axis, it was split into 8 slices along the X-axis and 8 sets of these features were obtained.

In total, this resulted in a feature vector of length 33. The features were standardized to have zero mean and unit variance [2].

The bounding box and mean circles are visualized below for some of the objects.

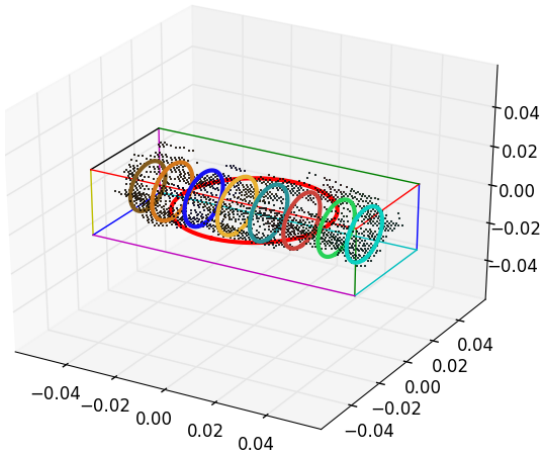

Figure 5: *Large aluminium profile (S40_40) with mean circles and bounding box*

It is observed from Figure 4 and Figure 5 that the aluminium profile has very similar sized mean circles along its X-axis, whereas the bolt has a noticeably larger mean circle at one end (corresponding to the cap of the bolt) compared to the remaining circles.

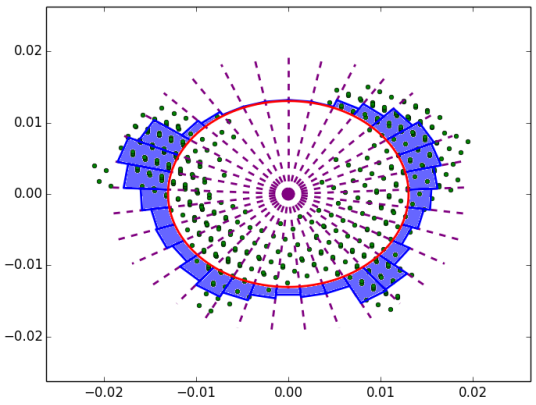

Figure 6: *Radial density distribution on M30 nut*

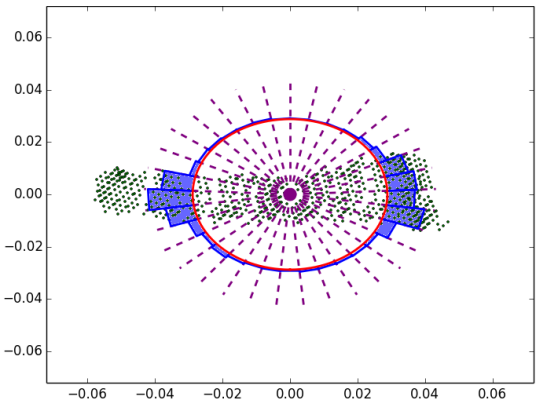

Figure 7: *Radial density distribution on M20_100 bolt*

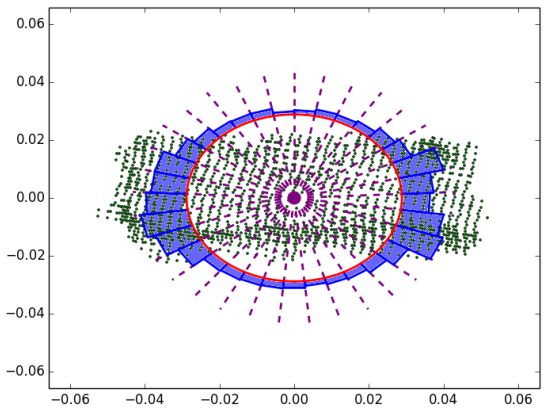

Figure 8: *Radial density distribution on S40_40 profile*

The radial density feature divides the mean circle into 33 sections along the circumference and builds a histogram based on how many points project onto each of the sections. A circular object would have a more uniform distribution around the circle (as seen in Figure 6), whereas a rectangular object would have most points projected onto two sections of the circle (as observed in Figure 7 and Figure 8)

3 Learning Method

A support vector classifier with a radial basis function kernel [3] was used (see code). The same features were also fed into a multi-layer perceptron neural network, but the performance was found to be slightly worse than with the SVM. The training data consisted of 500 samples and 120 samples were used for testing. The samples in both sets had objects in various poses - vertical and horizontal.

4 Results

Using all 10 objects, the classifier achieved a prediction rate of 85% on the test set. Most misclassifications involved *V20*, *R20* and *M20*. *V20* was removed from the object list and the prediction rate went up to 95.3%.
The ORBBox approach has a prediction rate of 97.3% with objects laying down, and 85.9% with objects in any pose (all results are excluding *V20*).
With the SVM approach, there still are a few misclassifications of *M20_100* with *F20_20_B* and *M20* with *M30*. However, overall the performance has increased compared to the existing approach.

5 Conclusion

The pointclouds we collected were sparse and did not define the objects very well. They seemed noisy and only vaguely represented the shapes of the objects. The majority of the time was spent thinking of and extracting features from the sparse pointclouds. Originally, colour was not used as a feature. However, it was decided that incorporating the colour into the learning process would be easier than post-processing (after object recognition) to distinguish between the aluminium profiles of different colours.
An important lesson learnt from this project was the need to standardize features. Before standardization both the SVM and neural network performed quite poorly (prediction rate <60%).
The method described needs to be tested on only laying down objects to see if it can improve on the 97.3% recognition rate achieved by the ORBBox approach.

References

[1] Nico Hochgeschwender Rainer Bischoff Daniel Kaczor Frederik Hegger Gerhard Kraetzschmar, Walter Nowak. Robocup@Work Ruleook. `http://www.robocupatwork.org/download/rulebook-2013-06-08.pdf`. [Online; accessed 13-June-2014].

[2] Andrew Ng. Gradient descent in practice I: Feature Scaling. `https://class.coursera.org/ml-003/lecture/21`. [Online; accessed 14-June-2014].

[3] scikit learn. Support Vector Machines. `http://scikit-learn.org/stable/modules/svm.html`. [Online; accessed 14-June-2014].

Appendix A: Code Samples

Transform cloud based on PCA

```
def normalize_pointcloud(pcd):
    pcdx = pcd[:,0:3]
    n_points,n_dims = pcdx.shape
    c = np.sum(pcd, axis=0) / n_points
    pcdx[:,0] -= c[0]
    pcdx[:,1] -= c[1]
    pcdx[:,2] -= c[2]
    pca_vecs = PCA_compress(pcdx).components_
    sq_pca_vecs = np.multiply(pca_vecs, pca_vecs)
    len_pca_vecs = np.sqrt(np.sum(sq_pca_vecs, axis=1))
    R = pca_vecs
    R[0,:] = R[0,:] / len_pca_vecs[0]
    R[1,:] = R[1,:] / len_pca_vecs[1]
    R[2,:] = R[2,:] / len_pca_vecs[2]
    if pcd.shape[1] > 3:
        pcd_ret = np.hstack([R.dot(pcdx.T).T, pcd[:,3][np.newaxis].T])
    else:
        pcd_ret = R.dot(pcdx.T).T
    return pcd_ret
```

Mean Circle and Radial Density

```
def fit_circle_2(point_cloud, dim1, dim2):
    point_cloud = np.vstack([point_cloud[:,dim1],\
        point_cloud[:,dim2]]).T
    n_bins = 33
    histogram = np.zeros([n_bins])
    w = np.arctan2(point_cloud[:,0], point_cloud[:,1])
    shift_i = np.where(w < 0)
    w[shift_i] += 2*np.pi
    w = np.round((n_bins-1)*w / (2*np.pi))
    for k in w:
        histogram[k] += 1
    if np.max(histogram) > 1e-1:
        radial_density = np.sum(histogram/np.max(histogram))/n_bins
    else:
        return 0, 0, 0, 0
    dist = np.sqrt(np.sum(np.multiply(point_cloud[:,0:2],\
        point_cloud[:,0:2]), axis=1))
    radius = np.mean(dist)
    diff = dist - radius
    inerr = diff[np.where(diff < 0)]
    outerr = diff[np.where(diff >= 0)]
    inerr = np.mean(np.multiply(inerr, inerr))
    outerr = np.mean(np.multiply(outerr, outerr))
    return radius, inerr, outerr, radial_density
```

SVM classifier

```
def train_svm(features, labels):
    label_encoder = sklearn.preprocessing.LabelEncoder()
    label_encoder.fit(labels)
    encoded_labels = label_encoder.transform(labels)[:,np.newaxis]
    encoded_labels = np.squeeze(encoded_labels.T)
    classifier = svm.SVC(kernel='rbf')
    classifier.fit(features, encoded_labels)
    return classifier, label_encoder
```