Robotics Practical

Teaching Robots to Accomplish a Manipulation Task

Spring 2016
# Contents

## 1 Introduction

1.1 Task Description ............................................. 4  
1.2 Steps to follow ............................................... 5  
1.3 Preparation of Practical ..................................... 6  
1.4 Practical Evaluation ......................................... 6

## 2 Object Recognition

2.1 Overview .......................................................... 7  
2.2 PCA-based Object Recognition ................................ 7  
  2.2.1 Background .................................................. 7  
  2.2.2 Efficient Image Storage .................................... 7  
  2.2.3 Implementation ............................................ 7  
2.3 Calibration of the camera frame to the robot frame .......... 11  
  2.3.1 Background .................................................. 11  
  2.3.2 Calibration .................................................. 11

## 3 Teaching a reaching movement to a robot

3.1 Overview .......................................................... 14  
3.2 Kinesthetic Teaching ........................................... 14  
  3.2.1 Robot reaching motion acquisition ......................... 14  
  3.2.2 Train the acquired trajectories and evaluate the results . 15  
  3.2.3 Robot implementation ...................................... 17

## 4 Building a tower game

4.1 Overview .......................................................... 19  
4.2 Steps to follow .................................................. 19  
  4.2.1 Object Localization .......................................... 19  
  4.2.2 Recognize object ........................................... 19  
  4.2.3 Robot Movement Generation ................................. 20  
  4.2.4 Integration .................................................. 21

## A Appendix

A.1 Algorithm for PCA .............................................. 22
A.2  Gaussian Mixture Regression (GMR) ......................................................... 22
A.3  Stable Estimator of Dynamical Systems (SEDS) ................................. 24
1 Introduction

In this robotic practical, you will be teaching a robot to build a tower by stacking several objects on top of each other. Each group is provided with several objects with different shapes and sizes. In order to accomplish this task, the robot must be capable of 1) identifying the objects; 2) estimating the location and pose of the objects; 3) generating appropriate motion to pick up each object and stack it on top of other objects. In this practical, you will learn how you can teach a robot through human demonstration to acquire the first and third competence. The second competence, namely estimating the position and orientation of the object is beyond the scope of the practical and hence, you will be provided with a ready to use package to perform this step.

During the practical, you are expected to understand and use two machine learning algorithms, namely principal component analysis (PCA) and Gaussian Mixture Regression (GMR), to complete the robotic tasks. These algorithms have been taught in the compulsory course in machine learning which you have taken in the fall semester. For those of you who did not take this course, you can find a description of these algorithms in Appendices A.1 and A.2. The basic libraries for these algorithms are provided, but you may need to slightly modify them based on your requirement. More importantly, you should be aware of the performance and limitation of these algorithms in practice.

1.1 Task Description

As described before, in this robotic practical, your task is to teach a robot to build a tower by stacking several objects on top of one another. The robot you will use is a light-weight, small-size, five degrees of freedom (DOF) Katana arm, developed by Neuronics (see Fig. 1a). There are different objects, such as cylinder, cube, triangle. All objects should be placed on the table and within the workspace of the robot to start with. Note that the objects do not need to be placed in any particular location within this workspace, as the robot should be able to adapt to arbitrary position and orientation of the object. You should teach the robot how to pick each of these objects. Since objects differ in size and shape, the approach motion may differ depending on where you grasp the objects, their sizes and shapes as well as how to place them on top of other objects to ensure that the tower remains stable (see Fig. 1).

This task should be accomplished in two steps: offline training and online implementation. The offline training includes object recognition and reaching movement learning, see Fig. 2. For the object recognition, you need to use a camera mounted on the robot arm to collect enough images for each object, as shown in Fig. 1b. Then for all the object images, use PCA to learn a model to recognize objects. For the learning of robot movements, you will teach the robot how to reach and grasp a block through a couple of demonstrations, see Fig. 1c. Because the Kanata robot can be back-driven, you just

Figure 1: (a) The kinematics of the Katana robot, 5 degree of freedoms in the arm with an additional one in the gripper. \( \theta_i, i = 1...5 \) refers to the corresponding joints. (b) The objects will be recognized using PCA. (c) The human teacher is showing to the robot how to place the robot through *kinesthetic teaching*, namely by passively moving the robot’s joints from the original position to the final position.
need to hold the robot arm and move the robot end-effector from the initial to the desired final position. The process of programming a robot based on some demonstrations is simply called programming by demonstrations (also referred to as imitation learning, apprenticeship learning, or learning from demonstration). See http://www.scholarpedia.org/article/Robot_learning_by_demonstration for more information on programming by demonstrations. Videos of practical examples of imitation learning for teaching complex robotic tasks are also available at http://lasa.epfl.ch.

1.2 Steps to follow

Following Fig. 2, the general pipeline of the task is as follows:

1. You will be assigned several (minimal 3) objects with different shapes and sizes.

2. Initially, you will take some photos of these blocks from different positions, which will be used later on for the object recognition (Section 2).

3. You will also need to measure the size of each object (Length $\times$ Width $\times$ Height). These information will be very important for you to design the stacking sequence.$^1$ Also mention these object sizes in your report.

4. You will teach your robot several demonstrations of how to reach for an object, and how to place it at a desired position (Section 3).

$^1$During the stacking task, the X – Y position of the tower will be fixed or chosen by student, however, the Z position will change as students need to put one object on another one.
5. You will train a general model of the robot motion based on your demonstration trajectories (Section 3).

6. You will put together Steps 1 to 5 to build a stable and high tower (Section 4).

1.3 Preparation of Practical

- Week 1: Read Sections 1 and 2 to get a general idea of how to perform object recognition and localization.
- Week 2: Read Section 3 and Appendices A.2 and A.3 to learn how to model a point-to-point motion.
- Week 3: Read Section 4 to get an idea how to integrate the materials from the first two weeks to finish a tower game.

1.4 Practical Evaluation

The grade for this project will be based on your answers to the questions in this document (object recognition 30%, reaching movement 40%), as well as the performance of your group in performing the task of building the tower (performance 30%), which will be evaluated based on the height, the stability and the time for the tower building. Note that this grading of the report will count for the entry one on the overall grading scheme and hence amount to 50% of the overall grade (see Table 1).

Write a report of maximum 20 pages in PDF format (pages beyond the twentieth will be ignored). Font size should not go below 10pt and use single column format. Do not forget to include all necessary graphs or tables in your report when answering the questions above. You need to justify why these graphs are important by referring to them in your response. Needless to say, all figures should have caption, axis label, legends, etc.

NOTE: When working with the robot, You need to precisely follow all the instructions provided in this document as well as those elucidated by the assistants during the practical.

!!PRECAUTION!!

- 1 Before you run the robot, make sure it will not collide any obstacle or person!
- 2 Move the robot to the initial position when you try to turn the motor off!
- 3 When you show the demonstrations to the robot, keep in mind the singularity of the robot!

Table 1: The overall grading scheme for the practical.

<table>
<thead>
<tr>
<th>Grading Item</th>
<th>Overall weight</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Accomplishments:</strong> Goals reached, quality of results, Q&amp;A (discussion)</td>
<td>50%</td>
</tr>
<tr>
<td><strong>Methodology:</strong> Systematic approach, understanding of the subject, personal contribution</td>
<td>20%</td>
</tr>
<tr>
<td><strong>Working style:</strong> Autonomy, communication with assistants, timeliness</td>
<td>10%</td>
</tr>
<tr>
<td><strong>Report:</strong> Structure (hierarchy, appendix, ...), completeness, clarity of presentation (general form, graphs, captions ...)</td>
<td>20%</td>
</tr>
</tbody>
</table>
2 Object Recognition

2.1 Overview

As mentioned in the introduction, the first step to accomplish a manipulation task is to detect what the object is and to know where it is located. While it is easy for humans to recognize objects when viewed through different angles and lighting conditions, or even when partially obstructed, computers/robots still struggle to robustly detect objects.

To perform the task at hand, it is crucial that your object detector be able to recognize the object when viewed from various angles. To enable the robot to get such a generalized model of the object, you will need to provide it with a variety of images of the objects that are representative of the variety of views you expect it to encounter when performing the task. To recognize objects, you will use the camera located on top of the robot’s end-effector, see Fig. 1b. You will embed a model of each object using Principal Component Analysis (PCA), see Appendix A.1. In the remaining part of this section, the object recognition will be described in more details.

2.2 PCA-based Object Recognition

2.2.1 Background

For object recognition, a natural way is to store the known objects as sample images or templates, then try to find the best match in the new given image. For each object, one needs to save a lot of images corresponding to different views. Then the number of sample images will increase very fast when more objects need to be recognized. A good solution for this situation is to store the sample images in an efficient way. PCA can be very helpful to find the similarities between these templates and save them in a more efficient way.

2.2.2 Efficient Image Storage

In order to highlight the importance of having a smart way of storing images efficiently, let us take an example using real object images: a bowl, a tissue and a stapler (see Fig. 3). For each object, ten $640 \times 480$ images are collected using a camera mounted on the robot, the camera 1 as shown in Fig. 1. These images are first resized to $128 \times 96$ gray scale as shown in Fig. 4. Then the PCA algorithm described in the Appendix A.1 is directly applied on them. The first 6 eigenvectors of these images are shown in Fig. 5, which correspond to 85% of the variance of the trained images. These eigenvectors can be used as the axis of the new coordinate frame. Then each image can be saved in the new frame with only 6 coordinates, which can be computed according to step (6) in Appendix A.1. For example, the $j$th image of object $i$ can be represented as depicted in Fig. 6.

Before doing PCA, in order to save 30 object images of the size $128 \times 96$, we need $30 \times 128 \times 96 \times \text{sizeof}(\text{int})$ memory. After PCA, in the example above, each object will only has 6 coordinates, then the total memory is $(30 \times 6 + 128 \times 96 \times 6)\text{sizeof}(\text{int})$. Reducing the dimensionality of the dataset using PCA saved us then around 80% memory.

2.2.3 Implementation

During the practical, you do not need to code the PCA algorithm (All students must read Appendix A.1 for further information about this algorithm). But you need to collect the block images for the training dataset as shown in Fig. 7.

---

For this practical, the object recognition algorithm should be invariant to the object orientation (rotation on the table). To this end, we collect images for each object with different orientation.
Steps to follow:

1. Take one of the objects and put it on the position marked by a cross on the table. This is the default image capturing position, where you could take the photos of the object using the webcam that is mounted on the robot arm (see Fig. 7)

2. Execute Katana control software by opening a terminal and typing the following lines:

   ```
   cd $PRACTICAL/cppcodes/LasaKatana/
   . /bin/PRACTICAL
   ```

3. A command list will be displayed like below:

   ```
   Command List
   1. Basic Tasks:
   [ calibrate ] : Calibrate Katana
   [ open ] : Open the gripper
   [ close ] : Close the gripper
   [ motor on ] : Turn on the motors
   [ motor off ] : Turn off the motors
   [ home ] : Go to home position
   [ quit ] : Quit the software
   2. For PCA Session:
   [ pca ] : Move the Katana to the default position
   [ grab ] : Grab a image from the camera mounted on the robot
   3. For SEDS Session:
   [ pick ] : Get demonstrations of picking task
   [ place ] : Get demonstrations of placing task
   ```

4. Before using the robot, we need to **calibrate** it. First, make sure the robot is in its home position (Fig. 1a shows the robot in its home position). Then type “calibrate” to start the calibration.

5. Type “pca” to send the robot to the default image capturing position. This bring the robot arm on the top of the object, where it can take a clear picture of it using the camera mounted on its wrist.

6. Type “grab” to take image for each object and type object index for each object:
Figure 5: The first 6 eigenvectors of the trained object images.

\[
g_{ij}^1 \times \text{Eigenimage 1} + g_{ij}^2 \times \text{Eigenimage 2} + g_{ij}^3 \times \text{Eigenimage 3} \\
+ g_{ij}^4 \times \text{Eigenimage 4} + g_{ij}^5 \times \text{Eigenimage 5} + g_{ij}^6 \times \text{Eigenimage 6} = \]

Figure 6: Reconstruction of an image as a linear superposition of the 6 eigenvectors shown in Fig. 5.

>> Please type image index (1 to 10): 
[ObjInd]
>> If you are ready to save images, press [ENTER]
>> If you want to change image index, type index id.
>> If you want to exit, press 'x'.

Note: take one image without any object as the background image! Change the image name to bknd.png.

7. Repeat Step 6 for all your objects and all the images will be saved in “PracticalCodes/data/img/-training”. Move some images from here to the “testing” folder, which will be used as a testing dataset.

8. Move robot to home position

9. Open MATLAB by typing the following line in a new terminal:

   >>matlab &

10. Open the file “PracticalCodes/matlabcodes/PCA/Object_recognition.m” and follow the instructions in this file as follows.
Figure 7: There are five blocks that will be used for the practical. In order to recognize the blocks, you need to collect images of each blocks from different view.

```matlab
% Students need to modify below variables
nbObj = 5; % Number of objects you are trained
nbImg = 10; % Number of images for each object

% Save the images as a matrix and compute the covariance
[MatrixImage, imgMat, imMean, imVec0] = ImageProcessAuto(obj_files);

% Compute the eigenvectors and eigenvalues
[D, ev] = CompEigen(MatrixImage, imgMat);

% Determining the number of principal components by changing the threshold
NK = number of pc
threshold = 0.85;
for i = 1:size(D, 1)
    temp = sum(D(1:i))/sum(D);
    if (temp > threshold)
        NK = i;
        break
    end
end

% Save the model in a struuture and write to a text file
LearnedPCAModel = SaveTrainedPCA(NK, ev, imMean, imVec0, nbObj, nbImg);

% Plot the eigen image if you want to see the results
for i = 1:NK
    figure(i)
    eigIMAGE = reshape(LearnedPCAModel.eigenVec(:, i), 96, 128);
    colormap(gray(256));
    imagesc(eigIMAGE);
    daspect([1 1 1]);
    xlabel(sprintf('The %dth eigen images', i), 'FontSize', 18);
end

% test the performance of the trained PCA
PracticalRootPath = getenv('PRACTICAL');```
Excercise 1, Object Recognition (30 % of the total score): Build a PCA model of your objects by following steps 1 to 10 above, and answer the following questions:

1. Explain how you created your training set: how many and what types of images did you pick for each object? Justify your choices. (5 %)

2. Validate your trained model with some new images (i.e. Step 7). How many and what type of test images did you use to have a good validation of the trained model? Justify your choices. What is the performance of your object recognition model? (10 %)

3. Is there any misclassification? If yes, explain the source(s) of misclassification? If no, explain what experiment conditions you have considered to obtain this performance? Could you think of other means to validate your model? (10 %)

4. How many bytes are used to store these images for each object in MATLAB before and after PCA (you need to understand the example mentioned in Section 2.2.2)? What is the optimal number of eigenvectors for your dataset and how did you pick it? (5 %)

2.3 Calibration of the camera frame to the robot frame

2.3.1 Background

In order to find the position of objects with respect to the robot frame, the transformation matrix $T$ between the camera frame and the robot frame must be calculated:

$$
\begin{bmatrix}
R^i_X \\
1
\end{bmatrix} = R_T O \begin{bmatrix}
O^i_X \\
1
\end{bmatrix}
$$

(1)

where $R^i_X$ ∈ $\mathbb{R}^{2 \times 1}$ and $O^i_X$ ∈ $\mathbb{R}^{2 \times 1}$ are the position of $i^{th}$ object in the robot and the camera reference frame, respectively. $R_T O = \begin{bmatrix}
R_{RO} & R_{PO} \\
0 & 1
\end{bmatrix}$ ∈ $\mathbb{R}^{3 \times 3}$ is the transformation matrix. $R_{RO}$ and $R_{PO}$ are the rotation and the position matrices. (1) can be re-written in more convenient form as follow.

$$
R^i_X = G R_{RO} O^i_X + R_{PO}
$$

(2)

where $G$ is the scaling parameter. There are several ways to calculate the parameters of the transformation, in this course we follow the optimization based calibration algorithm.

2.3.2 Calibration

During the practical, you do not need to code the optimization algorithm. But you need to measure the position of objects in the camera and the robot frame for the training dataset. In order to calibrate the position of the camera to the robot frame you need to do the following steps:

1. Regarding to your scenario, select at least three points in the workspace of the robot on the table.
2. Find the position of the points with respect to the robot frame by typing the following lines:

```
>> cd SPRACTICAL/cppcodes/LasaKatana/
>> ./bin/Practical
```

3. Calibrate the robot

```
>> calibrate
```

4. Type "pca" to send the robot to the default image capturing position. This bring the robot arm on the top of the table, where it can take a clear picture of it using the camera mounted on its wrist.

5. hold the robot firmly with your hand and type

```
>> motor off
```

6. move the robot to the desired positions. Then type

```
>> disp
```

. The first six number is the position of the joint and the last three numbers correspond to the end-effector position wrt the robot frame (x,y and z coordinates).

7. Type

```
>> motor on
```

and write down the position of the end effector. Repeat the same process for the next points.

**Always firmly hold the end-effector while you are moving it from one position to another.**

8. To find the coordinates of the point to camera frame open a new terminal and type

```
>> roscd MObjLocal/
>> ./bin/tester
```

9. Clean the table and press "s" to record a picture of a clean table. The image will be taken by the camera which is mounted on the table.

10. Place an object at the first point an type "s" to capture a picture. The program prints the x,y-coordinates of the object with respect to camera frame. Write down the coordinate of the position and repeat the process for the next points.

After recording the position of all the points with respect to the robot and the camera frame, you are ready to do the calibration.

11. Open MATLAB by typing the following line in a new terminal:

```
>> matlab &
```

12. Open the file “PracticalCodes/matlabcodes/calibration/Calibration_KATANA.m” and fill ?s in the code.

```matlab
XR1=?; YR1=?; \ %Position of Point 1 in the Robot frame
XO1=?; YO1=?; \ %Position of Point 1 in the camera frame
P1=[XR1 XO1;YR1 YO1];
XR2=?; YR2=?; \ %Position of Point 2 in the Robot frame
XO2=?; YO2=?; \ %Position of Point 2 in the camera frame
P2=[XR2 XO2;YR2 YO2];
XR3=?; YR3=?; \ %Position of Point 3 in the Robot frame
XO3=?; YO3=?; \ %Position of Point 3 in the camera frame
P3=[XR3 XO3;YR3 YO3];
XR4=?; YR4=?; \ %Position of Point 4 in the Robot frame
XO4=?; YO4=?; \ %Position of Point 4 in the camera frame
P4=[XR4 XO4;YR4 YO4];
obj_handle = @(p) obj_1(p,P1,P2,P3,P4);
```
p0=[0.0;0;0];
options.max_iter=5000;
opt=optimoptions('fsolve','Algorithm','trust-region-dogleg','Display','iter','
MaxFunEval',20000,'MaxIter',options.max_iter,'FinDiffType','central');
[popt,fval,exitflag]=fsolve(obj_handle,p0,opt);
if exitflag>0
Px=popt(2)
Py=popt(3)
G=popt(1)
opt=optimoptions('fsolve','Algorithm','trust-region-dogleg','Display','iter','
MaxFunEval',20000,'MaxIter',options.max_iter,'FinDiffType','central');
obj_handle = @(p)obj2(p,P1,P2,P3,P4);
p0=[G;1.57;Px;Py];
[popt,fval,exitflag]=fsolve(obj_handle,p0,opt);
if exitflag>0
clc
display('The problem is solved')
fval
Px=popt(3)
Py=popt(4)
G=popt(1)
cost=cos(popt(2))
sint=sin(popt(2))
else
display('The problem has no solution')
else
display('The problem has no solution')
end
end

13. Update the coordinates of the positions as you are guided by the comments of the file. Run the program.

14. If your measurements are current, the program will print the parameters: Px, Py, G, cost, and sint.

15. Update the following file “/devel/roscodes/PracticalCodes/data/localCalib/calibration_result_S_T.txt” by writing the calculated values with the same order.

**Exercise 2, Calibration (20 % of the total score):**

(a) Explain why the robot and the camera are needed to be calibrated. (5 %)

(b) Explain how you created the training set: how many points did you pick? Justify your choices. (5 %)

(c) What is the minimum number of the points you need to calibrate cameras? Is it enough? Justify your answer. (5 %)

(d) Validate your trained model with some new images. How many points did you use to have a good validation of the trained model? Justify your choices. What is the performance of your calibration? How can you improve it? (5 %)
3 Teaching a reaching movement to a robot

3.1 Overview

In this session, you will teach a robot how to move its end-effector to a desired position to pick an object and to move it to a specific position. All these movements are called point-to-point motions (also referred to as episodic motions, discrete motions, or reaching motions) because they always start from one point and stop at a terminal point, called target. We will use an imitation learning approach to teach a robot these point-to-point motions. Imitation learning enables a robot to learn a given task from a set of demonstrations shown by the user. Among the several imitation learning-based point-to-point-motion learning methods, we will use a Dynamical System (DS) approach, namely Stable Estimator of Dynamical Systems (SEDS) [1].

SEDS builds an accurate estimate of robot motions from a set of demonstrations while ensuring its global asymptotic stability at the target. SEDS uses finite mixture of Gaussian functions to encode the movements. The number of Gaussian functions should be defined by the user (usually lower than 10)\(^3\). Figure 8 shows an example of 2D training data and its reproduction by the DS model learned using SEDS. A brief overview of SEDS and Gaussian Mixture Model (which SEDS is built upon) is provided in Appendices A.3 and A.2, respectively. Next sections will guide you through the steps necessary to teach a robot point-to-point motion.

![Two-dimensional demonstrations and reproduction by the dynamics model learned through SEDS](image)

Figure 8: An example of two-dimensional dynamics learned from three demonstrations

3.2 Kinesthetic Teaching

3.2.1 Robot reaching motion acquisition

You will demonstrate several point-to-point motions using the provided Katana robot, and measure the robot’s gripper position and joint angle trajectories.

Step to follow:

1. Turn on the robot
2. Execute the Katana control software by typing the following lines in a terminal window:

```
>> roscd LasaKatana
>> /bin/Practical
```

An optimal number of gaussian functions can be found using various method, such as the Bayesian Information Criterion (BIC) [2], the Akaike information criterion (AIC) [3] or the deviance information criterion (DIC) [4]. However in this practical, the number will be given by the user
3. A command list will be displayed like below.

<table>
<thead>
<tr>
<th>Command List</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1. Basic Tasks:</strong></td>
</tr>
<tr>
<td>calibrate</td>
</tr>
<tr>
<td>open</td>
</tr>
<tr>
<td>close</td>
</tr>
<tr>
<td>motor on</td>
</tr>
<tr>
<td>motor off</td>
</tr>
<tr>
<td>home</td>
</tr>
<tr>
<td>quit</td>
</tr>
</tbody>
</table>

| **2. For PCA Session:** |
| pca | Move the Katana to the default position |
| grab | Grab a image from the camera mounted on the robot |

| **3. For SEDS Session:** |
| pick | Get demonstrations of picking task |
| place | Get demonstrations of placing task |

4. Calibrate the Katana robot by typing “calibrate”

5. Start record picking motion by typing “pick”

6. The robot arm will automatically move to a proper initial posture.

7. **Hold the robot firmly with your hand.** If you are ready to grab a data, press [Enter]. After pressing enter, the motors will switch off and the robot will be back-drivable. **Failing to grab the robot properly will cause the robot fall on the table and be damaged!**

8. Demonstrate a motion by back-driving the robot (see Fig. 9).

9. When the demonstration is finished, press [Enter].

10. You needed to do the same demonstrations a few times from different initial positions. The optimal number of demonstrations depends on several factors. Try to figure out these factors and come up with an optimal number of demonstrations (See Exercise 2.1).

11. The demonstrations will be stored on the “$PRACTICAL/data/trj/” folder.

12. You can record placing motion in the same way of grabbing picking motion by typing “place”.

13. When you finished the demonstrations, type “quit” to send the robot to its home position and to quit the software. **Note that all the robot motors are turned off when the software is closed, and thus the robot may fall down on the table if it is not properly placed at its home position!**

By following the steps above, you acquire several demonstrations for both picking and placing motions. Next, we will use Matlab to learn a generic and stable model of these motions from the demonstrations using the SEDS library.

### 3.2.2 Train the acquired trajectories and evaluate the results

You will train the acquired trajectory from section 3.2.1 using SEDS Matlab package. The package is located at “$PRACTICAL/matlabcodes/SEDS”.

**Step to follow :**

1. Start Matlab software by typing “matlab” in the Ubuntu terminal.

   >> matlab &
Figure 9: An example of picking and placing demonstrations through back-driving the robot.

2. Change current folder to the SEDS Package folder “$PRACTICAL/matlabcodes/SEDS”.

3. Open the file “practical.m”.

4. You need to modify the parameters defined in lines 9 to 22 based on your need. The functionality of each parameter is described in front of it.

```matlab
%% User Parameters

demoType = 'pick';  \% Specifying the type of demonstrations. Its value should either be 'pick' or 'place'
K = 4;              \% Number of Gaussian functions
options.objective = 'mse'; \% The criterion that optimization uses to train the model. Possible values are:
                          \% 'mse': use mean square error as criterion to optimize parameters of GMM
                          \% 'likelihood': use likelihood as criterion to optimize parameters of GMM
options.max_iter = 500; \% Maximum number of iterations. Optimization exits if it does not converge within the specified iterations
```

5. Execute the code by pressing “F5”.

6. The code automatically train the demonstration and save the trained model in the text format on “$PRACTICAL/data/model”.

7. Change the demoType to place and train the model.

**Exercise 2. Learning the picking and placing motions using the SEDS Matlab package (20% of the total score)**

1. Analyzing the effect of number of Gaussian functions (10%)

   (a) Set `options.objective = 'mse'` and the number of Gaussian functions $K = 1$. Run the code by pressing “F5”, and write down the final estimation error. Repeat this step 3 times. You should observe that the estimation error may vary at each iteration. This is due to the fact that SEDS is initialized randomly. Write down the best estimation error that you obtain from the three trials.
(b) Repeat (a) by incrementally increasing $K$ from 1 to 6, and discuss the effect of increasing the number of Gaussian functions on the final estimation error.

(c) Is there an optimal number of Gaussian functions? If yes, what is this value? If no, could you come up with some criterion to choose the optimal number?

(d) Repeat the steps above for the placing motion.

2. Analyzing the effect of the optimization objective criterion (10%)

   (a) Set options.objective = ‘likelihood’, and repeat the steps above.

   (b) Discuss the performance difference between using mse or likelihood as the optimization criterion. Compare the performance. The performance of which one is better? Why? (The “why” part is an added bonus.)

3. Export the best model of both picking and placing motions by typing the following line:

```
>>> SaveGMM(structGMM, [PracticalRootPath '/data/model/'], demoType);
```

3.2.3 Robot implementation

Using the two trained models of the previous exercise (picking and placing motion models), you can now move the robots to reach for an object and to place it at a desired position. In this section, you use a provided program to test your trained models.

Steps to follow:

1. Execute the DSTest control software

```
>>> roscd LasaKatana/
>>> ./bin/DSTest
```

2. A command list will be displayed like below.

```
Command List
[ calibrate ] : calibrate Katana
[ pick ] : execute pick motion
[ place ] : execute place motion
[ save ] : save the last Cartesian trajectory
[ home ] : go to the home position
[ set ] : set the target position
[ quit ] : quit
```

3. First, calibrate the robot by typing “calibrate”.

4. Get the target position by typing “set”.

5. Send the robot to a proper initial position by typing “initial”. This step is essential to avoid some possible issues when solving the inverse kinematic problem (interested students can ask the assistants for further details).

6. Type “pick” to move the robot from the initial to the target point using the trained picking DS model. Similarly, you could type “place” to use the placing DS model.

7. After the motion was executed, type “save” to store the generated trajectory in a text file. You could use the saved trajectory later on to evaluate the performance of your trained models.

**Exercise 3. Evaluation of the generated motions on the real robot: (10%)**
1. Execute the pick and place motions for three different positions of the target, and save the generated trajectories (see the steps described above). Compare these trajectories to those you demonstrated for the training, and discuss similarities and differences.

2. Read the initial and target positions of each trajectory, and re-generate the motion in the simulation using the provided matlab code. Discuss the similarities and differences between the generated motions in the simulation and on the real robot?
4 Building a tower game

4.1 Overview

In this last session, you will implement a code to let the robot build a stable and high tower. In the previous two sessions, you have become familiar with PCA to perform object recognition and with SEDS to generate robot point-to-point motions. Based on these methods, you can now make an automatic robotic system to build a stable and high tower using the given toy blocks. For this task, each group receives a Katana robot equipped with a camera and five toy blocks to build the tower.

4.2 Steps to follow

A template code providing you with basic features such as position estimation, object recognition, and movement generation is available in the file “myTower.cpp”. You should extend this template file to implement your own autonomous building tower robotic system. Next, we will describe different parts of this file, which can be accessed as follows:

1. Open the project file in “$PRACTICAL/cppcodes/LataKatana/” using eclipse by typing the following line in the Ubuntu’s terminal:

   ```shell
   $> eclipse &
   ```

2. Open “myTower.cpp” in eclipse. This is an example code describing how to use the GMR library, how to communicate with the Katana, how to perform object recognition, etc.

4.2.1 Object Localization

Provided objects will be distributed randomly in the robots workspace as shown in Fig. 10a. The number of object and their 2D positions on the table will be measured using an object localization module. Localization is done using a fixed camera located above the set-up, see Fig. 10. It takes an image using the camera and calculates the objects’ 2D Cartesian position with respect to the Katana’s base coordinate.

The hardware setup and basic procedure for the localization module is shown in Fig 10. Furthermore, the following lines, taken from “myTower.cpp”, shows an example of how to localize the objects.

```cpp
MObjLocal *mLocalization;
// create a tracking class
mLocalization = new MObjLocal();
// get object position list
MathLib::Matrix ObjPositions;
ObjPositions = mLocalization->MObjLocalization(MAX_OBJ, LOCALIZATION_THRESHOLD);
// MAX_OBJ : Maximum number of object to be localized, default 10
// LOCALIZATION_THRESHOLD : default 20
```

4.2.2 Recognize object

Once object positions are measured, the robot should move the camera (which is attached near the end-effector) and put it above the object position. When the camera reaches the desired position, you can identify the object using your trained PCA model. The following lines shows how it is done in “myTower.cpp” file.
Camera 2

Camera 1

(a) The vision system for object localization and recognition

Object image

Background

Object image

Background

subset background

gray scale

binary scale

(b) The Basic procedure for object localization.

Figure 10: The 2D object position will be measured by camera 2.

```
// Object positions = Mog->MObjLocalization(MAX_OBJ, LOCALIZATION_THRESHOLD);
for (int cnt = 0; cnt < MAX_OBJ; cnt++) {
  lObjPositions.GetColumn(cnt, lPos);
  // set target position for robot
  lTargetPos(0) = lPos(0);
  lTargetPos(1) = lPos(1);
  lTargetPos(2) = 0.3;
  bool rst = mKatana->MoveStraight(lTargetPos, 0, lTargetOri);
  sleep(1);
  // do pca
  lObjID[nbObj] = Mog->ObjIden(2, 1, 10); // 18, 6, 12
  xPos[nbObj] = lPos(0);
  yPos[nbObj] = lPos(1);
  printf("Object index: %d", lObjID[nbObj]);
  nbObj++;
}
```

### 4.2.3 Robot Movement Generation

After detecting the objects and estimating their position, we could now move the robot to reach for objects and to place them in a desired position. The following lines describe how we could move the robot to reach for an object, and grasp it. You could easily extend this code to perform the placing motion.

```
// gmr pick
mKatana->GetJointangle(lJoints);
mKatana->SetGMRDyn(mGMRPick);
mKatana->GenerateMotion(lJoints, lTargetPos); // lTargetPos : object position
```
4.2.4 Integration

After searching and recognizing the objects, you will make a different strategy for determining stacking order. For instance, you may make the robot to put an object on the bottom which has the largest flat surface. Then, you should make a complete autonomous software based on the previous implemented codes.

Exercise 4. Integrate the works of previous sessions (PCA and SEDS) and develop a software to make the Katana build a tower (20% of the total score).

1. Write a code to move the camera close to each object so that the object become visible in the camera mounted on the Katana's wrist and hence be recognizable by the PCA algorithm.

2. Design a stocking strategy to maximize the tower height, and describe your strategy in word and using a block diagram illustrating the information flow. Furthermore, describe the results with illustrative pictures, including graphs/figures of succeeded and failed attempts with an explanation of the cause of failures.
A Appendix

A.1 Algorithm for PCA

Given an $N \times N$ image template, one can save it as a column vector $x \in \mathbb{R}^{N^2}$ whose elements are the pixel values of the image. Given a dataset with $n$ such vectors or images, $x^i \in \mathbb{R}^{N^2}, i = 1, ..., M$, we denote their mean as $ar{x} = \frac{1}{M} \sum_{i=1}^{M} x^i$ and proceeds as follows:

1. Subtracting the mean for each vector $x^i = x^i - \bar{x}, i = 1, ..., M$;
2. Computing the covariance matrix $C = [x^1 ... x^M] \begin{bmatrix} x^1^T \\ \vdots \\ x^M^T \end{bmatrix}$;
3. Computing the eigenvectors and values of the covariance matrix: $e^i, \lambda^i, i = 1, ..., M$;
4. Sorting the eigenvector and values: $\lambda^1 > \lambda^2 > ... > \lambda^M$;
5. Assuming only first $k$ components are important, which can be done by computing the cumulative energy content for each eigenvector.
6. For each object $i$ and each image $j, j = 1, ..., J$, compute the new coordinates:
   $g^{ij} = (x^{ij} - \bar{x})^T [e^1, ..., e^k]$
7. For each object $i$, compute the cluster center $\bar{g}^i = \frac{1}{J} \sum_{j=1}^{J} (g^{ij})$

Now, given a new image (In this practical, images are taken using a camera mounted on the robot arm), precess the image to the same format as the training dataset, write it as $x^* \in \mathbb{R}^{N^2}$. Note that here one can easily define different metrics for the object classification, we simply use the distance between the cluster center and the new object as a recognition criteria. When the distance is larger then some threshold, the algorithm will return a null value of object index to indicate a more accurate PCA model is required.

1. computing the coordinates: $g^* = (x^* - \bar{x})^T [e_1, ..., e_k]$;
2. Finding the closest object cluster center $\bar{g}^i$ to $g^*$ by $\min \|g^* - \bar{g}^i\|_2$;

A.2 Gaussian Mixture Regression (GMR)$^4$

Nonlinear regression techniques deal with the problem of building a continuous mapping function $f: \mathbb{R}^n \rightarrow \mathbb{R}^m$ based on a set of $T$ training data points $D = \{\xi^i_x, \xi^i_o\}_{i=1}^T$, where $\xi^i_x \in \mathbb{R}^n$ and $\xi^i_o \in \mathbb{R}^m$ correspond to vectors of input and output variables, respectively. The regression function $f$ is usually described in terms of a set of parameters $\theta$, where an optimal value of $\theta$ can be determined during the training. Once an estimate of $f$ is obtained, then it can be used to predict the value of $\xi_o$ for a new input $\xi_x$:

$$\xi^*_o = f(\xi^*_x; \theta)$$  (3)

$^4$The presented materials in this section are taken from [5].
Gaussian Mixture Regression (GMR) is one of the well-known nonlinear regression techniques that works on the joint probability $P(\{\xi_I, \xi_O\})$ between input and output variables\(^5\). The joint probability is formed by superposition of $K$ linear Gaussian functions:

$$P(\{\xi_I; \xi_O\}) = \sum_{k=1}^{K} \pi^k N(\{\xi_I; \xi_O\} | \mu^k, \Sigma^k)$$  \hspace{1cm} (4)$$

where $\pi^k$, $\mu^k$ and $\Sigma^k$ respectively are the prior, mean and covariance matrix of the $k$-th Gaussian function $N(\{\xi_I; \xi_O\} | \mu^k, \Sigma^k)$ that is described by:

$$N(\{\xi_I; \xi_O\} | \mu^k, \Sigma^k) = \frac{1}{\sqrt{(2\pi)^{n+m}|\Sigma|}} e^{-\frac{1}{2}(\xi_I - \mu^k)^T (\Sigma^{-1}) (\xi_I - \mu^k)}$$  \hspace{1cm} (5)$$

where $(\cdot)^T$ denotes the transpose. From different perspective, Eq. 4 can be rendered as:

$$P(\{\xi_I; \xi_O\}) = \sum_{k=1}^{K} P(k) P(\{\xi_I; \xi_O\} | k)$$  \hspace{1cm} (6)$$

in which $P(k) = \pi^k$ is the probability of picking the $k$-th component and $P(\{\xi_I; \xi_O\} | k)$ stands for the probability the datapoint $\{\xi_I; \xi_O\}$ belongs to this component. In mixture modeling, the unknown parameters of the joint distribution are the priors $\pi^k$, the means $\mu^k$ and the covariance matrices $\Sigma^k$ of the $k = 1..K$ Gaussian functions (i.e. $\theta^k = \{\pi^k, \mu^k, \Sigma^k\}$) and $\theta = \{\theta^1, \theta^K\}$, which can be estimated by using an Expectation-Maximization (EM) algorithm [6]. EM proceeds by maximizing the likelihood that the complete model represents the training data well. Given the joint distribution $P(\{\xi_I; \xi_O\})$ and a query point $\xi^*_I$, the GMR process consists of taking the posterior mean estimate of the conditional distribution:

$$\xi^*_O = f(\xi^*_I; \theta) = \mathbb{E}[P(\xi_O | \xi_I, \theta)]$$  \hspace{1cm} (7)$$

By defining the components of the mean and the covariance matrix of a Gaussian $k$ as:

$$\mu^k = \begin{pmatrix} \mu^k_I \\ \mu^k_O \end{pmatrix} \quad \& \quad \Sigma^k = \begin{pmatrix} \Sigma^k_{II} & \Sigma^k_{IO} \\ \Sigma^k_{OI} & \Sigma^k_{OO} \end{pmatrix}$$  \hspace{1cm} (8)$$

the expected distribution of $\xi^*_O$ can be estimated as:

$$\xi^*_O = \sum_{k=1}^{K} h(\xi^*_I; \theta^k) (\Sigma^{-1}_{II} (\xi^*_I - \mu^k_I) + \mu^k_O)$$  \hspace{1cm} (9)$$

where

$$h(\xi^*_I; \theta^k) = \frac{P(k) P(\xi_I | k)}{\sum_{i=1}^{K} P(i) P(\xi_I | i)} = \frac{\pi^k N(\xi_I | \mu^k_I, \Sigma^k_{II})}{\sum_{i=1}^{K} \pi^i N(\xi_I | \mu^i_I, \Sigma^i_{II})}$$  \hspace{1cm} (10)$$

Figure 11 illustrates an example of using GMR to build an estimate of $f$ from a set of noisy samples using 3 Gaussian functions. For illustrative purpose, a uni-dimensional input and output variables are considered in this example.

\(^5\text{Note that we use the expression } [\xi_I; \xi_O \text{] to vertically concatenate the two column vectors } \xi_I \text{ and } \xi_O. \text{ The resulting vector } [\xi_I; \xi_O \text{] has the dimension } n + m.\)

23
A.3 Stable Estimator of Dynamical Systems (SEDS)\(^6\)

Classical approaches to modeling robot motions rely on decomposing the task execution into two separate processes: planning and execution [7]. The former is used as a means to generate a feasible path that can satisfy the task’s requirements, and the latter is designed so that it follows the generated feasible path as closely as possible. Hence these approaches consider any deviation from the desired path (due to perturbations or changes in environment) as the tracking error, and various control theories have been developed to efficiently suppress this error in terms of some objective functions. Despite the great success of these approaches in providing powerful robotic systems, particularly in factories, they are ill-suited for robotic systems that are aimed to work in the close vicinity of humans, and thus alternative techniques must be sought.

In robotics, DS-based approaches to motion generation have been proven to be interesting alternatives to classical methods as they offer a natural means to integrate planning and execution into one single unit [8, 9, 1]. For instance when modeling robot reaching motions with DS, all possible solutions to reach the target are embedded into one single model. Such a model represents a global map which specifies instantly the correct direction for reaching the target, considering the current state of the robot, the target, and all the other objects in the robot’s working space. Clearly such models are more similar to human movements in that they can effortlessly adapt its motion to change in environments rather than stubbornly following the previous path. In other words, the main advantage of using DS-based formulation can be summarized as: “Modeling movements with DS allows having robotic systems that have inherent adaptivity to changes in a dynamic environment, and that can swiftly adopt a new path to reach the target”. This advantage is the direct outcome of having a unified planning and execution unit.

When modeling robot motions with nonlinear Dynamical Systems (DS), ensuring stability of the learned DS (from a set of demonstrations of the task) is a key requirement to provide a useful control policy. In Appendix A.2, we have described how one can use GMR to build an estimate of a nonlinear function from a set of sample datapoints. However, as EM (the conventional learning algorithm for GMM) do not optimize under the constraint of making the function \( f \) stable at the target, it is not guaranteed to result in a stable estimate of the motion. Stable Estimator of Dynamical Systems (SEDS) is a learning algorithm developed at LASA as an alternative to EM in order to train a GMM from user demonstrations while ensuring its global asymptotic stability at the target. More details about SEDS framework is available at [1].

The estimated DS from SEDS captures the invariant features in the user demonstrations, and can generate motions that resemble the user demonstrations. For instance, Fig. 12 shows an example of learning 20 human handwriting motions using SEDS. For each motion, the user provides 3 demonstrations by drawing a pen on a Tablet-PC. These demonstrations are shown with red dots. By using SEDS, a DS

\(^6\)The presented materials in this section are taken from [5].
Figure 12: Learning a library of 20 human handwriting motion by using SEDS framework. This library is not fixed and the user can freely add a new style of writing, or modify one of the existing motions just by introducing a new set of sample trajectories.

(a) Generation of tennis swing motion as a composition of a swing and a resting primitives. Only three examples of generated trajectories are shown here (plotted as red, green, and blue lines).

(b) In this example \( f^1(\xi) \) and \( f^2(\xi) \) are two basic movement primitives that represent an angle and sine-shaped motions, respectively. The new movement primitive \( f^3(\xi) \) that includes a mixture of both behaviors is obtained through a linear superposition of \( f^1(\xi) \) and \( f^2(\xi) \).

Figure 13: Illustration of two examples exploiting modularity of DS models to generate (a) a more complex motion and (b) a new movement primitive. In this figure, the black star and circles indicate the target and initial points, respectively.

model is estimated for each of these motions. By using these models, we are now able to generate motions that follow the same behavior (i.e. follows the same shape, curvature, nonlinearity) from different point in space (see the solid curves in Fig. 12). In other words, we are now able to give the robot the ability to follow a specific behavior (e.g. writing a letter ‘z’) just with a few examples. Additionally, the user can also teach the robot a new style of writing the letter ‘z’, by just providing a new set of sample trajectories.

Another feature of SEDS is its modularity. In fact, each DS model codes a specific motion (behavior), which is called a movement primitive (also known as motor primitive). These motion primitives can be seen as a building block that can be used to generate more complex or new motions through sequencing or superimposition of the primitives. This modularity of DS-based movement primitives is essential as it allows controlling a wide repertoire of movements from a (small) set of basic motions [1]. Figure 13 shows two examples of exploiting this modularity of movement primitives to generate new motions.

To summarize, a conceptual workflow describing how SEDS works is depicted in Fig. 14. As it is illustrated, first the user provides a set of demonstrations describing how robot should perform a task. Then a DS-based control policy is build using the ‘Learning Core’ of SEDS. This DS model can be used to generate robot commands to properly execute the desired task based on the situation in the robot’s workspace. The ‘Execution Core’ of SEDS is able to adapt in realtime a new trajectory if any change happen in the workspace (e.g. the target point or other objects are replaced). For example in the image shown in the block ‘robot’ in Fig. 14, the robot is required to put a (transparent) glass in front of the user in a cluttered environment. As the robot approaches, the person intentionally moves the red glass in a way that crosses the robot trajectory. Despite this quick change, the robot is able to robustly handle this perturbation and successfully put the transparent glass in front of the user without hitting other objects (more information on this experiment is provided in [10]).

References

Figure 14: A conceptual workflow describing how the SEDS framework works. For further information please refer to A.3.


