Senior Project – Computer Science – 2014

Using Spacial Context and Clarification Questions to Interpret Ambiguous Natural Language Commands

Joseph Plaus
Advisor – Prof. Striegnitz

Background

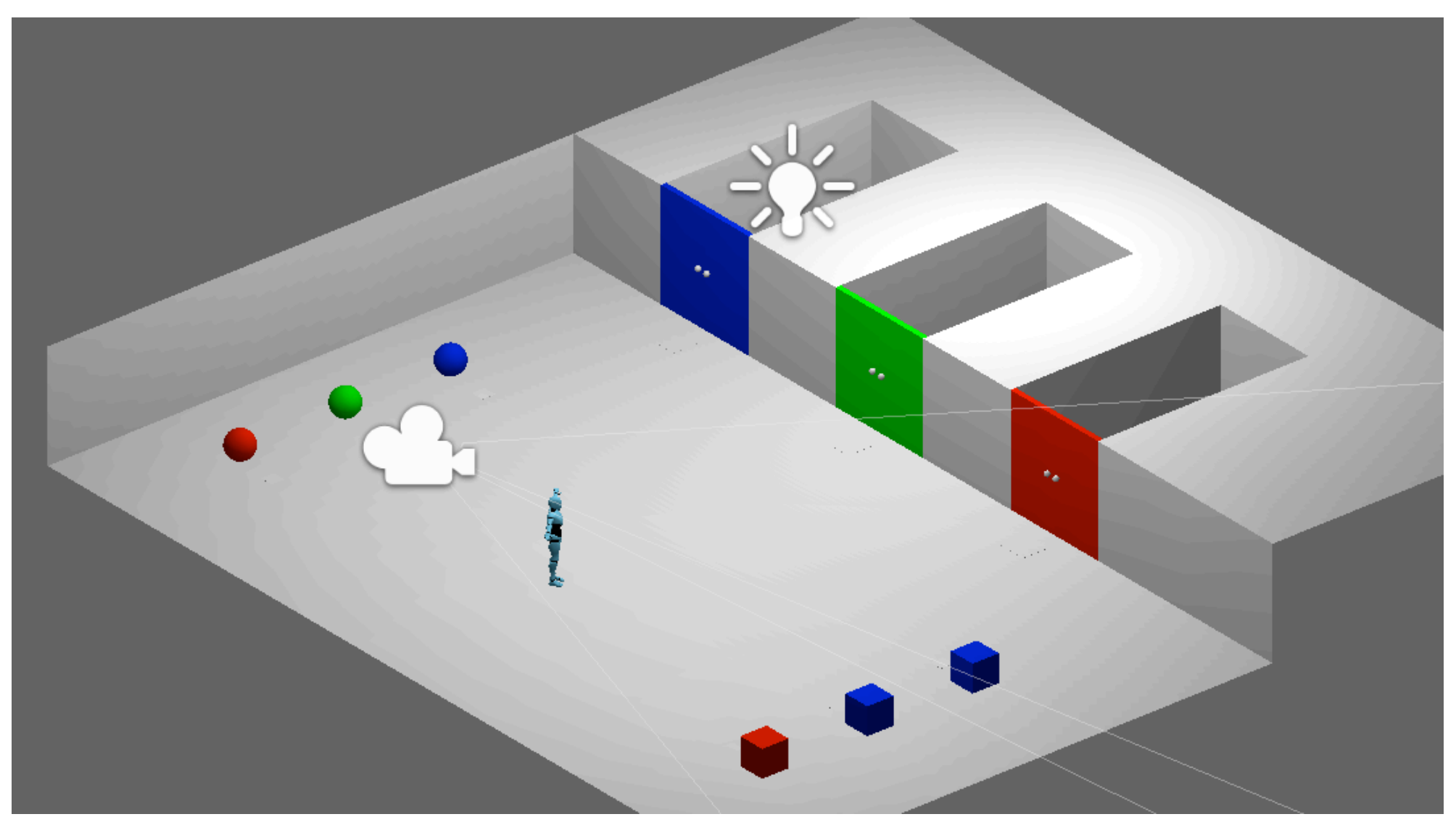
Natural Language Processing (NLP) is a complex and inherently difficult task to approach. The ability of a machine to understand and interpret natural human language is seemingly impossible as the inner-workings and thought processes of a computer are vastly different from that of a human brain. However, years of research and experimentation have made this concept increasingly feasible. In today's constantly evolving society, the need and potential usefulness of NLP-based systems is at an all-time high. The increased presence of robots and computer systems to automate processes that previously required human control allows for greater efficiency, but lacks operational ease due to required system-specific domain knowledge and language. The implementation of NLP capabilities in these systems could allow for increased operational diversity and ease by ideally allowing anyone with minimal system knowledge to control their functionality through natural language commands.

Overview

Natural Language Command systems are a type of NLP system in which a user provides commands in the form of natural language that are subsequently interpreted and executed. Many previous Natural Language Command systems have successfully implemented this process, but few employ the ability to handle ambiguous or vague commands. This ability is crucial in order to further the true understanding of natural language because it is an unnatural expectation that all commands will contain every piece of necessary information. This project attempts to address ambiguity by implementing a Natural Language Command system that employs various techniques to counter the possibility of failure due to confusion from such commands. Using a virtual 3D environment, a robot avatar can be given various navigational and basic operational commands. It executes multiple processes to semantically parse the commands into a form that it understands. If ambiguity exists, it utilizes both the knowledge of the surrounding environment and information given in previous commands to rank its executional options and choose the best course of action. If no decision can be made, it asks a clarification question of the user in order to obtain the missing information. This system has proven successful in its ability to identify if and when additional information is needed. The reusable construction of the environment allows for the system to be tested under constantly changing conditions, with which it has coped successfully. When compared with a similar system that lacks the ability to handle ambiguous situations, it successfully provides an alternative that accepts a more natural way of speaking.

Environment

The 3D virtual environment that this system employs was designed using the Blender 3D Animation Suite and later imported into the Unity Game Engine as the "room" in which the commands are executed. Below is a basic room layout:



The room is then populated with reusable, pre-fabricated objects. The color, location, and quantity of these objects are all variables that can easily be changed. The room is generally set up so that there are multiple instances of a given object in one location. This tactic intentionally sets up the robot for confusion due to the potential ambiguity of *which* instance of that certain object the user is referring. Additionally, the environment allows for basic operational commands such as moving objects or opening doors.

1. Parsing the Command

After a natural language command is received, it enters a process called **Pragmatic Parsing**. The objective of this operation is to break down the command into a more practical structure so that deriving a meaning will be easier. Written in Python, this program uses the Natural Language ToolKit to define context-free grammars for three types of commands: **navigational**, **operational**, and **response commands** (for when a clarification question is asked). The grammars were created to find not the syntactical role of each word in the command, but the practical role in terms of executing that command. The program then parses the input command using the appropriate grammar. An example can be seen below:

Input: *Walk towards the green ball.*

Traditional Syntactic Parsing:
[Input: [Verb: Walk] [PrepPhrase: [Prep: towards] [NounPhrase: [Determiner: the] [Adj: green] [N: ball]]]]

Pragmatic Parsing:
[NavInstr: [Action: walk] [Spacial: towards] [Destination: [Determiner: the] [Type: ball] [Color: green]]]

Utilizing a pragmatic approach is clearly more useful for this system. The tags associated with each part of the command make more sense in the scope of its final objective.

2. Interpreting the Parsed Command

After the command is parsed into a pragmatic format, the system passes it into Unity where various scripts written in C# derive its meaning. The following are the main components of this step:

1. **Command object**: a 'command' script defines and creates an object for each command that the system receives.

2. **Command Manager**: this part of the system stores all of the received commands and is updated each time a new command object is created. If needed, this allows previous commands to be accessed to determine the meaning of the current command.

3. **Command Processor:** this is the component that actually derives the meaning of the parsed command string. It initially decides how to approach each command based on its type (navigational, operational, or response). Basic commands like "*walk forwards*" or *"turn left"* are fairly easy to interpret, but more complex commands that reference an object or destination are much more difficult.

- **Explicit commands** occur when the destination or object in question is either the only one of its type in the entire environment, or the only one of its type that is visible to the user. If the command has an explicit meaning, no further information is needed.
- **Ambiguous commands** occur when the object or destination in question occurs more than once in the user's field of view. If there is more than one "visible target", the system enters a process to determine the **salience** of each target. In other words, it ranks the visible targets based on the likeliness that they are the intended target. This process takes into account the proximity of each target to the robot, and the distance of each target from the center of the user's field of view (i.e. is the target directly in front of the robot or off to the side?). If the visible targets are all ranked equally, the system asks the user to specify which one. This creates a **Response command** that adds on to the information provided in the previous command. Some examples of Response commands are "*the left one*" or "*the green one*".

3. Executing the Command

As soon as the meaning of the command is determined, the appropriate functions within the 'robot' script are called. This script contains basic movement functions as well as movement functions that take the destination as a parameter. It also has operational functions that trigger animations of the robot either picking up and putting down a moveable object, or opening a door. All objects within the environment have their own scripts that define both appearance and behavior.

Image Sources: "U logo" : http://content.sportslogos.net/logos/35/5074/full/6187_union__dutchmen_-primary-0.png
"Union CS" : http://cs.union.edu/media/images/CS_LOGO.png