Sliding Puzzle Using Robotic Arm Controlled by *Alexa* *

Luoyuan Xiong †
Electrical and Electronic Engineering
Nanyang Technological University
LXIONG002@e.ntu.edu.sg

ABSTRACT

This is an interim report on one independently developed component of a larger composite puzzle, designed under the project title *Escape Room using Electromedical or Mobile Computing Technologies*.

The rapid advancement in machine learning algorithm in recent years has enabled applications like human speech recognition, natural language processing etc. On top of that, cloud computing has come to prevalence, which shifts the ground for traditional product development. Together with the trending notion of "Internet of Things" and smart home, we decide to make use of state-of-the-art technologies to showcase a different paradigm for human machine interaction while demonstrating lightning fast, low cost, modern software development.

Specifically, we embody our design into a physical sliding puzzle board played by a robotic arm which is controlled by player's vocal commands.

Keywords

microcontroller; voice recognition; cloud computing

1. INTRODUCTION

System on Chip (SoC), utilizing a integrated circuit that incorporates a microcontroller with other electronic components, has been popularized by the growing ecosystem developed around *Single-Chip Microcomputers*. These microcomputers are now widely used in many embedded system, from audio player, to smart fridge, to automobiles. They essentially combines a microcontroller with memory space and I/O ports to enable programmable digital control over physical devices.

Modern microcomputers, like *Raspberry Pi*[1] and *Arduino*[2], normally comes with a hardware (i.e. an extendable SoC) and a software package (i.e. an IDE). Their modularity and extensibility, plus most importantly open-sourced design drive hackers to build thousands of applications.

Meanwhile the wave of cloud computing has penetrated into our daily life. Products, especially many mobile applications, take the advantage of the robustness, excellent uptime and scalability properties of cloud services(e.g. AWS, Azure). Instead of investing millions in building an infrastructure from scratch to handle all the traffic, medium to small companies and startups could just outsource their computation, storage and request handling on those cloud platforms. Those cloud services are robust because they mostly utilize duplicated state machines(e.g. Zookeeper) and distributed computing; they are economically and technically scalable because of their pay-per-usage pricing that's commensurate with the demand and fast seamless scaling by utilizing idle resources and pre-configured machines provided by those tech giants.

Finally the speech recognition technology powered by digital signal processing and accelerated by deep learning techniques. Starting from Markov Model and statistical approach to Recurrent Neural Network[4], computers are equipped with amazing text-to-speech, speech-to-text capabilities, thus bringing a new user-friendly interface for human machine interactions.

The remainder of the report is organized as follows. We begin by introducing the background information and basic knowledge about all the main technologies or tools being used. Section 3 describes the architectural design of the entire system, including interfaces between subcomponents, a walk-through of a new feature development cycle and an interaction flow from player's perspective. Section 4 discusses the current progress of the project as well as technical challenges encountered along the way. Section 5 lays out the future roadmap. We conclude with a summary of our lesson learned in designing and developing innovative projects.

2. BACKGROUND

There are three major components we use in the project, designed and manufactured by their corresponding companies, which requires a brief introductory description—the microcomputer *Arduino* and its auxiliary Motor Shield and Ethernet Shield; the robotic arm; and the voice assistant *Alexa* plus necessary cloud services.

*This serves as an interim report on one of the components of a bigger design for *EE3080 Design and Innovation Project*. with a topic called *"Escape Room using Electromedical or Mobile Computing Technologies"*

†Joint work with Le Phuong Duy from Group 2

[1]https://www.raspberrypi.org/
[2]https://www.arduino.cc/

Special thanks to School of EEE for funding the project.

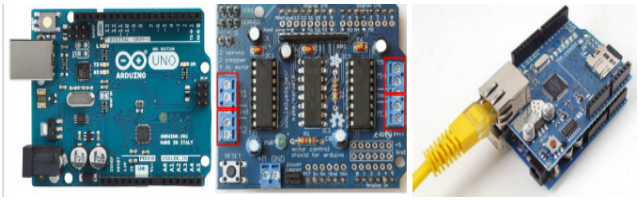

Figure 1: From left to right, Arduino Uno, AdaFruit Motor Shield, Ethernet Shield

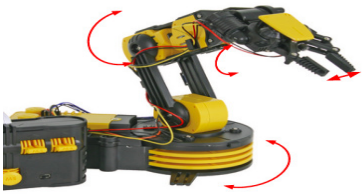

Figure 2: VELLEMAN KSR10 Robotic Arm (4 motors)

2.1 *Arduino Uno* and Shields

Arduino Uno (Figure 1) uses *ATmega328P*[3] microcontroller with 14 digital I/O pins, 6 analog input pins and 32 KB flash memory(0.5 KB of which used by bootloader). The bootloader simplifies the process of loading program from Arduino IDE editors to on-chip flash memory through serial connection[3]. With a USB-to-Serial adaptor chips, programming Arduino is as easy as uploading your code to Arduino via USB cable.

To provide richer set of functionalities for *Arduino* PCB, a wide spectrum of pluggable shields could be used. By stacking on top of the *Arduino* board with pin headers soldered, thereby connecting all necessary pin ports and extending the capability.

Concretely speaking, there are two shields required in our project: a motor shield to drive different motors embedded in the robotic arm, and a ethernet shield to connect to the internet(Figure 1).

2.2 *VELLEMAN KSR10* Robotic Arm

KSR10 robotic arm[4](Figure 2) is one of the most affordable, streamlined and hackable mechanical arms in the market.

Out of the box, it has four motors taking charge of rotational motions on different coordinate planes. The product was designed to be controlled by a game-pad handle, but those input wires could be directly connected to any microcontroller, which enables the integration with Arduino board possible.

2.3 *Alexa* and Amazon Web Service

Alexa is a voice assistant developed by Amazon. At its core, *Alexa Voice Service(AVS)*[1] is a module providing powerful speech recognition functionality with well-defined API for easy integration. It was trained through statistical modeling and neural-net deep learning and supports multiple languages.

[3]belongs to AVR family manufactured by Atml Corporation
[4]https://www.velleman.eu/products/view/?country=es&lang=en&id=375310

Figure 3: Amazon Alexa Voice Assistant

The consumer facing devices, such as Amazon Echo, Echo Dot(Figure 3), are the "Alexas" that most people refer to. Fundamentally they are just beautiful-looking speakers with internet connections — by sending signal captured to *AVS* and outputting sound according to the signal composed by *AVS*.[5]

Each *Alexa Skill* instantiates particular service the voice assistant could offer. A skill encapsulates a piece of internal program logic and intended interaction flow. For example, a smart home skill might turn off the lights when receiving a request from *AVS*; or a "flash briefing" skill may update you on latest national news when you ask for it.

Any verbal requests will be firstly translated into text and formatted into HTTP request by *Alexa Voice Service*, then matched up with certain *invocation name* which uniquely represents one particular skill, henceforth "wake up" the program responsible for this request. Upon that, it either launches a new session for further interaction or continues the latest saved conversation. Any request will only be valid if it contains words or phrases that belong to a list of predefined *utterances*, each of which corresponds to an *intent*. An *intent* describes a functional unit of the skill program, for instance, a *ResumeIntent* may start playing the music from where it dropped off; or a *HelpIntent* may end up having the voice assistant read out a piece of explanatory note.

All of the settings and code for *Alexa Skill* need to reside on a highly accessible cloud and hooked up with the physical *Alexa Echo* device. The recommended hosting service is *Amazon Lambda*[2], which works as the server responding to all incoming requests from the users based on the programs or instructions you uploaded. It significantly simplifies the process from development to production as all *Lambda* servers save developers from provisioning or managing the server themselves. Furthermore, for skills that require saving intermediate states, *Amazon S3*[6] cloud storage and a NoSQL-style *Amazon DynamoDB*[7] have super convenient native supports.

3. SYSTEM ARCHITECTURE

Adhering to common wisdom in software engineering, the architectural design of the system embodies two major principles: *Separation of Concerns* and *Agile Development*.

The overall construction consists of three functional components: robotics controller unit, sliding puzzle program and voice assistant unit (Figure 4). Each component enforces high modularity and abstraction, which boils down to simplified interactions between two modules with clearly defined

[5]over-credited device that does the least but received all the attention, which reflects the natural experience nowadays these product provide.
[6]https://aws.amazon.com/s3/
[7]https://aws.amazon.com/s3/

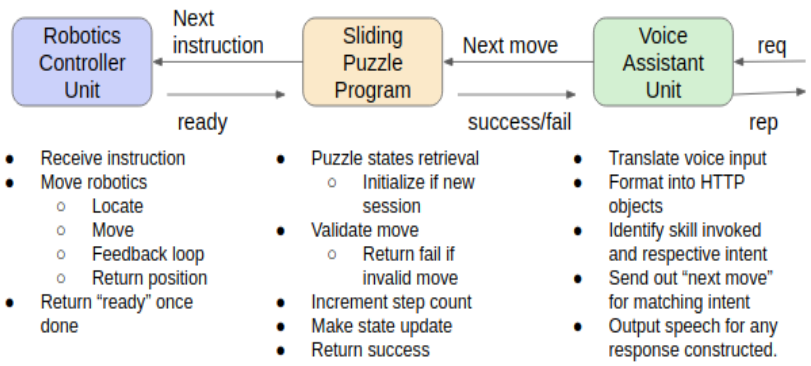

Figure 4: System Architecture

APIs, contained propagation of effects for easier debugging and scalability for future extension.

Robotics controller unit is responsible for instructing the mechanical arm through motor drivers and potentiometers for feedback loop. In our implementation, an Arduino Uno board, a AdaFruit Motor Shield and an Ethernet Shield soldered together are deployed to support all functionalities. More preciously speaking, the Ethernet shield serves as the interface between physical stacks and (virtual) puzzle stacks (i.e. sliding puzzle program). It interprets the HTTP requests sent across internet and translate them to specific function calls for Arudino Uno which directly delegate motor drivers on Motor Shield to rotate, thus move the arm to the intended position. Apparently, robotic controller unit doesn't need to know anything about the puzzle logic, or whether the instruction is valid, or the current state of the puzzle (e.g. successfully solved or still in progress), *separation of concern* principle applies here in a sense that this module is oblivious of anything other than the definitive next instruction to be executed.

Sliding puzzle program is the piece of logic that maintains and updates the state of the game. Intuitively, it is the gatekeeper of entire game as it decides whether a move is legitimate and whether the puzzle has been solved. From our system design point of view, this program acts as a relayer who receives formatted request from *voice assistant unit* and processes it after which forwarded to *robotics controller unit* to execute the move. This program plus game states of players are stored in *Amazon DynamoDB* and are linked with the voice assistant unit.

Voice assistant unit manages all verbal requests from actual players, categorizes the request, passing to *sliding puzzle program* and responds to players based on result status. In reality, it is accomplished through the inter-play of *AVS* and *Alexa Skill*.

To re-emphasize the modular design, bugs or errors in any one of above three units will not propagate to its adjacent module, thus making the entire development and maintenance experience more effective and light-weighted.

3.1 Player Interaction Flow

Supplementary to the high-level overview of the system architecture, this section describes a complete interaction flow (see Figure 5) in chronological order.

Firstly it starts with a player invoking *Alexa Echo Dot* by saying its *invocation name* (e.g. "Sliding Puzzle") and specific command (e.g. " move right").

Once capture the analog signal, the Echo dot device will convert it into digital signal and pass on to *Alexa Voice Service* for processing.

Then the speech recognition algorithm translates the signal into text and recognize the particular skill to be invoked, henceforth direct the command together with some metadata to *Alex Skill*.

Upon receiving the command text, the *Alex Skill*, residing on *Amazon Lambda*, matches it with "ActionIntent" and further constructs a HTTP request based on the intent after which sends it across the internet to the end point (i.e. Ethernet Shield).

The Ethernet shield quickly strips off unrelated metadata and extract out the exact command which ultimately corresponds to a functional calls on *Arduino UNO*.

Arduino UNO then could instruct the motor shield to mobilize the robotic arm in a pre-defined position and execute hard-coded actions.

After the robotic arm completes the task and returns to its original position, a "success" message will be transmit back all the way to *Alexa Skill* who will ask Echo speaker to notify the player about the status of the execution or asking for the next move instruction, subsequently completes one interaction flow.

3.2 New Feature Development Cycle

To briefly illustrate the development cycle for a future new feature in our system, one might want to consider *bottom-up approach*, which means starting from the lowest execution level with the least logical complexity — in our context, the motions of the robotic arm. Actions and functions in Arudino UNO has a one-to-one relationship. After figuring out the new class of motions for robotic arm to perform, we move up along the development stack to *sliding puzzle program* where combination of movement might be specified contingent on a new intent to-be-added. The final step is defining *utterances samples* for this newly designed intent by imagining how would a normal user express his/her intention of calling this function.

Unit testing is always recommended for a modular system. In the case of *Amazon Alexa Skill*, a command line tool

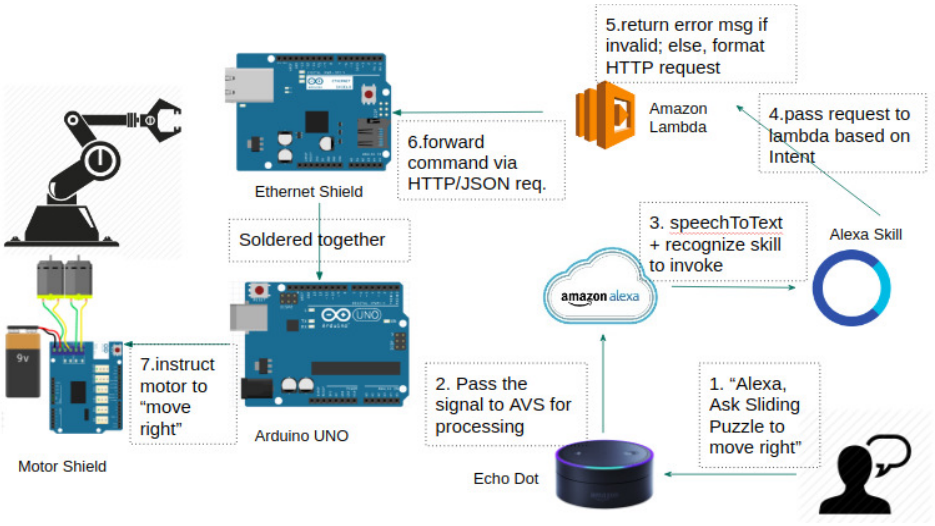

Figure 5: player interaction flow

Table 1: Project Progress

Task	Type	Status
Purchase components	logistic	80%
Assemble robotic arm	hardware	100%
Soldering 3 boards	hardware	70%
Physical Puzzle and Track	hardware	10%
Motion Program (Arduino)	software	20%
Ethernet connection	software	0%
Utterances, Intent Specification	software	100%
Skill logic	software	70%

named *ASK*[8] is super handy and for better visual effect and user interface, a test simulator(beta)[9] is now available for Amazon developers.

4. CURRENT PROGRESS

Please see Table 1 for the current progress of this project. During the interim presentation, we presented two short video clips of *Alexa skill*[10] and rotating *robotic arm*[11].

Code snippet for alexa skill as an example is given in Appendix A.

4.1 Challenges

Our developing process is agile yet still bumpy. There are many iterations on what the final product will look like, or how could we achieve certain functionalities and most troublesome part is the gap between mentally depicted solution and cold hard reality.

Here are a list of operational and technical challenges encountered:

- Poor project management results in a stretching purchase period, with necessary electronic components still in shipping 3 weeks in.
- Inaccurate motor drivers causes an awkward situation where robotic couldn't return to the "original position" perfectly without feedback loop through potentiometer, which adds complexity.
- Moving physical puzzle piece is non-trivial. Drop and Pick method requires high accuracy which is difficult for the lower-end robotic arm we had; whereas Push and Pull method requires a track to which all pieces attach.

5. FUTURE ROADMAP

- Solder Ethernet Shield with Arduino once it arrives.
- Finish Sliding Puzzle Program with unit testing.
- Finish Robotic Arm Motion Function with feedback loop.
- Overall testing and stress testing.

6. CONCLUSION

Despite the sophistication of the architectural design of the system and the large number of components or services required, the development process is quite smooth and fast with the help of well-established development frameworks and guidelines.

Readability, maintainability and scalability are more importantly than ever. With the assistant of "Infrastructure as a Service" from Amazon Cloud Platform, which frees us from struggling through administrative boilerplate during deployment, and focus on the product itself.

It is delightful to integrate cloud computing and latest speech recognition technology into one single project, and even though the managerial skills are yet to be sharpened, the project is on the right track with reasonable pace of progress.

[8]https://github.com/aws/aws-cli
[9]https://developer.amazon.com/blogs/alexa/post/577069bd-d9f9-439a-b4bf-3b0495e3d24b/announcing-new-test-simulator-beta-for-alexa-skills
[10]https://drive.google.com/open?id=1JaKZuQhPw_HKcjmBshrK1fJbhOLizJ-S
[11]https://drive.google.com/open?id=1leIj_kYexSMz1nejvq3bZymZx9FxL9i2

7. ACKNOWLEDGMENTS

The author would like to thank Le Phuong Duy for collaborated work on this project, Professor Ji-Jon Sit for his tremendous support, and all other teammates for dedicated discussion and helpful feedback. Finally, big appreciation for School Of Electrical and Electronic Engineering for its generous funding to make this implementation even possible.

8. REFERENCES

[1] Amazon Alexa developer document. https://developer.amazon.com/documentation. Accessed: 2018-03-06.

[2] Amazon Lambda documentation. https://docs.aws.amazon.com/lambda/latest/dg/welcome.html. Accessed: 2018-03-06.

[3] ARDUINO UNO REV3 technical specification. https://store.arduino.cc/usa/arduino-uno-rev3. Accessed: 2018-03-06.

[4] A. Graves, A. Mohamed, and G. E. Hinton. Speech recognition with deep recurrent neural networks. *CoRR*, abs/1303.5778, 2013.

APPENDIX

A. ALEXA SKILL CODE SNIPPET

```
const Alexa = require('alexa-sdk');
const https = require('https');

exports.handler = function(event, context, callback) {
    const alexa = Alexa.handler(event, context, callback);
    alexa.appId = "amzn1.ask.skill.366897d9-f60f-4e9d-83cb-a731c5855e6f"; // APP_ID is your skill id which can be found in the Amazon developer console where you create the skill.
    alexa.registerHandlers(handlers);
    alexa.execute();
};

const handlers = {
  'NewSession': function() {
    console.log("in NewSession");
    // when you have a new session,
    // this is where you'll
    // optionally initialize

    // after initializing, continue on
    routeToIntent.call(this);
  },
  'LaunchRequest': function(){
    this.emit('InitializeIntent');
  },
  'InitializeIntent': function() {
    // emit response
    this.response.speak('Welcome to the final rounds, your mission should you choose to accept it. What you are looking at is not your typical sliding puzzle'+'every move is done through voice control'+
      ' There are four commands you could shout out:'+' move up'+' move down'+' move left'+' and move right'+' Now, enjoy the game!');
    this.emit(':responseReady');
  },
  'MoveIntent': function() {
    console.log("Move");
    // Handle https request here on every move intent
    // ...
    var slotValues = getSlotValues(this.event.request.intent.slots);

    this.emit(':tell',"Okay, moving "+slotValues['direction']['resolved']);
  },
  'SessionEndedRequest' : function() {
    console.log('Session ended with reason: ' + this.event.request.reason);
  },
  'AMAZON.StopIntent' : function() {
    this.response.speak('See you, will miss you!');
    this.emit(':responseReady');
  },
  'AMAZON.HelpIntent' : function() {
    this.response.speak("");
    this.emit(':responseReady');
  },
  'AMAZON.CancelIntent' : function() {
    this.response.speak('Bye');
    this.emit(':responseReady');
  }
};

// ***********************************
// ** Route to Intent
// ***********************************

// after doing the logic in new session,
// route to the proper intent

function routeToIntent() {
  switch (this.event.request.type) {
    case 'IntentRequest':
      this.emit(this.event.request.intent.name);
      break;
    case 'LaunchRequest':
      this.emit('LaunchRequest');
      break;
    default:
      this.emit('LaunchRequest');
  }
}

//COOKBOOK HELPER FUNCTIONS

function getSlotValues(filledSlots) {
  //given event.request.intent.slots, a slots values object so you have
  //what synonym the person said - .synonym
  //what that resolved to - .resolved
  //and if it's a word that is in your slot values - .isValidated

  return slotValues;
}
```