Task Dialog Manager

Technical Overview

David Orn Johannesson
david@iiim.is

April 10, 2018

Contents

1 Introduction

The Task Dialogue Manager (TDM) is a part of a cognitive system for the CoCoMaps project, currently in development by the Icelandic Institute of Intelligent Machines(IIIM) and Cognitive Machine Labs (CMLabs). The TDM takes high level decisions based on current system status and input from the user.
This document depicts the design aspects of the TDM and technical notes for how the system is set up. It uses the following structure, Section 2 explains the structural overview of the system. Focusing on showing how it is meant to work and how it is working. Section goes into the code structure, explaining how the files are connected and how to get the method to working. Section 4 goes into the specific module MEx, meaning extraction.
If required all additional and more detailed explanations are in the appendix.

2 Structure Overview

The TDM is a part of a larger distributed robotic system. It's outside the scope of this report to describe the entire system as well as the underlying messaging board *Psyclone*. The major modules in the system are ROS and movement, Vision, Audio input output.
The ROS and movement system is a ROS based structure using built in methods of generating a local map, creating a path and executing it. For further understanding read ROS documentation for ROS, distribution Indigo Igloo.
The Vision has various applications, the most important one is the facial detector. This is written in c++ usign the OpenCV framework and built in Haar-cascade methods.
Audio is run via built in linux devices and pushed out to a Jabra speaker. The audio to text and text to audio is done via Nuance and is has custom built code to use the psyclone framework.
The above systems all use the Psyclone structure to communicate, senda and recieve messages.
The TDM connects to the psyclone waiting for input messages of from the Nuance, using strings to connect words with concepts and decide what the best course of action would be. The most basic structure for the TDM are the task definitions, currently stored in .json files. They describe the keywords connected to the concept and the ouput questions that are supposed to help the user figure out what the robot needs to know.
One major functionality of the TDM is the keyword search for the input sentence, this is done via a Meaning Extractor or MEx, see subsection 4 for specific detail. The MEx searches the input stack for a keyword connected to the current task. By connecting concepts/tasks to keywords the same word can have different meaning based on what the information the robot currently needs.

3 Code Structure

The directory contains the following directories

- MEx - Contains class file for the Meaning extractor.
- TDM_objects
  - tasks, Contains .json files
  - actionlib.py
  - algorithms.py
  - objects.py
  - Tasks.py
- TDM_psyclone.py

3.1 actionlib.py

The specific functions when running a task. Define where to jump in the task tree.
passive Cuts the system into a passive state
action_greet Control flow for greeting
action_get_objective Control flow for the first question after greeting, what to do now.
action_select_joke Joke function, ask robot to tell you joke.
action_knockknock Ask robot to tell you a knock knock joke
action_movement Control flow for movement of robot.
action_startgen Control flow for the first part of starting up a generator.
action_PanelA Control flow for the general application of interacting with a panel.
action_panel_pin Control flow for trying to get the pin input.
action_panel_navigate General flow for navigating panel.
action_demo2 Demo 2 specific control flow, starting method.
action_startgen_demo2 Question flow for starting generator in demo 2.
action_demo2_question Intermediate question for the demo 2.
action_PanelB Control flow for panel interaction, demo 2.

3.2 algorithms.py

Containters for the various algorithm approaches. The follwoing containers are available.
TDM_base Base class structure that all the other storages use.
TDM_AA Active action stack, container for maintaining hwat current action, if any, is active.
TDM_SS Speak stack, a stack that stores sentences to be output once the system will allow it. **TDM_AS** Action stack. A stack for all actions that are suppose to be started.

3.3 objects.py

Dialog An dialog control object, not currently used.
Action_Parent A parent class creating some values repeatedly used in code.
Move_object A container for wrapping infomration about movement.
Talk_object An object for storing infomration about the output sentence.
Panel_query An object for querying a panel
Screen_navigation_object An object for screen navigation of panel.
Pin_Query An object for the specific task of interacting with a pin number.
Word_Bag A touple that stores the input sentences and allows the MEX to search it.

3.4 Tasks.py

A first attemt of connecting the .json files to the actionlib.py by creating a set of containers/objects/classes for various values.

4 MEx

The Meaninig extractor uses the values defined in the task .json files, specifically keywords to create a set of connected words. The meanings currently have weights based on time, the time function is computed as

$$\beta = \frac{1.0}{1.0 - \exp\left(-(i+1)\right)} \tag{1}$$

Where i is the counter of the current sentence, counting backwards. I.e. given a set of sentences, the newest one being first. The older sentences have a higher i. Therefore as sentences increase the words have less meaning. This function converges to 1.0.

4.1 MEx Mathematics

MEx uses cumilative meaning association to estimate form a sentance, what in its knowledge database it can return.
The knowledge database is represented as possible meanings(objectives) and these objective have words and weights associated with them. MEx assumes it gets a whole sentance and tries fit all words to possible meaning. It then sorts a tuple of meanings and values based on the cumilative values. The user (tdm) then decides which templates to fill based on what information it has.
This can be expressed mathematically as follows:

Assume we have a database of meanings, $M = \{m_1, ..., m_m\}$, and words, (w). We define a function f such that it maps the connection between words and meanings to specific values $f(w_i, m_j) = \kappa_{i,j} \in [0, 1]$.

Given a sentance represented as a set of words, $W = \{w_1, w_2, ..., w_n\}$ MEx takes the set of words maps the word value to a meaning through the mapping function, then sums up the overall function values and returns with the cumulative measure of each meaning. This results in a vector representing the weight of each meaning.