Central Processor for Prezence

Helen Root

***Abstract*—The Central Processor plays a crucial role in the execution of Prezence, an automated presentation delivery training system. This report examines the current research in the field of action selection architectures to inform the design of the Processor. Additionally, it considers research to outline an ideal model of presentation delivery, based on the metrics chosen to be measured, in order for the Processor to be able to command relevant feedback for the user.**

I. Introduction

Prezence is a robotic system for coaching people in presentation delivery. Aldeberan's NAO humanoid will be used to deliver feedback to the user regarding various metrics we have identified as being important. The design of the system is highly modular therefore it is vital to have a Central Processor to aggregate processed data from the speech processing and computer vision components, decide what feedback to give and send commands to the audio-kinetic feedback module which controls the NAO. In order to make this decision, we are faced with the "action-selection problem" [2], the need to determine which action is to be performed next according to input data. This report will discuss how the precise design of the Central Processor is informed by literature in the field of action selection architectures. This report will also describe how thresholds were chosen for various metrics according to available research, or through future experimentation where research is lacking, in order to inform the selection process.

A. Requirements and constraints

In order to design the architecture, requirements and constraints need to be identified. Firstly, as mentioned, the design must be modular. This allows the members of the group to work relatively independently on their own components, thus making the development process more efficient. Modular designs are also often used in software engineering as it helps to break a challenge down into more manageable pieces. Additionally, modularity allows for extensibility in the future should we or others decide to build upon what we have created.

Related to modularity, it would be preferable for the design to be flexible. In this context, flexibility means decoupling the components from the Processor to allow developers to change large amounts of their code to maintain independence during development. Furthermore, it would be ideal to create a system that would allow future components to be declared without having to rewrite the Processor to give us or others the freedom to build upon the system easily.

The system should run quickly to be able to give relevant feedback during a presentation. This is more important for some actions rather than others.

We have defined many metrics we intend to monitor, requiring the processing of speech and imaging. This means that the system must be able to run multiple modules in parallel at the same time as the Processor.

This project must be completed within the next 4 weeks, and this must be taken into consideration when building the system. This may result in making sacrifices in other requirements, but this is our most significant limiting factor.

The system must be able to run from an ordinary laptop. We decided that we would not attempt to run Prezence directly on the NAO because of its limited processing power.

The NAO has constraints on how quickly it can move which the Processor needs to take into account to regulate the rate at which it sends commands to the NAO. Otherwise, a build up of actions may take place resulting in feedback being irrelevant by the time NAO performs it. The Processor will therefore have to prioritise input data so that the most useful feedback can be given to the user.

The constraints are of varying importance. The greatest constraint is the time we have to build, so the system should foremost make module integration easy. If this means sacrificing how many metrics can be monitored at one time or making the system slower than real-time, these can be worked around. For example, the former could be solved by focusing on metrics which have the greatest impact on presentation, as found in literature or future experimentation. The latter, real-time execution, can be compromised up to a certain point. If the NAO gave feedback every 1 or 2 minutes, instead of immediately, this could give some leeway to make implementation faster without compromising efficacy too greatly.

II. Background

A. Research

The action selection white paper [2] outlines a number of general approaches to system design. This became the starting point of my research, and helped choose which approaches to research in more depth.

The first approach discussed is the symbolic approach. This focuses on computing "a provably optimal plan" and then executing it. An example of an implementation of this approach is Soar. The paper introducing the Soar architecture demonstrated that this approach would not be the most suitable for Prezence as it is described as a "problem-solving architecture" which "searches until a solution is found" [5]. Prezence needs to give real-time feedback, however, so a symbolic approach would not work for this project.

Next discussed is the distributed approach. In this method, modules determine the "best action based on local expertise" [2]. This style is similar to neural networks research, in that usually there ends up being some "centralised system determining which module ... has the most salience" [2]. This distributed method therefore fits nicely into our modular

design, however puts more responsibility on the speech and computer vision modules. Considering the time constraints of the project, it is more efficient to put the task of action selection in the Central Processor as opposed to the modules.

The white paper then discusses dynamic planning and the various approaches associated with it. Instead of coming up with a set of actions in advance, dynamic planning determines only the next action based on the current context the system is working in. Since feedback during the presentation is only relevant to the most recently processed data, this method appears to be promising. A purely reactive approach would not be appropriate, however, as this does not store any information at all, whereas our post-speech feedback requires an assessment of the user's metrics over the course of the presentation thus requires some persistent memory.

The first technique in dynamic planning that Brom and Bryson discuss is deterministic condition-action rules, or productions. Each production is a rule in the form of **if** condition **then** action and productions can be organised in flat or hierarchical structures in such a way that builds a "conflict resolution mechanism". For example, if a robot had to avoid obstacles above all else, the first production rule it evaluates will check if they sense an obstacle and move to avoid it if true. The following productions are of descending importance. For Prezence, this could be a useful starting point for a system with basic conflict resolution, but this does not allow for more complex, dynamic resolution. This kind of resolution would be necessary when the most recent data read by the Processor is that the presenter's speech accuracy has fallen slightly below the threshold, for instance, but they have been looking down at their notes for the past two minutes. In this situation, it is more important for the NAO to encourage the user to look up than it is to warn them about speech accuracy, so that would need to take priority. Additionally, the flat structure assumes that feedback will still be relevant after an action from a previous production takes place but, in the time taken for the robot to move, the speaker will have continued talking and feedback from the older data may not be relevant anymore.

The next technique within dynamic planning is deterministic finite state machines (FSMs). These, however, describe a system made of states and transitions between those states which does not seem beneficial to Prezence. The reason for this is because the NAO has to always move through a neutral state in order to get the most up to date information to move to the correct feedback state. Therefore it would make for an unusual FSM, which suggests that there are better designs.

The last technique in the area of dynamic planning that the white paper discusses is the use of connectionist networks, such as artificial neural networks. This approach would normally also involve a level of adaptation. This could be an interesting approach, especially when tailoring actions to an individual's taste or prioritising feedback based on common pitfalls, but considering the time we have to implement the project, this is likely to be out of scope.

The paper does not go into detail about the various conflict resolution (CR) strategies in the field. Further research found that Bullinaria suggests that the 5 most common general CR strategies are:

1) Delete the commands, generated by rules, that have already fired.
2) Select the command to fire based on the most recent data read.
3) Choose the first satisfied condition based on order in the code, this approach is similar to the tactical structuring approach described earlier in the discussion of production rules.
4) Prefer the rules for which the greatest number of conditions is met.
5) Randomly select the rule. [3]

These are helpful for basic CR however does not address the desire for the system to dynamically prioritise conditions according to which metric deviates the greatest from the ideal. In this area, Saaty et al. [8] considers the use of weighting the conflicting parties according to which one will have the greatest impact on the outcome. It goes deeper into then weighting objectives within the party, but this extra level is unnecessary for Prezence since all the input metrics are associated with one feedback action, and do not group together to form a new action. Saaty also describes a matrix-based approach to calculate a ranking of best solutions based on the relative weighting between each conflicting party, which was useful in informing the final design choice.

B. Tools

The tools considered were ROS and standard Python. ROS (Robot Operating System) is a popular middleware used in many robotics applications today. There are many useful features described in the paper that introduces it [7]. Particularly salient are the significant support for various methods of debugging, the encouragement of developing modular systems through creation of nodes, the focus on making ROS a tool for collaborative development through the use of packages, the ability to instantiate a cluster of nodes in one command through an XML file and the support for communication between nodes. These are all features that could benefit this project greatly, but ROS is also known, even by its own creators, for its "significant learning curve" [11]. As a result, we decided to look to Python for the initial implementation, so that we can start quickly producing code and leave the option to develop upon it by integrating ROS if we have time. This is feasible because of ROS's integration with Python.

Python is a programming language we are familiar with and it is easy to debug and run. This makes it more time effective than ROS, as we will not have to learn new concepts and debug previously un-encountered issues, so offers a good starting point for a basic system. Additionally, the libraries are extensive enough to allow for powerful support to the input modules, and writing the Processor in the same language will make integration easier.

C. Final design

The design of the system will be a Python program in combination with a shell script which will run the Central

Processor and other modules in parallel. Since Python does not have a dedicated method for intra-script communication like ROS does, each module will write to an individual text file. The name of these text files will be discovered from a configuration file, detailed further in this report, that the Processor will read during the initialisation process. There are benefits and drawbacks to this method of sharing data. The drawback is that this could be memory intensive during long presentations. Assuming the modules wrote approximately the same amount of data in a given time period then the memory usage would vary linearly with time taken for one module, but exponentially per module added. If a module wrote to a text file every 10 seconds with a float value rounded to 2 decimal places, assuming that one number takes one byte in memory as is the case for UTF-8 formatting [12], then each line will take 6 bytes in memory (4 for the number and 2 for the newline character), resulting in 360 bytes of memory used per module for a 10 minute presentation. This is not very large, so has been deemed as an acceptable cost. The benefit of this method allows there to be persistent data about the presentation, which can easily be re-read and processed all together at the end of the presentation to inform post-speech feedback. Since this is a fundamental feature the drawback is outweighed. The other option of saving the data dynamically in the program and overwriting text files so there was only ever one line in persistent memory was considered, but ruled less favourable as it would increase the complexity of the data structures in the Central Processor and would mean the system was less tolerant to failure, losing information that could still be helpful to the user even if the system crashed before the end of the speech.

As discussed previously, the final architecture will employ reactive planning over classical, also known as symbolic, planning because the NAO will need to provide feedback in real-time and giving feedback to a presentation does not result in a provably optimal approach in the same way that navigating through space, for example, does. A centralised approach is chosen over a distributed approach to be able to more easily split tasks up between developers and achieve more. The Central Processor will use production rules in deciding which action to take based on the input that is considered of the most importance after CR takes place. There is an implicitly hierarchical structure introduced by the modular nature of the whole system, for example the lower-level motor control of the NAO is represented as a single command in the Processor, but, with regards to conflict resolution, the architecture is essentially flat.

To tackle the problem of CR, Prezence will make use of several of the discussed methods. In the absence of two or more inputs deviating from the model of ideal delivery, which will be defined later in this report, the system will respond to the most recent feedback read from the files written to by the other modules, in accordance to the 2_nd strategy Bullinaria recognised. Because files will be continually written to, thus staying up to date, there is no need to explicitly code for Bullinaria's 1^st CR strategy. The calculations that Prezence must make, however, can be simplified from the matrix approach of relative importance values suggested by Quigley to a weighting directly proportionally to how far, as a percentage, the user's behaviour deviates from the delivery model. This simplification will make the calculation process faster to enable more immediate feedback to the user.

I have already implemented a script which demonstrates the feasibility of the basic communication between modules using dummy modules. I have begun to implement configuration of acceptable thresholds, as can be seen in the example configuration txt file below. This makes the Processor aware of a number of things including:

1) The file name each module will write to.
2) The type of condition checking to perform for that module (eg. max, min, in range).
3) The value(s) the user's data should stay within.
4) Whether the Processor should be strict with feedback or not.

```
speech_speed inRange 140 180
speech_accuracy min 0.8 strict
cv_headGaze min ANGLE
```

The next steps would be to implement the more sophisticated approaches to strictness and weighting of inputs in order to achieve the CR approaches described in this report. For something like head gaze, the input data will be a gaze angle, written at regular intervals, and ANGLE in the file below will be configured for the according to the room, an elaboration on this reasoning will be given in the next section. The modifier of 'strict' indicates that if that metric is outside the allowed range once, then the user should be warned immediately. Other metrics may be more allowing, instead adding to a weighting to increase the priority of that feedback every time it is monitored, with a negative weighting for every time it isn't measured. In this way, if the user looks down briefly and it is captured once but not captured again for a long time, feedback will priorities will reset away from improving gaze. On the other hand, if the user is looking down for a long period of time, then the weighting will keep increasing until it takes priority over other inputs.

The other next step is to make the Processor aware of the feedback commands that can be written to the text file the feedback module reads from. In the simple, purely Python approach, the configuration file will also have to contain these commands and what input metric variation they relate to. For example, the line for speech_speed can be added to to include slowDown and speedUp commands at the end of the line. The program will then have to map these to the commands to give when the input goes below the range or above the range respectively. In a more sophisticated approach, the introduction of ROS would allow for subscribing and publishing to topics available to all nodes, which would help coordinate this mapping of feedback to input modules and allow for a more flexible approach since the process could be automated instead of manually updating text files to stay in line with the features implemented.

III. Model of ideal delivery

In order to give feedback to the user, the Central Processor needs to have a defined model for ideal delivery with which

to compare the incoming data against. The feedback would then be tailored to try and get the user to fit the model. The model will be a set of values or ranges of values for each metric measured for the user to attempt to move towards. Some of these values can be found in research, others will require experimentation to be identified.

The metrics as outlined in the preceding design report for this project are as follows:

- Speech speed measured in words per minute
- Speech volume measured in decibels
- Speech clarity from software confidence levels
- Time taken vs time allocated in percentage over or under
- Head gaze measured in degrees
- Gestures measured by frequency of gestures performed in a given time period

A. Background research

The effect of speech speed and other non-verbal cues on how audiences perceive the speaker has been studied by many researchers. The studies focus on how different rates affect different aspects of perception. As part of Miller et al's 1976 study, the team found that speakers who spoke faster, of three rates, were perceived as more intelligent, knowledgeable, objective and persuasive [6]. Smith et al studied a larger range of rates and found that apparent competence increased monotonically with rate increase, but apparent benevolence peaked around a normal speaking speed [9]. Apple et al found that an "inverted U" could be seen when graphing rate vs truthfulness of the speaker, with the slowest speaker judged as less credible than the fastest [1]. On the other hand, Burgoon et al failed to find a significant relationship between tempo and credibility [4]. The only paper that mentioned exact rates in words per minute, as opposed to the increase from an unknown "normal" speaking rate as in [9], was Miller, [6], who defined the three speeds as 191, 140 and 111 wpm. Since the research results tend to say that speed makes perception of the speaker more favourable, but not conclusively, I will start by choosing the range of 140 to 180 wpm as the ideal range. The lower maximum rate will hopefully avoid the diminishing effect seen at greater speeds.

Speech volume can be measured in decibels at the point where the laptop is set up in the room. The initial, naive approach would be to first make sure that the measured volume stays above 60 dB, which is typical of normal human conversation [10]. A more sophisticated approach would allow for the user to input in some way the maximum distance an audience member may be from the speaker and estimate the sound level at that point based on the inverse square law. Any more sophisticated approaches of modelling sound reflections in different kinds of room are too complex to be in the scope of this project.

Head gaze will be measured instead of the eye gaze previously stated in our design report because at the distance at which the Kinect will have to be to monitor body movement, it would not be able to measure eye gaze. Head gaze, however, can still give the system an indication of if the user is looking at the audience or elsewhere. The effect of eye gaze on audience's perception of the speaker has been studied, for example Burgoon et al found that primarily increased eye contact, and facial pleasantness, resulted in increased character and sociability judgements [4]. Specific durations were not recorded, however, so we will have to fine tune the threshold experimentally. The angle of the head will be passed to the system so that we can configure for different room set ups, because in some settings one angle may be the speaker looking at a teleprompter whilst in a different room it could suggest the speaker looking on students in a classroom.

B. Experiments required

Speech clarity can be fine-tuned in implementation. We have already found, however, that the software has very high confidence intervals beyond the expected human ability to understand, therefore the threshold will be very high and the user will have to stay above that threshold.

The metric for time taken vs time allocated is very straightforward. This will output a measure in the post-speech feedback which will let the user know if they have gone too far over or under their desired time, as input before the presentation begins. An additional feature in the robotic feedback module could warn the user how much time they have left by raising a number of fingers to indicate the minutes remaining.

The acceptable angle for head gaze will be determined according to room size. The time allowed for the user to be looking down will be small, but experiments to decide how long is acceptable need to be carried out.

REFERENCES

[1] William Apple, Lynn A Streeter, and Robert M Krauss. "Effects of pitch and speech rate on personal attributions." In: *Journal of Personality and Social Psychology* 37.5 (1979), p. 715.

[2] Cyril Brom and Joanna Bryson. "Action selection for intelligent systems". In: *European Network for the Advancement of Artificial Cognitive Systems* (2006).

[3] John A. Bullinaria. "IAI : Production Systems". In: (2005).

[4] Judee K Burgoon, Thomas Birk, and Michael Pfau. "Nonverbal behaviors, persuasion, and credibility". In: *Human communication research* 17.1 (1990), pp. 140–169.

[5] John E Laird, Allen Newell, and Paul S Rosenbloom. "Soar: An architecture for general intelligence". In: *Artificial intelligence* 33.1 (1987), pp. 1–64.

[6] Norman Miller et al. "Speed of speech and persuasion." In: *Journal of Personality and Social Psychology* 34.4 (1976), p. 615.

[7] Morgan Quigley et al. "ROS: an open-source Robot Operating System". In: *ICRA workshop on open source software*. Vol. 3. 3.2. Kobe, Japan. 2009, p. 5.

[8] T.L. Saaty and J.M. Alexander. *Conflict Resolution: The Analytic Hierachy Approach*. RWS Publications. ISBN: 9781888603200. URL: https://books.google.co.uk/books?id=CY0FAQAAQBAJ.

[9] Bruce L Smith et al. "Effects of speech rate on personality perception". In: *Language and Speech* 18.2 (1975), pp. 145–152.

[10] Steven W Smith et al. "The scientist and engineer's guide to digital signal processing". In: (1997).

[11] Dirk Thomas. *ROS/Introduction*. May 2014. URL: http://wiki.ros.org/ROS/Introduction.

[12] Wikipedia. *UTF-8 — Wikipedia, The Free Encyclopedia*. [Online; accessed 17-11-2016]. 2016. URL: %5Curl%7Bhttps://en.wikipedia.org/wiki/UTF-8%7D.