A Lightweight Artificial Cognition Model for Socio-Affective Human-Robot Interaction

Andrés A. Ramírez-Duque
University of Glasgow
Glasgow, UK

Alan Lindsay
Heriot-Watt University
Edinburgh, UK

Mary Ellen Foster
University of Glasgow
Glasgow, UK

Ronald P. A. Petrick
Heriot-Watt University
Edinburgh, UK

ABSTRACT

The software submission presents a fully working artificial cognition model, which controls a NAO social robot. The model was specifically designed to control a socio-affective companion robot for use in a medical setting. It was deployed using embedded hardware: a Raspberry Pi 4B and a Jetson Nano Board, and an external RGB-D camera. Based on the ROS operating system, this software package includes components for social signal processing, behaviour selection, affective behaviour rendering, and a web-based user interface. The robot's behaviours are selected by a planning system, which generates the robot's behaviours based on the state of the interaction, the progress of the medical procedure, and the user's affective state. The system has been tested in simulated environments and is currently being used in two clinics to perform a usability test and will subsequently be used to carry out a series of clinical trials.

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics.

KEYWORDS

Social Robotics, ROS, Cognitive Architecture, Embedded Hardware

ACM Reference Format:
Andrés A. Ramírez-Duque, Alan Lindsay, Mary Ellen Foster, and Ronald P. A. Petrick. 2024. A Lightweight Artificial Cognition Model for Socio-Affective Human-Robot Interaction. In *Proceedings of the 2024 ACM/IEEE International Conference on Human-Robot Interaction (HRI '24), March 11–14, 2024, Boulder, CO, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3610977.3637469

1 BACKGROUND AND SUMMARY

This system was implemented as part of a project aiming to develop a companion robot to assist children during a painful and distressing medical procedure. In the specific clinical scenarios that we are targeting, the robot is placed in a small room together with the patient, along with one or more carers and a Health Care Provider (HCP) during the course of a single clinical procedure. We have developed a fully functioning companion robot for operating in this scenario, which was designed using both a co-design (involving several stages and children, parents and HCPs) and targeted meetings between the technical team and the HCPs. The robot positions itself as a friendly and supportive companion, setting out positive expectations. It can present various supportive behaviours, including diversions and humour, practising coping strategies, role modelling, and providing positive reinforcements. Its behaviour is controlled by a planning system, which conditions its action selection based on the user's affective state.

The proposed cognitive model can be used and extended to other social robots and other scenarios in healthcare. Our implementation offers a simple and practical integration of basic functionalities in a prototype that can explain the user's mental state and analyse the effect of actions on it. This software can be used as a starting point for prototyping social robots with the ability to automatically adapt the execution of their behaviours by reasoning and managing the user's affective state.

2 PURPOSE

The primary function of the cognitive model is to use the human-robot interaction paradigm to produce behaviours in the robot that help manage the affective state of a child undergoing stressful clinical procedures. This means that our system must assess how the affective state detection and the robot's behaviour realisation can support an intervention that responds to the child's emotional state. To achieve this, we use a lightweight architecture that comprises several components, such as social signal processing, an interaction manager, a plan-based affective manager, a memory module, and an affective behaviour renderer. The architecture also includes a web-based interface that complements the system by providing a mechanism to control the interaction. Furthermore, it provides an alternative input and feedback module, allowing an operator to monitor and update the affective state as necessary.

The system consists of several additional hardware components. The complete system includes embedded hardware to improve the NAO processing capabilities. An external RGB-D camera has also been added to complement the NAO's limited internal sensors. The entire system has been developed using the Robot Operating system (ROS) [8].

HRI '24, March 11–14, 2024, Boulder, CO, USA

ACM ISBN 979-8-4007-0322-5/24/03…$15.00
https://doi.org/10.1145/3610977.3637469

The social signal processing system constantly estimates the child's emotional state and attention towards the robot by analyzing its head posture and location. This information is provided to the planner on demand, generating and updating plans and actions based on the affective sensing data. The web app performs various functions, such as loading the system, entering the child's name, and initiating a new top-level goal for the planner. During the robot intervention, the operator used the web interface to indicate the different stages of the procedure, such as the beginning of the pre-procedure phase. The interaction manager requests an action from the planning system each time a procedure step is initiated, and it executes the action by calling the appropriate modules, such as requesting a behaviour from the robot or querying the user's state.

3 CHARACTERISTICS

Our model of artificial cognition is developed using ROS Melodic and Ubuntu 18.04 LTS to ensure compatibility between the signal processing and planner components. Each module is wrapped in a ROS node and can be executed on embedded platforms. We use a Raspberry Pi 4B to host the ROS master, the planner and web server nodes, and a Jetson Nano that handles video processing tasks exclusively. The ROS master node and a Python HTTP server automatically start up when the Raspberry Pi boots. The 'gui_management' process executes the other nodes using a roslaunch command when the web interface requires them. The Raspberry and Jetson Nano are connected through a local Wi-Fi network, and all the nodes on the Jetson Nano are executed remotely from the ROS master host.

Listing 1: action_chain.msg

```
int32 caller_id
int32 plan_step
string action_type
string[] parameters
string speech_cmd
diagnostic_msgs execution_status
```

The interaction manager and the planner are at the centre of the ROS graph (see Figure 1); these two processes are responsible for the reasoning loop. On the other hand, the process related to the user's feedback is managed between the interaction manager and the web-based user interface. To optimise those loops, we create two ROS messages, 'action_chain.msg' and 'web_chain.msg', that allow us to propagate helpful information and diagnose the interaction. All interacting nodes share the 'action_chain' message (see listing 1) throughout the reasoning loop; for example, the NaoQi driver was modified to listen for these messages and maintain consistency throughout the loop. This message has three string fields and a 'diagnostic_msgs' field. The planner fills the 'action_type' and 'parameters' in each execution step, while the 'tobo_core' node fills the 'speech_command' field, and all are propagated until the NaoQi driver.

Listing 2: web_chain.msg

```
int32 plan_step
string request_type
string[] parameters
float32 duration
```

The 'web_chain' message comprises four fields, as seen in Listing 2. Once the planner generates an action requiring the user operator's intervention, the interaction manager enquires the web interface using 'request_type' and the appropriate 'parameters'. The duration of this request is represented in the last field of the message. Once the time is exceeded, the web app responds with the default option, which can be any specific value and is defined in each web request.

The 'next_action' and 'animated_speech' topics use 'action_chain' messages, while the 'listener_req' and 'request' topics use 'web_chain' messages. The other topics use standard ROS messages (See Figure 1). The 'get_an_action' and 'get_sensor_value' services use a modified version of the 'std_srvs/setBool' service that receives a 'step_plan' in integer format instead of a Boolean.

3.1 Knowledge database module

Managing the knowledge database is an essential characteristic of any cognitive model. In our case, the database comprises four components: the domain knowledge that is represented by a static part and is read when all the processes start and the dynamic feature that allows modification of the system's state based on the sensor readings and actions' effects on the planner's world model. Furthermore, we include the behaviour library and the action hierarchy to complement the knowledge database (see Figure 1). We have simplified the deployment of the knowledge database by employing the ROS parameter server. The ROS parameter server is a shared memory space accessible using XMLRPC, which runs inside the ROS Master. It is globally viewable, allowing any node to store and retrieve parameters at runtime [8]. However, to prevent simultaneous access or modification of the same memory space by the different components, the interaction manager implements higher-level threading interfaces and provides a thread-safe interface for exchanging data between running threads.

3.2 Interaction manager

The interaction manager runs several loops to check and update the system states. The first internal loop begins when the interaction manager requests the planner for a new action. Depending on the system's state, the plan can generate reactive actions of the type 'do_activity', 'query_response' or 'request_sensor_value'. If a 'do_activity' action is published, the behaviour renderer listens to the action and processes it, searching the behaviour library for dialogues and body movements that match the action. Thus, the 'tobo_core' transforms these into a command that the NaoQi bridge can execute; once the robot finishes the execution, NAO emits a signal publishing in a topic that lets the interaction manager know that the cycle has ended, and then it updates the system states. In the case that a 'query_response' is published, the interaction manager listens to it and sends a web query using the 'listener_req' topic. The web server executes the respective callback, processes the query using popup messages to capture the user's attention, and, once the user responds, publishes the feedback in the 'request' topic. Thus, the interaction manager receives the message, updates the system state again and continues requesting a new action. Finally, the loop that allows gathering information using sensing elements is triggered in the interaction manager by a 'request_sensor_value' action.

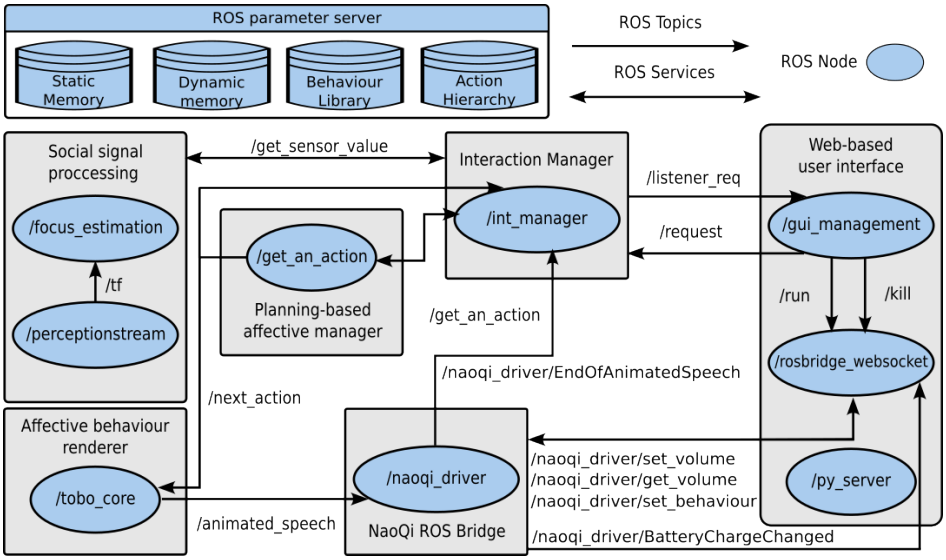

Figure 1: ROS Graph illustrating the nodes structure, topics and services developed

When this situation arises, a call is made to the 'get_sensor_value' service, and then the sensing module sends the estimated value for the sensing variable (engagement or anxiety).

Listing 3: service_provider class

```
class service_provider(object):
  def __init__(self):
  def initialise(self, action_broadcast_f,
  webserver_broadcast_f, nau_broadcast_f):
  def request_action(self, ...):
  def ask_for_user_input(self, ...):
  def on_received_planner_action(self, ...):
  def on_received_webserver_message(self, ...):
```

We have developed a simple API-style module definition that allows us to register a Python-based 'service_provider', including each interaction manager module (see Listing 3). The planner, the web server, the knowledge database, and the sensing and rendering actions represent the basic modules of our model. We can create independent dummies of each module and test them separately. This helps us to isolate dependencies such as ROS or Gstream.

3.3 Planning-based affective manager

We have developed an on-demand ROS service to wrap the planner, similar to the ROSPlan [2] system. The interaction manager evaluates the system's state, and when it determines that it is time to perform a new action, it calls the service. The service diagnoses the execution and publishes the next action chosen by the planner on the 'next_action' topic using an 'action_chain' message. Thus, the 'get_an_action_service' iterates through the plan (or replans) and feeds the pipeline with a new 'action_chain' ROS message on demand.

Our approach is built around the PRP planning system [7], which supports fully observable non-deterministic (FOND) planning models [7]. In our approach, propositions model the situation in the room and state of the procedure (e.g., a health care provider is in the room), abstract information (e.g., that a certain behaviour has already been used), and affective state (e.g., anxiety or engagement of the patient). Actions can be separated into four groups: robot behaviour, procedure updates, implicit signals and explicit queries. The robot can perform a range of actions, including distracting actions (e.g., dancing) and calming and instructive actions (e.g., stepping through breathing exercises), each of which is represented in the planning model.

We implemented our own plan manager using the API-style components definition (see Subsection 3.2) to allow for greater flexibility around the management of the state. We have incorporated features to simplify the transition from simulation to real-world deployment, particularly to allow early deployed testing. The system has, therefore, been designed to allow parameterisation so that elements of the planning model can be associated with alternative transition implementors. In the current system, the options are modelled, GUI provided, and sensed. These labels determine how each element is updated when an action is applied. Modelled elements are updated using information from the planning model; the values of world-determined (sensed and GUI-provided) elements are gathered after the action has been applied, either by a predictive model or by querying the user. More detail is provided at [6].

Our planning system has also exploited recent work in PDDL modelling [6], which supports inheritance in action definitions. As a byproduct, each action in the planning model has an associated hierarchy of super actions (see Subsection 3.5).

3.4 Social signal processing module

Our signal processing module was built using the Deepstream SDK as a core component. This allowed us to take advantage of its streaming analytic features. Deepstream applications are based on the GStreamer architecture, which provides code blocks in the form of easy plugins to connect, configure, and deploy. The main

benefit of Deepstream plugins is that they are designed to perform hardware-accelerated tasks.

Our video stream pipeline captures RGB frames (1920×1080) from a ZED2 camera at $30fps$ using the 'zedsrc' plugin. Once the frames are in memory, they are batched, scaled, and converted to Jetson-NV12 format. The first inference for face detection is performed using a RetinaFace [3] model with TensorRT from the 'Gst-nvinfer' plugin. The 'Gst-tracker' plugin is used for face tracking just after face detection; then, a landmark detection model [5] is applied using 'Gst-nvinfer' configured as a secondary inference process, optimising the inference time. Metadata such as bounding boxes, region of interest (RoI), landmarks, and head 3D pose [1] are attached to 'Gst-buffer' using the standard structure 'NvDsBatchMeta'. Finally, two additional inferences are performed in cascade to estimate facial expression and gaze. We use EmotionNet, a classification network with five fully connected layers, which NVidia trained on the MultiPie dataset to classify six emotions. GazeNet detects the patient's gaze vector and point of regard, and it was trained on a Nvidia proprietary dataset. Our pipeline offers two options to process the output: showing the bounding boxes, landmarks, and gaze on the screen or publishing the user's gaze vector as a ROS coordinate frame using a 'tf' broadcasting transform.

Our DeepStream application has an additional feature that allows us to capture raw data frames in real time and save them into a file. The above is achieved by dynamically reconfiguring the subpipeline and linking its pads at runtime. The subpipeline comprises a hardware-accelerated video converter, an H264 encoder, a Matroska muxer, and a filesink element. A ROS service called 'recording_service' triggers the recording process.

3.5 Affective behaviours renderer module

The behaviour renderer has two main parts. The first part is responsible for receiving an 'action_chain' message and interpreting it. It then converts the message into a tuple that consists of a dialogue and body movements or expressions. This tuple is selected from a 'library of behaviours' that has been built and fine-tuned by hand with the stakeholders. The goal is to ensure that the expected effect on the user's emotional state corresponds with the planning model. The second part of the renderer takes the dialogues and behaviours and sends them to the 'NaoQiBridge' using the 'animated_speech' topic.

In the planning component (see Subsection 3.3), we noted that the planning model describes an action hierarchy. We observe that in many cases, the distinctions made in the low levels of the hierarchy result from the specific representational details in the planning model, which are not reflected in different robot behaviours. As a consequence, we associate the robot's behaviours with the appropriate level in the hierarchy. This has two benefits: the behaviours are defined at the most appropriate level, saving duplication, and it adds robustness because if when a new action is added, no behaviour is associated with the action, then the system will still continue, using behaviour from higher in the hierarchy. See [6] for more information.

Given the requirements of the renderer, we increase the essential functions of the NaoQi_Bridge by implementing a series of services and additional topics that allow monitoring and controlling directly from ROS, the LEDs, the language, the aliveness and posture features, volume, and battery, among others. One of the main modifications was creating a topic that publishes when the NaoQi 'EndOfAnimatedSpeech' event is triggered internally, allowing the external execution loop to be closed. We also implement a service that directly controls the NaoQi 'AlBehaviorManager' module, allowing us to stop running behaviours and speech calls anytime.

3.6 Web-based user interface

We created a web application interface that enables the user to control the system and intervention through any web browser. The web app is stored on a Raspberry Pi and is deployed using the Python3 HTTP Server package. To enable the web app to interact with ROS, we integrated the JSON API RosBridge Suite package, which allows registering a WebSocket to channel ROS topics with callback functions in JavaScript. Additionally, we added a web-terminal emulator to access the CLI ROS functionality without requiring additional browser plugins.

The web application comprises four pages: a welcome page, a configuration page, a personalisation page, and a dashboard page. The welcome page introduces the project and the research team and provides an interaction overview. The configuration web page enables the user to monitor the position of the external camera and adjust it based on the user's location. The above is achieved by implementing a video stream module that uses the HTTP Live Streaming (HLS) protocol on the front-end. A Gstream pipeline is implemented on the back-end with a UDP sink as its final element. The personalisation page allows the user to register fields such as the institution name, username, and age to modify the content of the behaviour library accordingly. The dashboard is the primary communication channel between the planner and the operator during the intervention. The user can connect/disconnect from ROS, start/stop the intervention, receive and respond to planner queries, control the robot's volume, view the battery level, and manually execute some pre-established behaviours.

4 SOFTWARE

The software submitted in this work is stored as a ROS meta-package in public GitHub repositories[1]. The planner details are in the 'tobo_planner/planning_service' package, and the specific domain model used by the planning-based affective manager is in the 'model0.9' subdirectory. Note that this repository does not include third-party modules such as 'planner-for-relevant-policies' and 'naoqi_driver'. The system requires extensive but non-critical configuration to ensure all components function correctly. Therefore, the meta package provides two files, 'rasp_deps_config' and 'jetson_deps_config', containing instructions to configure each board manually.

5 USAGE NOTES

The designer team includes experts in AI and robotic ethics who are guided by a moral commitment to carefully consider the interests of stakeholders, for more details see [4].

[1] https://github.com/andres-ramirez-duque/HRI2024-Metapackage

REFERENCES

[1] Tadas Baltrusaitis, Amir Zadeh, Yao Chong Lim, and Louis Philippe Morency. 2018. OpenFace 2.0: Facial behavior analysis toolkit. *Proceedings - 13th IEEE International Conference on Automatic Face and Gesture Recognition, FG 2018* (2018), 59–66. https://doi.org/10.1109/FG.2018.00019

[2] Michael Cashmore, Maria Fox, Derek Long, Daniele Magazzeni, Bram Ridder, Arnau Carrera, Narcis Palomeras, Natalia Hurtos, and Marc Carreras. 2015. Rosplan: Planning in the robot operating system. In *Proceedings of the international conference on automated planning and scheduling*, Vol. 25. 333–341.

[3] Jiankang Deng, Jia Guo, Yuxiang Zhou, Jinke Yu, Irene Kotsia, and Stefanos Zafeiriou. 2019. RetinaFace: Single-stage Dense Face Localisation in the Wild. https://doi.org/10.48550/ARXIV.1905.00641

[4] Mary Ellen Foster, Patricia Candelaria, Lauren J. Dwyer, Summer Hudson, Alan Lindsay, Fareha Nishat, Mykelle Pacquing, Ronald P. A. Petrick, Andrés Alberto Ramírez-Duque, Jennifer Stinson, Frauke Zeller, and Samina Ali. 2023. Co-Design of a Social Robot for Distraction in the Paediatric Emergency Department. In *Companion of the 2023 ACM/IEEE International Conference on Human-Robot Interaction* (Stockholm, Sweden) *(HRI '23)*. Association for Computing Machinery, New York, NY, USA, 461–465. https://doi.org/10.1145/3568294.3580127

[5] Xiaojie Guo, Siyuan Li, Jinke Yu, Jiawan Zhang, Jiayi Ma, Lin Ma, Wei Liu, and Haibin Ling. 2019. PFLD: A Practical Facial Landmark Detector. https://doi.org/10.48550/ARXIV.1902.10859

[6] Alan Lindsay, Andres Ramirez-Duque, Ronald P. A. Petrick, and Mary Ellen Foster. 2022. A Socially Assistive Robot using Automated Planning in a Paediatric Clinical Setting. arXiv:2210.09753 [cs.RO]

[7] Christian Muise, Sheila McIlraith, and Christopher Beck. 2012. Improved non-deterministic planning by exploiting state relevance. In *Proceedings of the International Conference on Automated Planning and Scheduling*. 172–180.

[8] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. 2009. ROS: an open-source Robot Operating System. In *ICRA Workshop on Open Source Software*.