Copernicus: Arm Control and Speech AI
ECE 478/578- Intelligent Robotics

Team 3: Christopher Carlson, Dawit Amare, Shashwat
Dr. Marek Perkowski
TA: Melih Erdogan and Mathias Sunardi
Credit: Josh Sackos, Mathias Sunardi and Students of Dr. Perkowski

CONTENTS

1. Introduction

Copernicus is a humanoid robot based on InMoov design. The robot is designed to be an interactive display in the department having knowledge about Nicolaus Copernicus, medieval science, history, and Portland State University. Our project focused on arm design and interactive speech of the robot. In our project we completed the forearm design of the robot, further we created gestures of the arm that were used to enact robots interaction with the world using a small script of monologue. As we also had the task of working on interactive speech software, we integrated the robotic arm monologue gestures with the interactive speech software and created a dialogue as one would have between a robot and human. Originally, as a four members team, we were assigned to work on head and arm of the robot. However, with three of us working on the project, we focused our efforts of completing the arm design and interactive speech interface of the robot.

Copernicus robot's current condition is decommissioned. It has a torso, a neck, an arm attached to his left shoulder and another arm removed from his right shoulder. Copernicus does not have a face. The servos in copernicus' robot are functional and they can easily be interfaced with. This project used the free arm to show control and developed a basis for Copernicus mind through AIML. Copernicus, now can control 5 of his fingers and speak when asked through a terminal.

Copernicus needs much work. Some 3D printed parts are falling apart and some are broken. He needs a brand new face with new servos and sensory "organs". The Pololu driver used here has 24 outputs and it can control some of thes servos. However, there will be more servos to control when the whole body is assembled. Therefore more drivers will need to be integrated.

With this project, Copernicus speaks through a speaker as his mind is in a PC. Initial plan was to give Copernicus a Raspberry Pi brain running on ROS. Time wouldn't permit to complete this task with a testable result. So, snippets of code and AIML code used to build Jeeves robot were used.

This report shows how speech synthesis and hand gestures can be synchronized using a simple python code. This can help get the Copernicus project responsive and encourage others to complete it using an even more powerful and complete platform, ROS.

We will start with speech synthesis and AIML integration and show how to create a bot that can be used on any robot to give random response through speech. To make this work, a linux platform (Ubuntu 16.04 LTS) is used here and is used and explained. Installation of software is required in the following sections.

2. Festival

Festival is an open source tool that takes textual input, and synthesizes speech synthesis a speaker. Festival supports many languages and is based on the Edinburgh Speech Tool Library. Festiva has two modes Text to Speech(TTS) and command mode. “In command mode, information (in files or through standard input) is treated as commands and is interpreted by a Scheme interpreter. In tts-mode, information (in files or through standard input) is treated as text to be rendered as speech. The default mode is command mode, though this may change in later versions.” Next we are going to install festival. Home directory is assumed to be the working directory in this report. You may,however, choose your own folder as long as you are consistent with all other files that we will include later.

To install festival type the following in terminal:

sudo apt-get install festival

To test it type anything and save it as .txt file in home folder. Then type this in terminal:

festival --tts text.txt

Festival should narrate your text file, if installation is successful.

3. Artificial Intelligence Markup Language (AIML)

AIML is an XML based script language that was developed by the A.L.I.C.E. Foundation. AIML provides a simple yet powerful method to creat a chatbot using ALICE's 41000 categories. It is possible to create AIML from scratch but it is much easier and more responsive to modify and add to the open source ALICE AIML. In our case, we have modified Jeeves AIML which itself is a modified ALICE AIML. Below is an example of AIML skeleton taken from Josh Sackos report on Jeeves AIML.

```
<category>
.
.
Pattern and response tags
.
.
</category>
```

1 - AIML category example.

```
<category>
        <pattern>HI JEEVES</pattern>
.
.
.
Response tags
.
.
.
</category>
```

2 - AIML pattern example.

```
<category>
        <pattern>HI JEEVES</pattern>
                <template>
                        <random>
                                <li>greeting command Hi there</li>
                                <li>greeting command Hello there</li>
                                <li>greeting command Greetings</li>
                                <li>greeting command Good day</li>
                        </random>
```

```
            </template>
</category>
```
3 - AIML randomized response example.

```
<category>
        <pattern>HELLO JEEVES *</pattern>
                <template>
                        <srai>HI JEEVES</srai>
                </template>
</category>
```
4 - AIML linked response example. Points to category “HI JEEVES”.

This skeleton and examples show how simple the AIML format is to generate response and to randomize it. It can also refer to a category from a separate location. The “random” and “li” keywords enable randomization of responses where the “srai” keyword creates a reference to a category mentioned. Since we are using python here, AIML installation using python is discussed next.

4. Python

Python is used here to create a simple speech synthesis bot that synchronized speech with hand movement for copernicus robot. This python file uses AIML file and festival program alongside Pololu Maestro program. Here is installation of required steps.

Install Python:

```
$ sudo apt-get update
$ sudo apt-get install python3.6
```

Then install pip. Pip is a crucial third-party Python package that allows you to install and uninstall any compliant Python software product with a single command.

To install pip, securely download get-pip.py. [1]:

```
$ curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
```

Inspect get-pip.py for any malevolence. Then run the following:

$ python get-pip.py

Now we are ready to install AIML:

$ pip install python-aiml

Now the AIML part is complete. Importing AIML file with some basic python code would give you an output. Python code is supplied.

Next Copernicus' Arm is added to the python code to synchronize speech with gestures. To do this maestro.py file is downloaded from pololu website. After importing this file into python a simple call as shown below controls the servos:

```
servo = maestro.Controller()
servo.setAccel(0,4)      #set servo 0 acceleration to 4
servo.setTarget(0,6000)  #set servo to move to center position
servo.runScriptSub(11)   # 11 is gesture sequence number as listed below
                        # from pololu Maestro software
```

```
Subroutines:
Hex Decimal Address Name
00  000     0000    COUNT_FWD
01  001     0033    COUNT_REV
02  002     00AE    THUMB
03  003     00B2    POINTER
04  004     00B6    MIDDLE
05  005     00BA    RING
06  006     00BE    PINKY
07  007     00C2    WRIST
08  008     00C6    GRAB
09  009     00D4    RELEASE
0A  010     00E2    HAND_CONTROL
0B  011     00F2    HANDWAVE
0C  012     010B    WAVE1
0D  013     011B    WAVE2
0E  014     012B    WAVE3
0F  015     013B    HAND_WRIST_CONTROL
10  016     014E    IMCOPERNICUS
11  017     0175    ALLINBUTPOINTY
12  018     0185    INIT
13  019     0195    OKAY
14  020     01AD    YOHO
```

Next,we'll show you how to create the gestures and use the pololu Maestro software.

5. Arm Design

5.1 Component List

As Copernicus is based on InMoov design, to complete the arm design we referred the InMoov - Build Your's webpage that gives a detailed parts list required for all the design components. Based on the status of arm we created a list of parts to be 3D printed for completion of the arm design. components referring to the InMoov's Build Your's body parts library. There were also some broken parts that need to be reprinted. Following table lists various parts to be reprinted

PART	COMPONENTS
Wrist Repair	Bolt_entretoise7 RotaWrist3V3
Bicep	GearHolderV1 HighArmSideV3 LeftRotTitV4 LeftRotcenterV3 PistonanticlockV2 PistonbaseantiV2 PivPotentioRoundV3 PivPotentioSquareV3 ReinforcerV2 RibonPusherV1 RotGearV6 RotMitV3 RotTitV4 RotWormV5 RotCenterV3 elbowshaftgearV1 gearpotentioV1 lowarmsideV1 servibaseV1 servoholderV1 spacerV1
Shoulder	ClaviBackV2 ClaviFrontV2 LeftPivcenterV3 LeftPivMitV2 LeftPivPotholderV3 LeftPivTitV3 LogoLeftMyrobotlabV2 LogoRightMyrobotlabV2

PistonbaseV6
PistonClaviV3
PivConnectorV1
PivGearV6
PivMitV2
PivPotentioRoundV3
PivPotentioSquareV3
PivPotholderV3
PivTitV3
PivWormV5
RotcenterV3
servoholderV1
servoHolsterV1

However, due to limited availability of the printer we were not able to complete the 3D printing. So, Melih suggested to repair the existing wrist of right arm and get the fingers and wrist to function correctly.

5.2 Dynamics and Servo Interface

To exercise the dynamics of the arm we tried to interface the servo motors with HROS1 used in our first project initially, however, it would have meant creating a completely new software structure for the Copernicus arm. So, we decided to use Maestro Mini Servo control software from Polulu to study the dynamics and create a functional arm.

We had to adjust tension of all the strings attached to the servos to get the fingers and wrist moving. We had to iteratively change the tension based on direction of motion because the forward and reverse action strings were not a single thread but a joint as a knot of the string at the far end tip of the finger. This needed extra angular motion to offset the torque loss at the joint.

We studied the Polulu scripting interface and created various motion subroutines .

The Pololu script is easy to pick up, and their website[1] provides good basic information on how to use their code. One of our teammates made a sleep deprived video walking through the use of the code, and those videos should be made available to you.

One important concept of the Pololu code is the use of the stack. For efficiency's sake, you can input nearly all your numerical commands at the start of the script. These numbers are stored, and displayed, on the stack. The two most used command in Pololu are *servo* and *delay*.

The *servo* command takes the top two numbers from the stack as arguments. The first number it takes it will use to identify which servo you want to affect, and the second number will be used to represent the position the servo should adjust to. This is demonstrated in the video.

The *delay* option is also useful. It takes the top number from the stack and delays according to the magnitude of the number it takes. If you tell a servo to turn from an original position and then return to that original position, the servo may not respond as the code will immediately tell the servo to go back to its original position before the servo ever moves. The use of *delay* between the two commands can solve this issue. Another great solution is to set the speed and acceleration of the servo, but we did not use this in our project.

Covered in the video is the use of subroutines. A subroutine simply jumps the compiler from one part of the code to another, and then returns after running whatever code it jumped to. Consider using subroutines as a makeshift variable. Call a subroutine that merely sets up the stack with numbers you need.

---

[1] https://www.pololu.com/docs/0J40/6

We created various gestures using the Pololu script to enact the Copernicus. This involved determining the range of individual motor to complete the action. Using the Polulu software's manual slider feature

for setting the motor position we derived following table to determine the range of play for each motor and accordingly set the motor positions at desired values for the gesture.

		min.	max.	normal	wave-1	wave-2	wave-3	allinbutpointy	pointyin	ok	grab	yoho
thumb	servo 0	208	1600	800	800	800	800	1451.25	1451.25	1600	1221.5	208
pointer	servo 1	256	1504	650	650	650	650	256	855.75	1304	1188.75	256
middle	servo 2	368	1504	650	650	650	650	1305.25	1305.25	842.25	1350	1140
ring	servo 3	304	1696	770	770	770	770	1601	1601	945.75	1696	1493.25
pinky	servo 4	304	1744	800	800	800	800	1702	1702	1100.75	1744	304
wrist	servo 5	304	2352	1800	2200	1250	1450	1800	1800	2252.5	1800	1800

While determining the motor positions, we studied the relation between position of the motor as specified by the 'Status' tab of the Polulu GUI and actual servo motor position that needs to be programmed in the script as:

Actual Position = 4 * Position as on Status Tab

This finally gave us following table of motor positions for script programming

		min.	max.	normal	wave-1	wave-2	wave-3	allinbutpointy	pointyin	ok	grab	yoho
thumb	servo 0	832	6400	3200	3200	3200	3200	5805	5805	6400	4886	832
pointer	servo 1	1024	6016	2600	2600	2600	2600	1024	3423	5216	4755	1024
middle	servo 2	1472	6016	2600	2600	2600	2600	5221	5221	3369	5400	4560
ring	servo 3	1216	6784	3080	3080	3080	3080	6404	6404	3783	6784	5973
pinky	servo 4	1216	6976	3200	3200	3200	3200	6808	6808	4403	6976	1216
wrist	servo 5	1216	9408	7200	8800	5000	5800	7200	7200	9010	7200	7200

6. Links on Google Drive for Reference

Demo Video : https://drive.google.com/open?id=1BI-6bch9EawVXh_yHGvdopOfjfkZB6nc

Script : https://drive.google.com/open?id=1VeflHycRMxZlHn0Ik48bquiu6i5cmqGi