Multi-Channel Expression of State Information in a Mobile Service Robot using Animated Lights*

Kim Baraka and **Manuela Veloso**
Carnegie Mellon University
kbaraka@andrew.cmu.edu, mmv@cs.cmu.edu

Abstract

Human-robot interfaces are increasingly using more than one modality to communicate information to humans. In this paper, we present a general method for enabling a robot to express information about its state on multiple communicative resources that we call expression channels. We propose a flexible mapping architecture from robot state to expressions which allows for modulating the latter as a function state of information. In particular, we focus on multiple expressive light strips to express different types of information about a mobile service robot's state, such as high-level, task-related versus low-level, navigation-related information. We implement our multi-channel state/expression mapping on CoBot, an autonomous mobile service robot, and illustrate our work using a diverse set of robot states.

1 Introduction

Mobile service robots travel long distances in human-populated environments [Veloso *et al.*, 2015], which results in different types of interactions between these robots and humans. For example, human-robot proxemics [Mumm and Mutlu, 2011] can greatly vary. In cases where humans are not in close proximity to the robot or when human presence fails to temporally coincide with robot expression, some robot-to-human communication mechanisms (e.g., speech, on-screen text) may fail. For this reason, Baraka *et al.* have proposed to use expressive lights, in addition to verbal methods, as a modality of expression of robot state that is both persistent and perceivable at a distance. More generally, mobile service robots would benefit from the established benefits of multimodal communication, such as complementarity and redundancy [Oviatt, 2012], when expressing useful information about their state.

Current robots increasingly make use of different channels for both user input and feedback, across modalities including visual, auditory, and tactile modalities [Gorostiza *et al.*, 2006], such as gestures, speech and affective (e.g, facial) expressions [Bennewitz *et al.*, 2007]. Multimodality in robots has also been shown to make human-robot interactions, in particular dialogue, more efficient [Yang *et al.*, 2007]. From a more practical point of view, channels used for communicating information, which we call *expressive channels*, constitute, regardless of the modality, a limited resource that can only express a restricted amount of information at a given point of time. Therefore, independent expression channels can be used to express different types of information simultaneously to humans. Information can be distributed in different ways, for example according to the content, such as affective content (expression of emotions) versus functional content (e.g., battery level) [Yang *et al.*, 2007], or according to the level of detail, as is done in this work where we categorize state information into high-level task-related information and low-level navigation-related information.

In this paper, we present a general formal method for mapping robot state to expressions on different expression channels. We illustrate our method using two channels of the same modality (namely expressive lights) but our method generalizes as well to other modalities. Throughout the paper, we illustrate the concepts presented with our implementation on the autonomous service robot CoBot, using multi-channel state expression lights as shown in Fig. 1. CoBot is capable of providing services to people in a building across multiple floors, such as *going to a location* in the building, *transporting an object* from location to location, or *escorting* a person to a location [Rosenthal *et al.*, 2010]. Each service is composed of a series of tasks of three types: 'navigate', in which the robot the robot navigates from location to location, 'ask', in which the robot actively asks for help from humans in order to overcome its limitations (e.g., pressing the button of an elevator), and 'wait', in which the robot is waiting for users to initiate the service or dismiss the robot.

In the rest of this paper, we start by introducing a representation of robot state suited for expression on a given expression channel. We then present our mapping architecture from robot state to expressions, with a focus on light animations. Finally, we show how the mapping method generalizes to handle multiple expression channels and illustrate this generalizability through multi-channel light expression on CoBot.

*This research was partially supported by the FCT INSIDE ERI grant, FLT grant number 2015-143894, NSF grant number IIS-1012733, and ONR grant N00014-09-1-1031. The views and conclusions contained in this document are those of the author only.

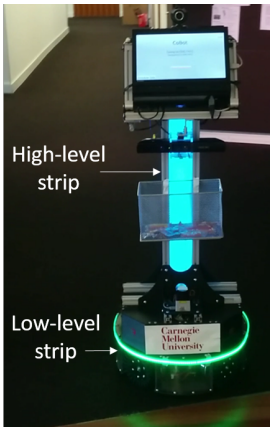

Figure 1: Setup of two light strips on CoBot expressing high-level (task-related) information and low-level (navigation-related) information, respectively

2 Robot State Representation for Expression

We start by introducing our representation of the service robot's state, which includes the following components: *robot variables*, *state features*, *expressible state tuples*, and finally *expressible classes*, each of which we discuss next.

2.1 Robot variables

Definition 1. *Robot variables* are quantities (discrete or continuous) that the robot maintains through its software. We categorize robot variables into *service* variables (related to the robot's services and tasks and the planning associated with them), *execution* variables (related to task execution in the environment), and *internal* variables (related to the internals of the robot's software and hardware).

The robot variables considered for CoBot are the following:

- *Service variables*:
  - the current service type service ('go-to-room', 'transport', or 'escort'),
  - the current task type task ('navigate', 'ask', or 'wait'),
  - the service path plan path-plan (list of position vertices (x, y, floor#)),
  - the service start location start-loc, and
  - the service goal location goal-loc.
- *Execution variables*:
  - the current robot location loc,
  - the current robot speed speed,
  - a boolean indicator path-blocked for whether the robot's path is blocked, causing the robot to stop,
  - the duration of the path blockage block-time (None if path-blocked is false),
  - a boolean indicator GUI-interrupt for whether the robot's task was interrupted from the GUI, and
  - the duration of the interruption from the GUI interrupt-time (None if GUI-interrupt is false).
- *Internal variables*:
  - the list sens-status of sensor statuses (1 for normal / 0 for faulty),
  - the list act-status of actuator statuses (same),
  - the software status soft-status (1 for normal / 0 for important error),
  - The gravity of the fault if it occurs fault-level (e.g., on a scale from 1 to 5),
  - a boolean indicator charging for whether the robot is charging, and
  - the percentage battery level batt-level

Note that the value of some of these variables might be undefined depending on the situation the robot is in; in that case, we assign a value of None to those variables with undefined values.

2.2 State features and robot state

Building upon robot variables, we generate state features, defined below.

Definition 2. Robot *state features* are discrete (usually high-level) aspects of the robot's state. They are represented as *logical expressions over robot variables*. The state features we consider are only those who are relevant to humans that potentially interact with the robot such as users, humans in the navigation environment, and developers.

The state features for CoBot considered in this paper are listed below:

- 'Escorting': (service = 'escort') $\wedge$ (task = 'navigate').
  The robot is in the process of escorting the user.
- 'Blocked by an obstacle': (path-blocked = True) $\wedge$ (task = navigate).
  An obstacle is impeding the navigation task progress.
- 'Asking for help at an elevator': (task='ask') $\wedge$ (isElevator(loc)) (where isElevator(a) returns True if location a is an elevator and False otherwise).
  The robot is waiting to get help at an elevator.
- 'Charging': (charging = True).
  The robot is connected to power and charging its battery.
- 'Interrupted by user': $\neg$ (task = None) $\wedge$ (GUI-interrupt = True).
  The robot has been interrupted by a user through the GUI.
- 'Waiting for object loading': (service = 'transport') $\wedge$ (task = 'ask') $\wedge$ (loc = start-loc).
  The robot is waiting for the object to be transported at the service start location.
- 'Stopped because of faulty component': (contains1(sens-status)) $\vee$ (contains1(act-status)) $\vee$ (contains1(soft-status)) (where contains1(a) returns True if list a contains at least one 1 and returns False otherwise).
  (An) execution-time fault(s) occurred in the robot software or hardware.
- 'Waiting for dismissal': (task = 'ask') $\wedge$ (loc = end-loc).
  The robot is waiting for the user to dismiss it by pressing its "Done" button on its touch screen.

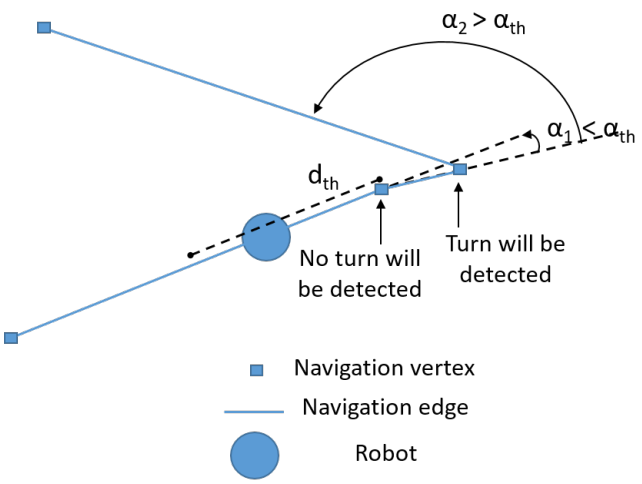

Figure 2: Turn detection check used for computation of feature 'Turning'

- 'Navigating': (task = navigate).
  The robot is performing a navigation task.
- 'Turning': (dist2NextVertex(loc,path-plan) $< d_{th}$) $\wedge$ (|nextTurnAngle(loc,path-plan)| $> \alpha_{th}$) (where dist2NextVert(.,.) returns the distance between the current location loc and the next vertex in the path-plan and nextTurnAngle(.,.) returns the upcoming turn angle α shown in Fig. 2; d_{th} and α_{th} are thresholds for these quantities).
  The robot is about to take a turn in its navigation path.

Note that state features are not necessarily mutually exclusive: more than one state feature could be true at the same time. This might pose a problem for the purpose of expression since we can only visualize one or at most a few of these features at a time on a given expression channel; this issue will be handled in section 4.

Definition 3. The robot *state* is defined as the set of all state features that are true at a given time t. Since the state of the robot is constantly changing as a function of time, the state is a dynamic process $S(t) = \{s_1, s_2, \ldots, s_{m_t}\}$ where $s_{1\ldots m_t}$ are the state features true at time t.

3 What Part of Robot State to Express?

We have looked so far at robot state dissociated from robot expression. Even though state features have been defined to be relevant to users (and hence would gain at being expressed to them in some way), we haven't taken yet into account the nature of the medium used for expression.

In this section, we are interested in two aspects of state representation for the purpose of expression:

- Modulation: State features are a discrete representations of high-level, user-relevant elements of the robot's state. However, this representation is rigid and doesn't allow for modulation (no possible expression of "microstates" within the state feature or of continuous quantities that might be relevant when a particular state feature is true). For this reason, in this section we introducing what we call *expressible state tuples*, a data structure which takes into account modulation of expressive behaviors (such as light animations) as a function of *modulating variables*.
- Simplification: For a given expression channel, there is a trade-off between *complexity* of the expression vocabulary (total number of different expressions used) and *legibility* of expression (how readable or intuitive these expressions are). For better legibility, it might be useful to group or cluster some expressible elements of states together into classes which are expressed using the same expression. For this reason, in this section we will introducing what we call "expressible classes".

3.1 Expressible state tuples

We introduce *expressible state tuples*, which are tuples containing all state information relevant to a particular robot situation and expressible on a given expression channel.

Definition 4. An *expressible state tuple* on a given expression channel is defined as a tuple $\langle s, \mathbf{v_s} \rangle$, where s is a state feature and $\mathbf{v_s}$ is a vector of *modulating variables*, relevant for expression of state feature s on the considered expression channel. These additional variables can either be robot variables (defined in section 2.1) or variables computed from robot variables, referred to as computed variables.

To illustrate the concept of an expressible state tuple, here are a few examples based on the state features listed in section 2.2:

- $<$ 'Escorting', percentDone(loc,path-plan) $>$, where percentDone(.,.) computes the percent distance traveled along the path plan (progress so far towards goal location); this expressible state tuple could be translated into some sort of progress indicator visible to the escorted user.
- $<$ 'Blocked by an obstacle', block-time $>$; this expressible state tuple could be translated into a blockage indicator which gets more noticeable as the blockage time increases.
- $<$ 'Charging', batt-level $>$; this expressible state tuple could be translated into a visual indicator changing as the percentage battery level increases.
- $<$ 'Turning', (nextTurn(loc,path-plan), speed) $>$ (where next-turn(.,.) is a function returning 'left' or `'right'` according to the direction of the upcoming navigation turn); this expressible state tuple can be translated into a turn indicator showing the upcoming turn direction.

3.2 Clustering expressible state tuples: expressible classes

For decreased complexity and increased legibility of expressions, we introduce in this section classes of expressible state tuples, or *expressible classes* for short. The clustering is dependent on the expression channel considered (different expression channels might require different levels of abstraction

depending on their complexity/legibility tradeoff). We cluster in the same class those expressible state tuples which possess semantic similarities, allowing us to express them in a similar fashion through the expressive channel. The expressible classes suggested below were designed for expressive lights as our expressive channel; for other expressive channels, as noted above, the classification may vary, even for the same expressible state tuples.

Based on the expressible state tuples listed in the previous subsection, we propose the following expressible classes:

- Class **'Progressing through a process with known goal'**: There are different kinds of processes that the robot goes through in which the goal is known. The progress on such processes could be expressed in the same way across different kinds of processes such as when the robot is escorting a person to a specific location or when it is charging its battery towards the maximum charge level. For both of the escorting and charging cases, the additional variable in the expressible state tuple represents a percentage completion.

  The expressible state tuples corresponding to this expressible class are:
  $<$ 'Escorting', percentDone(loc,path-plan) $>$
  $<$ 'Charging', batt-level $>$

- Class **'Interrupted during task execution'**: There are different ways in which the robot's task execution gets interrupted. From the perspective of expressive lights, it doesn't matter knowing why the interruption occurred (e.g., because of an obstacle, a faulty robot component, or a user-initiated interruption through the GUI) as much as communicating a state of interruption. (Other expressive channels such as screen display or voice could eventually complement the expression with additional information.)

  The expressible state tuples corresponding to this expressible class are:
  $<$ 'Blocked by an obstacle', block-time $>$
  $<$ 'Interrupted by user', interrupt-time $>$
  $<$ 'Stopped because of faulty component', fault-level $>$

- Class **'Waiting for human input'**: As described previously, there are several situations for which the robot is waiting for some sort of human input (corresponding to tasks of type 'wait' or 'ask'): the robot can be waiting for a task to be initiated by a user or confirmed at completion time, or it can be waiting for help from humans to overcome some of its limitations.

  The expressible state tuples corresponding to this expressible class are:
  $<$ 'Asking for help at an elevator', None $>$
  $<$ 'Waiting for object loading', None $>$
  $<$ 'Waiting for dismissal', None $>$

4 Robot State / Light Animation Mapping

Now that we have discussed our representation of the robot state both on its own and in relation to an expression channel, we look at the problem of mapping expressible classes to specifically light animations.

4.1 State/animation mapping architecture (single channel)

The mapping we define between expressible classes and light animations is divided into two parts:

Animation determination: A base animation with a fixed **animation pattern** (e.g., blinking, fade in/out) and associated default parameters are determined by the 'feature' part of expressible state tuple, which is by definition discrete and communicates well the presence of distinct robot states.

Animation modulation: We allow modulation in the animation by modifying the value of animation parameters such as **speed**, **color**, or **spatial parameters** based on the value(s) of the 'variable' part of the expressible state tuple.

Fig. 3 summarizes our mapping architecture.

4.2 Expression of non-exclusive state features

Our definition of state features imposes no constraints on their mutual exclusivity, hence there could be two or more features which are true at the same time, resulting in more than one expressible tuple competing for expression on the same channel. If we assume that ony a single expressible state tuple can be expressed on a single channel then we need a way of selecting one out of the set of expressible state tuples, which we call state preference function.

Definition 5. A *state preference function* for a given expressive channel is a function $\phi : \mathsf{P}(F) \rightarrow F$ where F is the set of state features and $\mathsf{P}(F)$ is the power set of F. ϕ selects one preferred state feature given a set of true state features.

In practice, there might be sets of mutually exclusive features, which can reduce the domain of ϕ. Also, it might be possible to create a strict total order on the state features, which greatly simplifies the representation of ϕ, as is shown in the example below, but it might generally not be the case.

Example of preference ordering on sample features:
Consider the three features 'Escorting'$= s_1$, 'Blocked by an obstacle'$= s_2$, and 'Asking for help at an elevator'$= s_3$.

If the robot is blocked by an obstacle (s_2 is true), then it should express it even if it currently escorting a person (s_1 is true). (s_2 and s_1 are mutually exclusive since the robot cannot be blocked by an obstacle while it is statically waiting at an elevator).
Similarly, if the robot is escorting a person (s_1 is true) but across multiple floors, it will encounter situations where it is asking for help at an elevator (s_3 is true) while performing the escort task. In such situation, we prefer s_3 over s_1 since the service cannot continue if the 'ask' task is not completed, which is hence more important.

Therefore we have the following total order: $s_2 > s_3 > s_1$, where $x > y$ indicates that x is preferred over y.

4.3 Mapping architecture for multiple light strips / expression channels

The mapping method introduced above for a single channel can easily be extended to handle multiple expression channels. Even though our analysis focuses on two channels

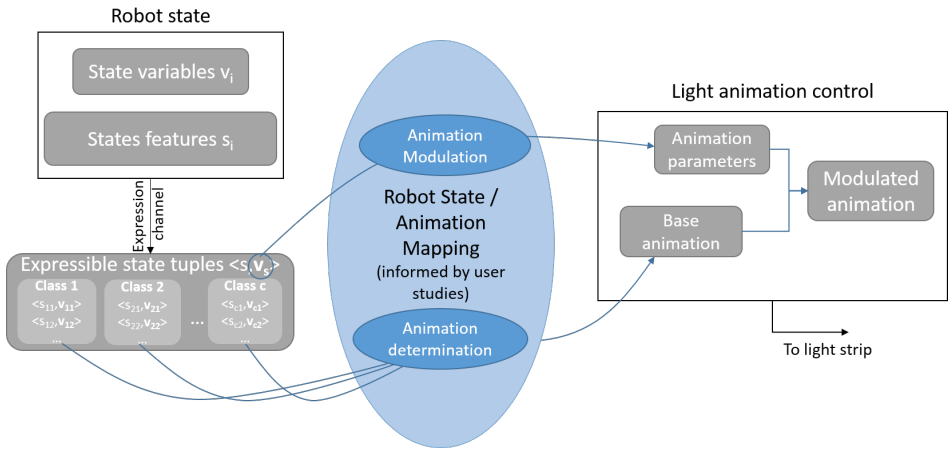

Figure 3: Architecture of the robot state / animation mapping for a single channel

of the same modality (expressive lights), the architecture presented is general enough to be applied to other modalities such as speech, expressive motion, etc.

Multi-channel mapping architecture
Fig. 4 shows our mapping architecture for more than one light strip, or more generally more than one expressive channel. Each channel has their relevant state features (possibly repeated), preference function, expressible state tuples and expressible classes.

Light strips for multi-level expression
To demonstrate the extensibility of our concept to more than one expression channel, we consider one light strip for higher-level state information (related to tasks and services) expression and another light strip for lower-level state information (related to navigation parameters). Implementation details on hardware mounting and animation control can be found in the next section. The high level strip animations were chosen from [Baraka *et al.*, 2016], and summarized below:

- 'Progressing through a process with known goal': for this class we used a bottom-up progress bar showing the completed distance traveled as a growing portion of the strip lit in bright green.
- 'Interrupted during task execution': for this class we used a fast red asymmetric fade in/fade out pattern on the whole strip.
- 'Waiting for human input': for this class we used a soft blue slow fade in/fade out pattern on the whole strip.

For the low-level strip, we consider a speed color indicator (red for fast, orange for medium speed and green for low speed) and a turn indicator animation, which makes either the left or the right half of the base light strip blink depending on the direction of the turn the robot is about to take. The rest of the strip also changes color as a function of speed. The corresponding expressible tuples for this strip's expression are: $<$ 'Turning', (next-turn(loc,path-plan), speed) $>$. A visual summary of the animations used for both strips (excluding the color change as a function of robot speed) is shown in Fig. 5.

4.4 Implementing the mapping on a mobile service robot

The robot state / light animation mapping was implemented on CoBot and has been robustly running for more than a year whenever the robot is deployed. Some details about the hardware and software components used to robustly integrate the light expression on CoBot are outlined below.

For our light sources, we used two programmable, fully addressable NeoPixel LED strips [1] with 91 and 144 pixels respectively, mounted on the robot's body and around its base respectively. Acrylic diffusers were added around the body LED strip to achieve omnidirectional visibility. The light strips are controlled by Arduino Uno microcontrollers (one per strip).

The light interface control architecture is summarized in Fig. 6. A Robot Operating System (ROS) node running on the robot itself keeps track of the robot variables (by subscribing to the corresponding ROS topics or calling the corresponding ROS services) and computes the corresponding expressible state tuples. It then maps these to animation parameters for each strip, which are sent to the corresponding microcontroller using serial communication. Based on the animation parameters received, the microcontroller controls the light strip by running an animation control algorithm. The flow of data from the ROS node to the microcontroller is synchronized with the animation episodes because serial communication is only reliable when the microcontroller is not sending any data out to the strip. We ensure such synchronization by a simple procedure similar to a handshake at the end of each animation episode. The actual data sent out to the LED strip on the digital pin of the Arduino is determined by the Adafruit NeoPixel library[2] and uses serial packets that the WS2811-based LEDs understand.

[1] https://www.adafruit.com/products/1507
[2] https://github.com/adafruit/Adafruit_NeoPixel

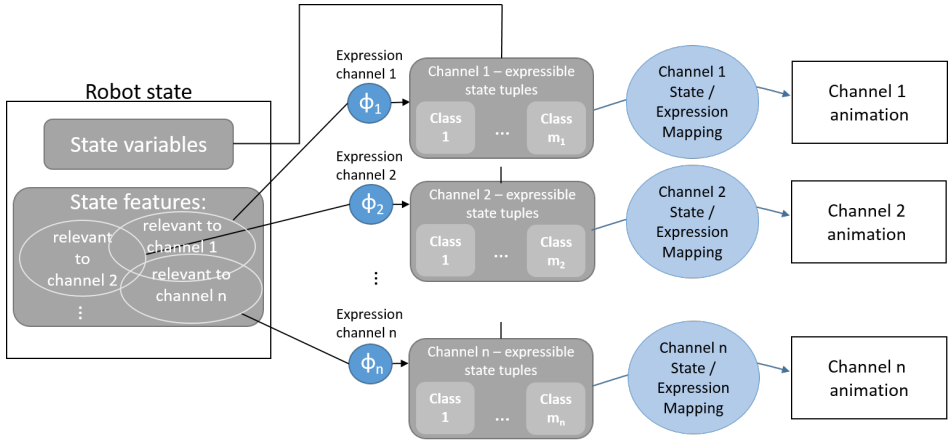

Figure 4: Mapping architecture for multiple expressive channels

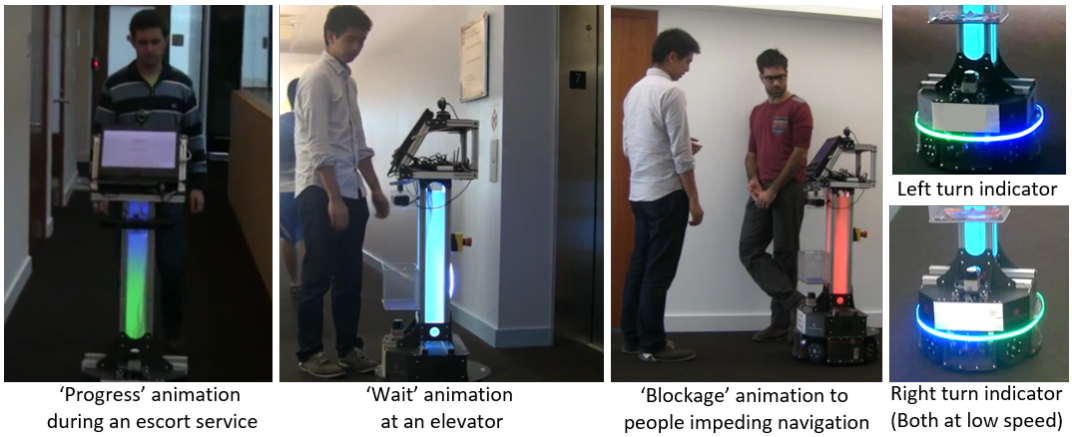

Figure 5: Summary of the animations used for the two light strips

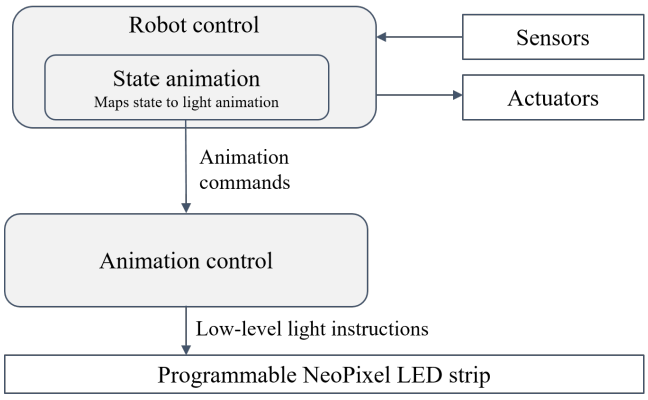

Figure 6: Control diagram of the state animation interface

5 Conclusion

We have presented a general method for mapping human-relevant state information of a mobile service robot to expression on expression channels. We first introduced a representation of robot state suitable for expression and show how we map the extracted state information to expressions on multiple independent channels. Our method was implemented on a mobile service robot, CoBot, using two light strips as its two independent expression channels revealing different aspects of robot state to humans, and complementing existing CoBot modalities such as speech and on-screen text.

This paper has assumed independent expressions channels. In reality, there could be a dependency between different expression channels, either because of the heterogeneity of modalities such as for instance speech and expressive and lights, which drastically differ in the level of abstractness of their expressions, or because these modalities require synchronization [Kim *et al.*, 2009]. In the future, we plan to extend the formalism presented in this paper to handle inter-channel dependency.

References

[Baraka *et al.*, 2016] K. Baraka, A. Paiva, and M. Veloso. Expressive lights for revealing mobile service robot state. In *Robot 2015: Second Iberian Robotics Conference*, pages 107–119. Springer, 2016.

[Bennewitz *et al.*, 2007] Maren Bennewitz, Felix Faber, Dominik Joho, and Sven Behnke. *Intuitive multimodal interaction with communication robot Fritz*. Citeseer, 2007.

[Gorostiza *et al.*, 2006] J. F. Gorostiza, R. Barber, A. M. Khamis, MMR Pacheco, R. Rivas, A. Corrales, E. Delgado, and M. A. Salichs. Multimodal human-robot interaction framework for a personal robot. In *Robot and Human Interactive Communication, 2006. ROMAN 2006. The 15th IEEE International Symposium on*, pages 39–44. IEEE, 2006.

[Kim *et al.*, 2009] W. H. Kim, J. W. Park, W. H. Lee, W. H. Kim, and M. J. Chung. Synchronized multimodal expression generation using editing toolkit for a human-friendly robot. In *Robotics and Biomimetics (ROBIO), 2009 IEEE International Conference on*, pages 706–710. IEEE, 2009.

[Mumm and Mutlu, 2011] J. Mumm and B. Mutlu. Human-robot proxemics: physical and psychological distancing in human-robot interaction. In *Proceedings of the 6th international conference on Human-robot interaction*, pages 331–338. ACM, 2011.

[Oviatt, 2012] S. Oviatt. Multimodal interfaces. In *The human-computer interaction handbook: Fundamentals, evolving technologies and emerging applications, 3rd Edition*, pages 405–430. Lawrence Erlbaum Assoc., Mahwah, NJ, 2012.

[Rosenthal *et al.*, 2010] S. Rosenthal, J. Biswas, and M. Veloso. An Effective Personal Mobile Robot Agent Through Symbiotic Human-Robot Interaction. In *Proceedings of AAMAS'10, the Ninth International Joint Conference on Autonomous Agents and Multi-Agent Systems*, Toronto, Canada, May 2010.

[Veloso *et al.*, 2015] M. Veloso, J. Biswas, B. Coltin, and S. Rosenthal. Cobots: robust symbiotic autonomous mobile service robots. In *Proceedings of the 24th International Conference on Artificial Intelligence*, pages 4423–4429. Citeseer, 2015.

[Yang *et al.*, 2007] H. S. Yang, Y. Seo, I. Jeong, and J. Lee. *Affective Communication Model with Multimodality for Humanoids*. INTECH Open Access Publisher, 2007.