Towards a Multimodal Ouija Board for Aircraft Carrier Deck Operations

by

Kojo Acquah

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Masters of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2015

ARCHIVES
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
AUG 20 2015
LIBRARIES

© Massachusetts Institute of Technology 2015. All rights reserved.

Signature redacted

Author .....
Department of Electrical Engineering and Computer Science
May 26, 2015

Certified by.. Signature redacted .....
Randall Davis
Professor
Thesis Supervisor

Signature redacted

Accepted by .....
Alber R. Meyer
Chairman, Masters of Engineering Thesis Committee

Towards a Multimodal Ouija Board for Aircraft Carrier Deck Operations

By

Kojo Acquah

Submitted to the Department of Electrical Engineering and Computer Science on

May 22, 2015

In partial fulfillment of the requirements for the Degree of Master of Science in Electrical Engineering and Computer Science

Abstract

In this thesis, I present DeckAssistant, a novel interface for planning aircraft carrier deck operations. DeckAssistant is an intelligent, multimodal aircraft carrier Ouija Board. DeckAssistant's design is inspired by traditional Ouija Boards, featuring a large digital tabletop display with a scale model of an aircraft carrier deck and aircraft. Users plan aircraft operations by using this interface to mock situations on deck. Unlike traditional Ouija Board technology, DeckAssistant is designed to contribute to the planning process. We enable DeckAssistant with a basic understanding of deck operations and add multimodal functionality to our system. Users manipulate our model of the deck by issuing commands using speech and gestures. The system responds to users with its own synthesized speech and graphics. The result is a conversation of sorts between users and the Ouija Board as they work together to accomplish tasks.

Thesis Supervisor: Randall Davis

Acknowledgements

I would like to thank my advisor, Professor Randall Davis, for his patience, insight, and incredible support during my thesis endeavor. He has continually helped me with the vision driving my thesis and his thought provoking questions and insights have greatly contributed to my understanding of systems design and multimodal interfaces.

I would like to thank Ying Yin, a former member of my research group, for her guidance and knowledge on gesture recognition systems. Ying Yin's initial work on hand and limb identification served as the basis for the hand tracking and pointing interpretation system used in DeckAssistant.

I would also like to thank Professor Missy Cummings and her former student Jason Ryan for their insight into deck operations and existing aircraft carrier technology.

Finally, I want to thank the rest of my research group, Jeremy Scott, Jason Tong, Jonathan Chien, and Katherine Hobbs for their insight and feedback.

This work was supported in part by the Office of Naval Research though contract ONR N00014-09-1-0626.

Contents

List of Figures

List of Tables

Chapter 1

1 Introduction

1.1 Overview

In this thesis, we present DeckAssistant, a digital version of an aircraft carrier Ouija Board for planning deck operations using multimodal interaction. Unlike traditional Ouija Boards, DeckAssistant supports multimodal interaction, i.e., it understands a range of speech and hand gestures from a deck handler and responds with its own speech, gestures and graphics. This style of interaction – which we call symmetric multimodal interaction – creates a conversation of sorts between a deck handler and the Ouija Board. DeckAssistant also has a basic understanding of deck operations and can aid deck handlers by carrying out common tasks.

Our system illustrates first steps toward a digital Ouija Board and, in doing so, suggests additional functionality available beyond traditional Ouija Boards. DeckAssistant, makes the following three main contributions to human-computer-interaction and intelligent user interfaces:

- We create the first, large-scale digital replica of an aircraft carrier Ouija Board for planning deck operations. This digital Ouija Board features a familiar interface of traditional Ouija Board, with its functionality centered around planning and maintaining aircraft placement on deck.
- We enhance our digital Ouija Board with a basic understanding of aircraft movement and arrangement on deck. We leverage this understanding through a set of coded actions to automate common tasks within our simulation. We build multimodal interaction around these actions, allowing the user to engage in conversation with the system to carry out tasks.

- We create a novel, graph-based conversation framework for scripting interactive conversations mixed with functionality in our system. We build these conversations into a dynamic action framework, which is used to carry out simple and complex tasks within our system.

1.2 Background and Motivation

1.2.1 Ouija Board History and Use

The deck of a modern aircraft carrier is often described as "organized chaos". Multiple personnel move about tending to a variety of tasks relating to high powered aircraft including maintenance, takeoff, landing, aircraft direction, and parking. To add to the complexity and potential hazards, high-powered aircraft move about in confined spaces in this high wind/noise environment. While seemingly chaotic, deck operations are highly organized and a hierarchy of personnel direct operations behind the scenes. On the deck, Aircraft Directors serve as "referees" for deck regions, directing all aircraft movement and placement around the regions they control. These directors receive their instructions from deck handlers, who plan all deck and surrounding flight operations from an elevated platform known as the Island (similar to the control tower at an airport). To organize deck operations, deck handlers work with a scale model representation of the flight deck known as an "Ouija Board" (see Figure 1-1). They move an assortment of aircraft models and other assets around to match their real-life deck counterparts and mark the models with pins and nuts to indicate individual aircraft status. Since Ouija Boards are accurate scale models, any layout of aircraft on the Ouija Board can exist on deck. The Ouija Board provides direct interaction for manipulating deck objects during planning as well as serving as an easily understood, though not necessarily up to date, status display of the deck,.

Figure 1-1: Aircraft Handler operating an Ouija Board (Source: Wikipedia).

Though seemingly primitive, Ouija Boards, first introduced on carriers during WWII, have numerous benefits. Planning "what if" scenarios is as easy as moving the pieces around, since if it fits on the board, it fits on the deck. Collaborating deck handlers can easily illustrate their plans to each other using the table. And the table is immune to technological or power failure. Even so, there is potential for technologically-enabled Ouija Board technology to further aid in planning deck environments.

1.2.2 Naval Push for Digital Information on Decks

Modern pushes for more technologically-advanced planning on decks are already present. As static displays, Ouija Boards offer no task automation or information processing to aid the deck handler in planning. "Spotters" constantly radio updates for aircraft, while deck handlers manually update the board. While creating potential plans, deck handlers manually move each aircraft and check a variety of conditions, such as aircraft placement and paths, to ensure plan validity.

To incorporate automation in deck planning, the Navy has started prototyping a new electronic Ouija Board as part of ADMACS (Aviation Data Management And Control System)

program. This system uses information from a carrier's Integrated Shipboard Information System (ISIS) network to display aircraft position and status on a conventional computer monitor. Deck handlers sit at a computer terminal and move ships around the deck using keyboard and mouse, instead of leaning over the Ouija Board. Currently being prototyped on the USS Abraham Lincoln, the Navy hopes to install systems on all carriers by end of 2015.

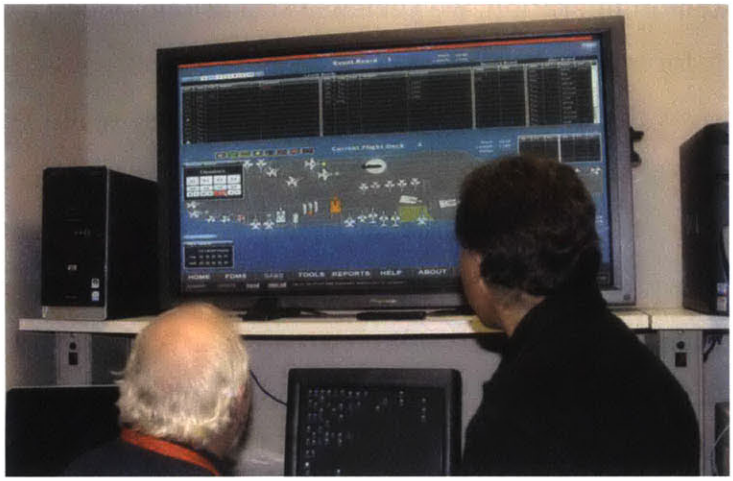


Figure 1-2: ADMACS Ouija Board prototype (Source: Google Images).

1.2.3 Possibilities of a Multimodal, Digital Ouija Board

Imagine if, instead of using a static tabletop with models or a single monitor display, deck handlers interact with a large digital display of the deck that could both illustrate and understand planning operations on deck. This digital Ouija Board can simulate a deck while updating aircraft movement and status from ADMACS sensor data. Deck handlers would still be able to interact with the board in a conventional way, i.e. move aircraft around deck by hand. But with modern technology, this system could utilize additional modes of interaction to aid deck planning. With speech and gesture recognition for instance, deck handlers can point, gesture, or speak to the Ouija Board to carry out tasks. The result of these actions could immediately play out in front of them with their consequences clearly illustrated (through simulation). This multimodal interaction can extend further to a two-way exchange, with the board that responds with visual and audio (speech) feedback to illustrate the consequences of actions. Even when limited to

predetermined speech inputs and outputs, the possibility with two-way speech, gesture, and graphical interaction could create a conversation between deck handler and the digital Ouija Board.

1.3 System Demonstration

To illustrate our system in operation, we review a scenario in which a deck handler wants to prepare an aircraft for launch on a catapult. The deck handler must come up with a new arrangement that moves the launching aircraft to the catapult while possibly clearing any additional blocking aircraft.

Our system consists of a large tabletop with a screen projected onto it from overhead (see in Figure 1-3). The display shows a digital version of an Ouija Board: a large graphical rendition of an aircraft carrier deck complete with aircraft (see Figure 1-4). Like a traditional Ouija Board, these aircraft are meant represent their arrangement on the actual deck.

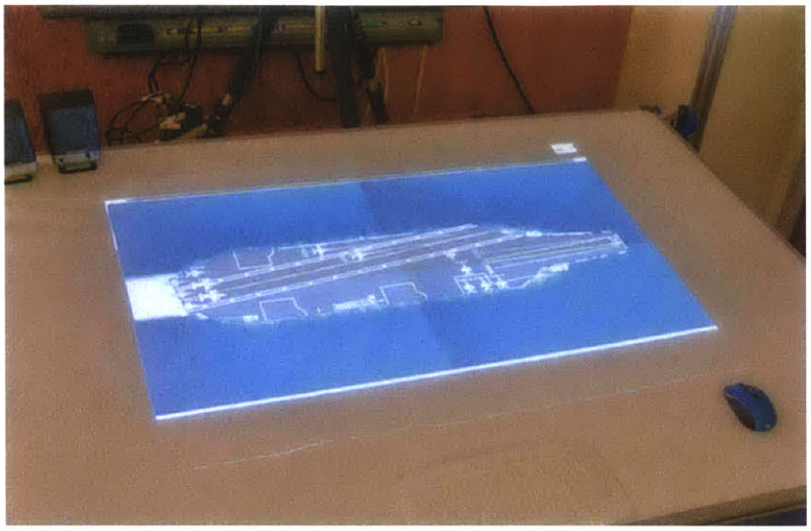


Figure 1-3(a): Large tabletop display of aircraft carrier Ouija Board, projected from overhead.

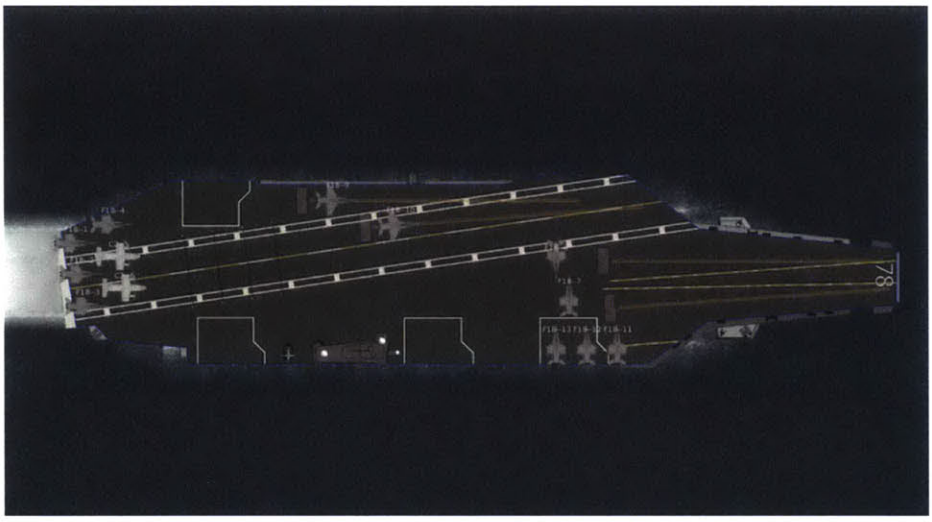


Figure 1-4: Digital rendition of scale aircraft carrier and aircraft. Tail numbers are shown above each aircraft.

To interact with the system, the deck handler stands in front of the tabletop and issues commands using a combination of speech and gesture (see Figure 1-5). The system uses a depth-sensing camera (mounted above the table) to track hands over table. A headset worn by the deck handler relays spoken speech commands to the system and synthesized speech responds to the deck handler.

Figure 1-5: deck handler issuing commands to the table with gestures.

To figure out how to move the aircraft for launch, the deck handler begins by viewing the initial configuration of the deck (shown in Figure 1-6): there are eleven F-18s (grey fighter jets) placed around deck and two C-2 aircraft (white carrier jets) at the rear of the deck. The deck handler's goal is to launch one of the C2 aircraft, which requires moving the aircraft from the rear to the one of the two open catapults at the front of the deck. There are four catapults on deck, labeled 1 to 4, beginning from the lower right catapult to the upper left catapult.

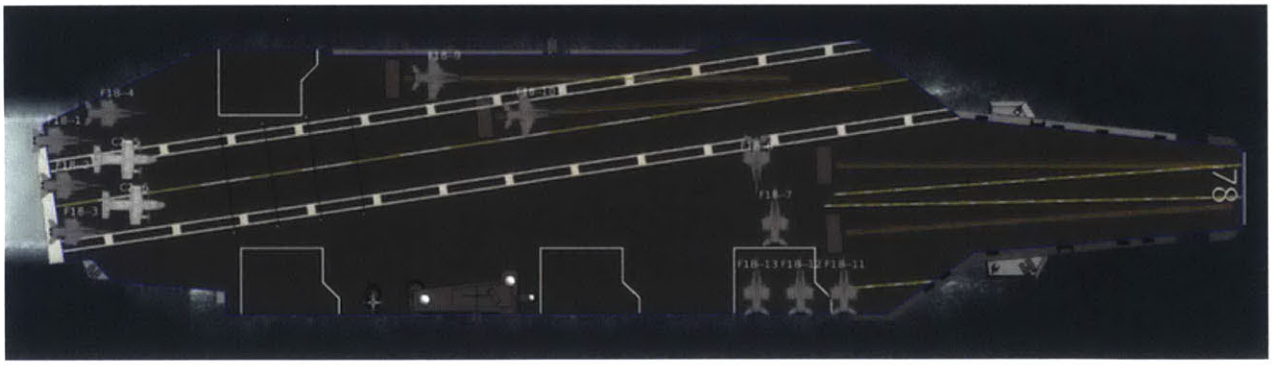


Figure 1-6: System showing the starting arrangement of the deck.

The deck handler points to the aircraft to move, in this case, the lower C2 aircraft. As shown in Figure 1-7, the system indicates which aircraft the deck handler has highlighted in green. While pointing, the deck handler initiates the move with the command: "*Move this C2 to launch on catapult 2*".

Figure 1-7: The system indicates where the user is pointing using an orange dot shown on screen. The selected aircraft is highlighted in green.

Our system is able to determine whether a command from the user can be carried out directly (in this case, simply moving an aircraft to launch), or whether additional actions are needed. The system checks for a clear path to Catapult 2 for the C2 and in this case recognizes that an aircraft must be moved out of the way to accommodate the launch. Using synthesized speech and graphics, the system informs the user of additional actions needed and provides the option to either accept or decline the changes (see Figure 1-8). In this case, the system plans to move a blocking F-18 out of the way (see Figure 1-9).

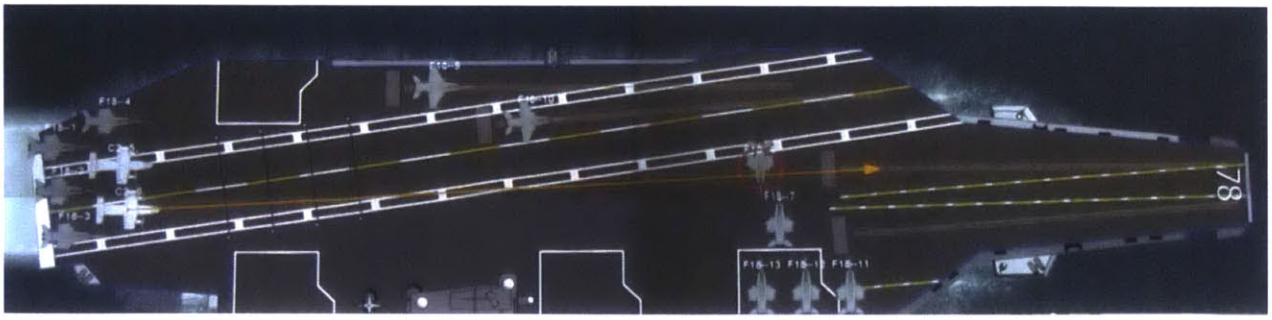


Figure 1-8: The system informs the user of an F-18 (circled in red) blocking the path to the catapult.

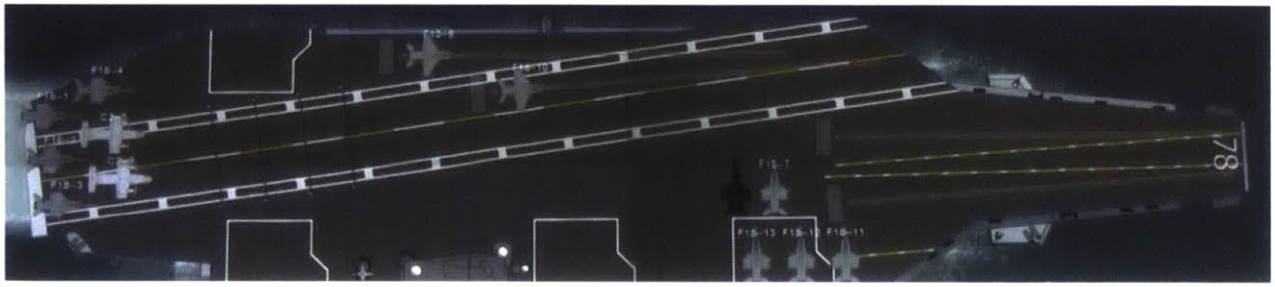


Figure 1-9: The system indicates an alternate placement for that aircraft (shown as an aircraft shadow).

If the deck handler accepts the modifications, the aircraft are moved in the simulation. If the deck handler declines, the deck is reverted to its original state before the command. The deck handler can issue different commands to achieve the desired result. For instance, after noting the need to clear a path the deck handler can specify an alternate re-arrangement of blocking aircraft, then move the C2 to launch.

With each additional command, the deck handler can update the deck and plan operations for aircraft movement. This two-way conversation allows the deck handler to focus on the higher-level goals of each operation, while the system helps implement the details.

1.4 Thesis Outline

In the next chapter, we present an overview of DeckAssistant and the multimodal commands used for interaction. We also explain the understanding of deck operations built into DeckAssistant. Chapter 3 gives an overview of the software and hardware powering DeckAssistant. This includes an overview of the software architecture, which explains the interactions of different components of DeckVewier detailed in later chapters. Chapters 4 through 7 detail the implementation and design of major components including the deck,

speech recognition and synthesis, gesture tracking, and the actions that control the system. Chapter 8 reviews related work. Chapter 9 offers conclusions from DeckAssistant in addition to potential future work.

Chapter 2

2 DeckAssistant Functionality

This chapter details the capability and interactions possible with our system. We explain DeckAssistant's understanding of the deck and give an overview of speech commands, gestures, and actions used to control our system.

2.1 Role/Function of DeckAssistant

We chose to focus our initial version of our digital Ouija Board as a planning aid for aircraft movement and placement on deck.

2.1.1 Actions in DeckAssistant

As a user operates our digital Ouija Board, they use a set of pre-determined actions to manipulate aircraft on deck. We call these actions Deck Actions. Each Deck Action centers on a specific task, such as moving an aircraft from one position to anther, or queuing an aircraft to launch on a catapult. Deck Actions include all logic to carry out their tasks, and are intended to be flexible and interactive (see Figure 2-1). As shown in the example demonstration in Chapter 1, a deck handler interested in launching an aircraft on a catapult gives a command, which creates the appropriate action in the simulation. The action is responsible for all the details of the getting the aircraft to its launch catapult. After surveying the deck, a Deck Action may engage the user for further input, suggest additional actions, or even suggest alternative actions if necessary. This allows a deck handler to focus on higher-level tasks, while using the system's understanding to aid with the details.

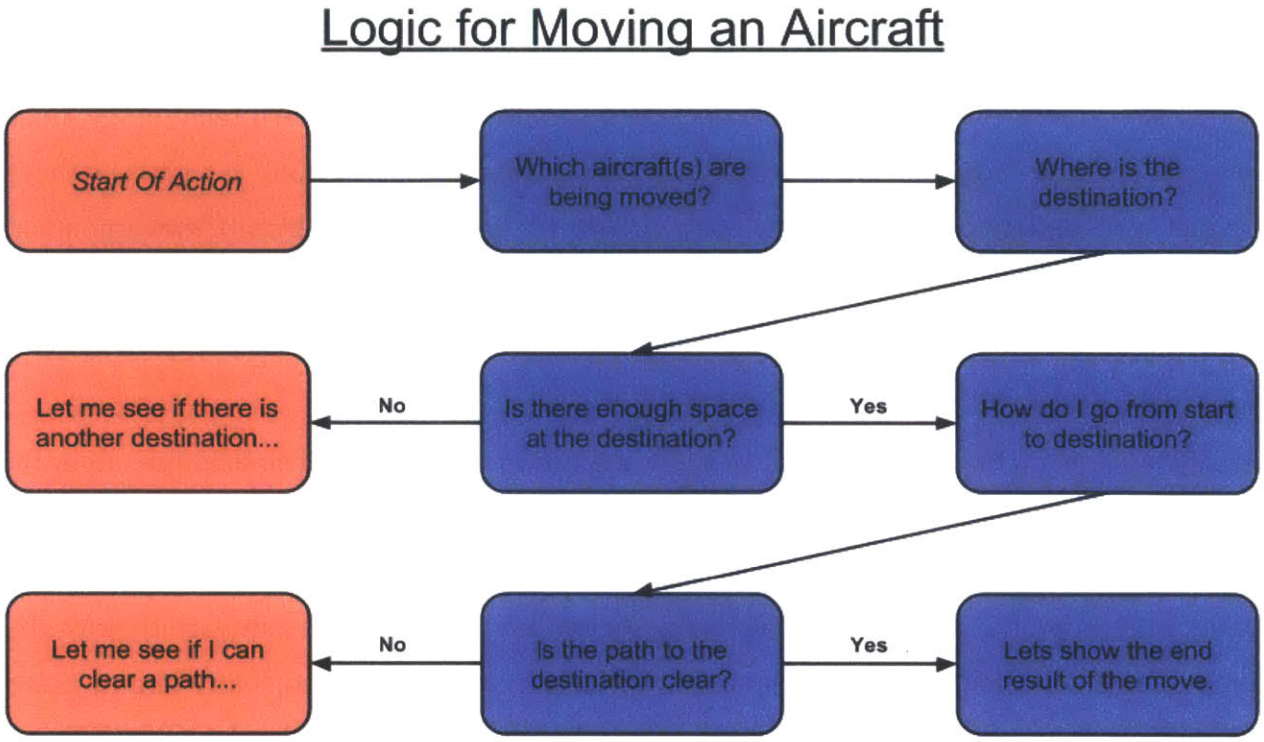


Figure 2-1: Logic for moving one or more aircraft to a destination. An initial action may result in subsequent or alternate actions depending on the state of the deck.

2.1.2 Current Actions

The current actions within DeckAssistant are:

- Moving aircraft from a start location to a destination.
- Clearing a path for aircraft to move from a start location to destination.
- Finding an alternate placement of aircraft when the intended destination is full.
- Moving aircraft to launch on one or more catapults. This may involve queuing aircraft behind catapults for launch.

2.2 System Understanding

Our system understands the deck environment, including different types of aircraft, space on deck, and possible moves for aircraft initiated by the deck handler.

2.2.1 Understanding of Deck and Space

Traditional Ouija Boards serve as scale models of the deck so that their display can represent realistic situations on deck. Similarly our simulation represents aircraft as scale models and tracks their status on the deck. With traditional Ouija Board technology, deck handlers refer to various regions on deck using names. Our system recognizes these regions by name, allowing deck handlers to specify them during aircraft movement. In addition, the system can determine if a given deck region has enough room for additional aircraft and check if there is a clear path to that deck region.

2.2.2 Aircraft and Destination Selection

Our system allows the user to select aircraft using various methods. The example scenario in Chapter 1 includes two different types of aircraft on deck (F-18 fighter jets and C2 transport jets). When initiating a Deck Action, a user can select an aircraft by pointing to it on deck or giving its tail number (displayed above each aircraft in the simulation) with the command. When pointing with a command for selection, the user can select one or more aircraft. Similarly, as previously noted users can denote regions on deck using names (such as the "*Fantail*") or point to destinations for placement.

2.2.3 Simple Path Planning & Rerouting

During aircraft movement, DeckAssistant calculates clear paths from aircraft start locations to specified destinations, taking account of aircraft's geometry (i.e. wingspan length).

In addition to path planning, DeckAssistant can find alternate destinations and routes when moving aircraft. Typically, if a specified destination cannot fit the aircraft being moved, DeckAssistant looks at a set of neighboring deck regions to arrange the additional aircraft, using a shortest path metric.

2.3 Multimodal Input and Output

Users communicate with our system using a combination of gestures and speech, DeckAssistant responds in kind with synthesized speech and graphics.

Our system interprets gestures by tracking hands over the tabletop. We calculate the orientation and position of hands and fingertips above the tabletop. In addition, we interpret pointing gestures from tracked hands for object selection. The current set of commands makes use of only pointing gestures towards aircraft and destinations on deck.

The user issues commands by speaking into the system microphone. Commands use one or more pointing gestures or operate based on speech alone. For instance, the user can move an aircraft by stating the aircraft's tail number followed by the name of the destination. Alternatively, the user can point to the desired aircraft and then point to the desired destination while giving a more general command (e.g. "*Move this aircraft over here*").

When pointing at the tabletop, the system's understanding of the pointing location is indicated with an orange dot (see Figure 2-2). When the pointing location is moved over aircraft, the corresponding aircraft is highlighted in flashing green to show their potential selection for Deck Actions (see Figure 2-3). As the user issues a command that manipulates aircraft on deck, the system indicates the selected aircraft for the command with a flashing orange highlight (see Figure 2-4(a)). The system can select single or multiple aircraft (see Figure 2-4(b)).

Figure 2-2: Pointing location is shown with orange dot.

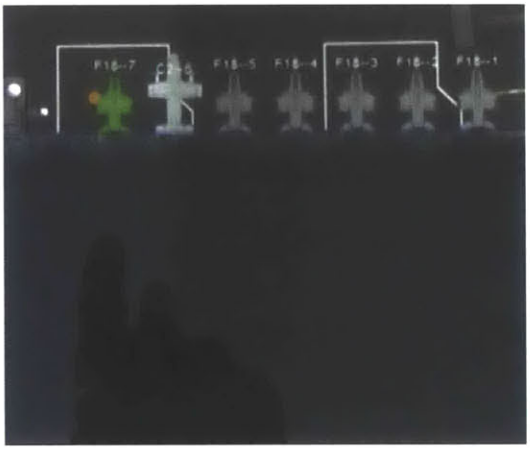
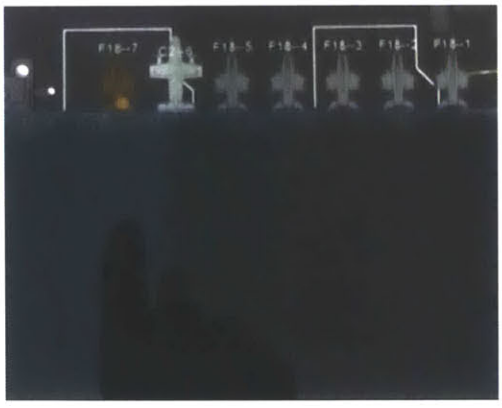
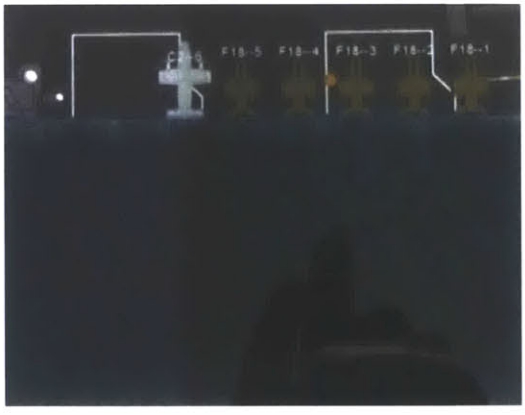


Figure 2-3: Highlight of an F-18 (in green) based on pointing location.



(a)



(b)

Figure 2-4: Aircraft (single or multiple) selected for an action are highlighted in orange.

The system responds to user commands based on the type of action and situation on deck. Typically after receiving a command from the user, the system updates aircraft on deck and speaks confirmation: "Ok done". If the system determines the command cannot be completed without additional actions, it will explain the situation using speech and graphics and may illustrate additional suggested actions, then ask for approval. If the user accepts, the system updates the deck, with spoken confirmation. If the user declines, the deck is reverted to its original layout, and the final response is instead: "Ok, please give another command".

When choosing additional actions, our system employs a variety of graphics and speech to illustrate changes to the user's original command. In the first chapter, we presented a scenario where the system determines that an aircraft launch cannot complete due to a blocked path. The system illustrates this by first drawing the path in orange while highlighting the blocking aircraft in red (see Figure 2-5). Then, it determines an alternate placement for the blocking aircraft. This change is illustrated with a blinking shadow for the potential placement of this aircraft (see Figure 2-6). When the user accepts the move, the deck is updated to the final results (with spoken confirmation).

In another situation, the system may need to choose an alternate destination for moving aircraft. Consider the case when a user tries to place an aircraft on a region of the deck that cannot accommodate it. Figure 2-7 gives an example where a C2 cannot be moved to the fantail due to the lack of space. Once again, the system highlights blocking aircraft (the aircraft on the fantail) in red and explains to the user the situation. The system then presents an alternate solution: moving the C2 to near the rear elevator instead, keeping it as close to the original destination as possible. Figure 2-8 shows how we highlight a new path and circle the potential alternate destination in blue. As before, if the user accepts these changes, the deck is updated, with a spoken confirmation. If the user declines, the deck is reverted to its original state.

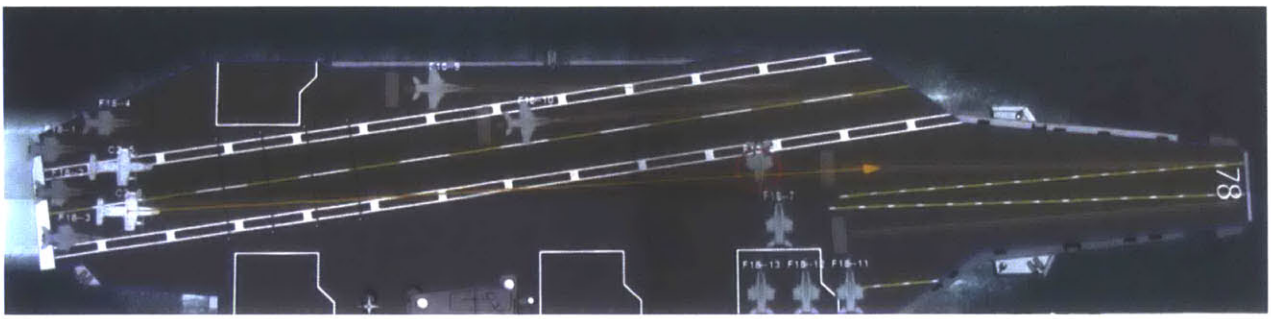


Figure 2-5: Calculated path for aircraft movement shown in orange. Aircraft blocking path circled in red.

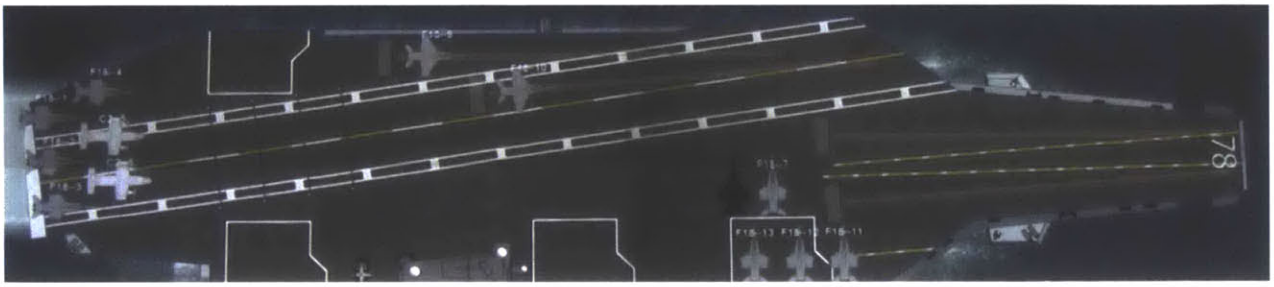


Figure 2-6: Indication of alternate placement for blocking aircraft using a shadow image.

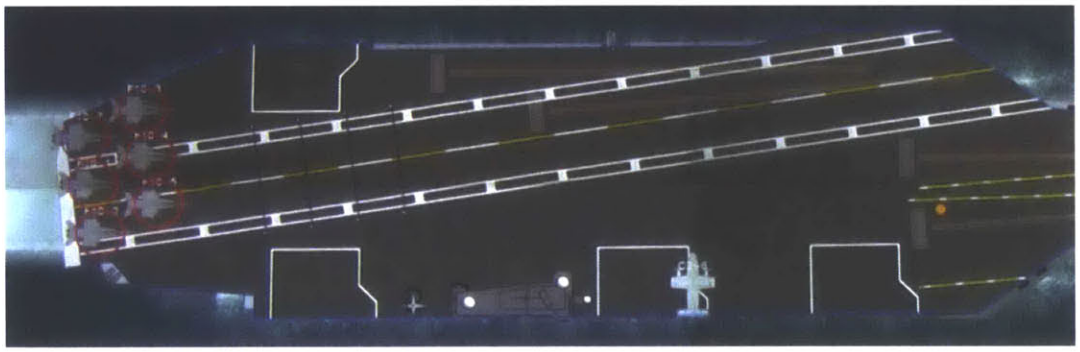


Figure 2-7: Command issued to move the C2 (white aircraft) to the Fantail, but there is not enough room. The aircraft filling the fantail are circled in red.

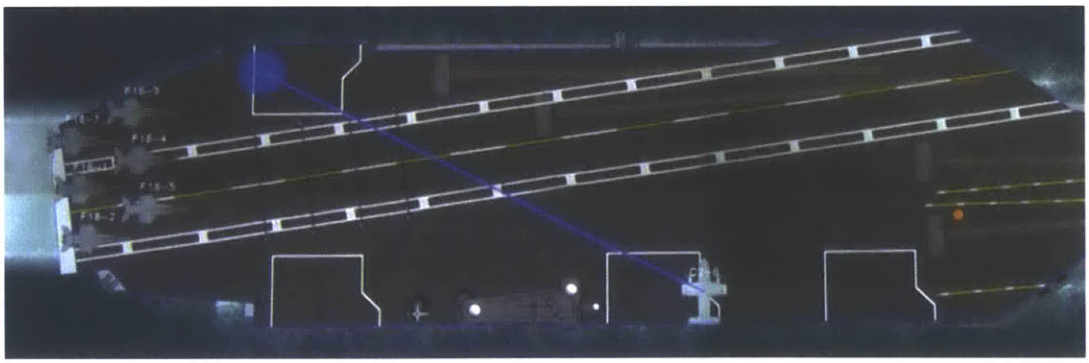


Figure 2-8: Indication for alternate destination for moving the C2 (shown in blue).

2.4 Example Speech Commands and Actions

We provide some example commands to illustrate ways users can command aircraft in our system:

- "*Move this aircraft to the fantail.*" Spoken while pointing to the selected aircraft.
- "*Move this aircraft over here.*" Spoken while first pointing to the selected aircraft followed by pointing to aircraft destination.
- "*Move aircraft #18 to the fantail.*" Spoken without additional gestures. Aircraft selection is based on tail number (visible on the display).
- "*Move this F-18 to launch on catapult 1.*" Spoken while pointing to a specific F-18.

Chapter 3

3 System Implementation

In this chapter, we detail our hardware setup for running DeckAssistant and give an overview of DeckAssistant software.

3.1 Hardware Setup

Figure 3-1 shows an overview of our system hardware. Our system consists of a large tabletop display, powered by a Windows 7 desktop computer with an AMD Radeon HD 6870 graphics card. We use 4 downward-facing projectors mounted over the tabletop (Dell 5100MP) to create a 42 by 32 inch seamless display with a pixel resolution of 2800 x 2100. Between the projectors sits a depth-sensing camera (Kinect V1) for tracking hands over the table surface. Beside the Kinect is a webcam (a Logitech C920) viewing the entire tabletop surface. We use this webcam to calibrate our seamless display with ScalableDesktop Classic software.

To support two-way conversation, the user wears a wireless Bluetooth headset. The headset connects to the computer through a USB D-Link Bluetooth Dongle (DBT-120).

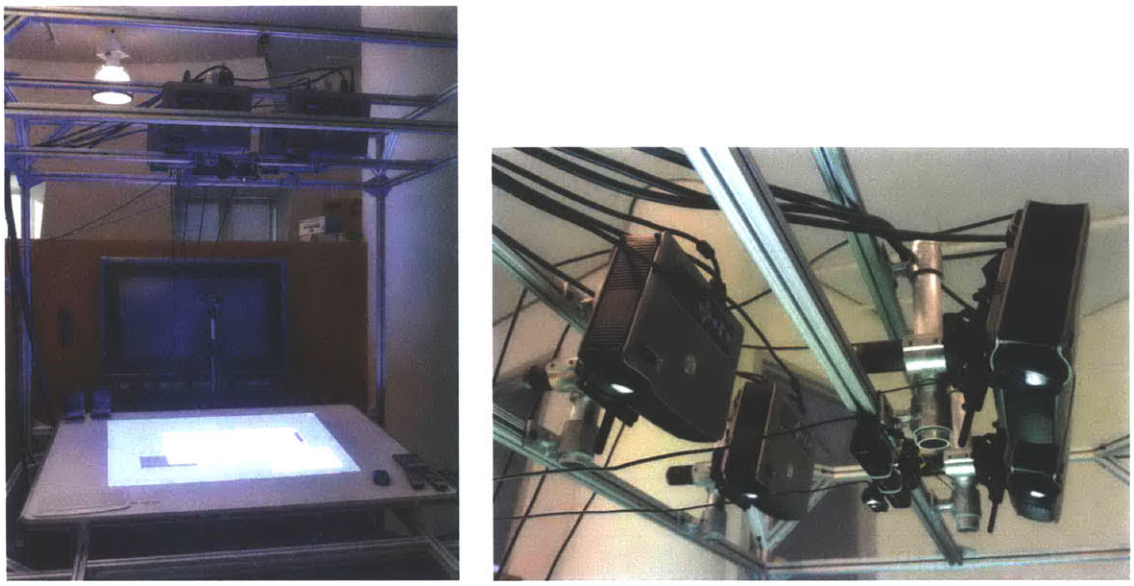


Figure 3-1: An overview of our seamless display hardware. Mounted above the table are 4 projectors, a Kinect sensor, and a webcam. A Windows 7 desktop sits under the table.

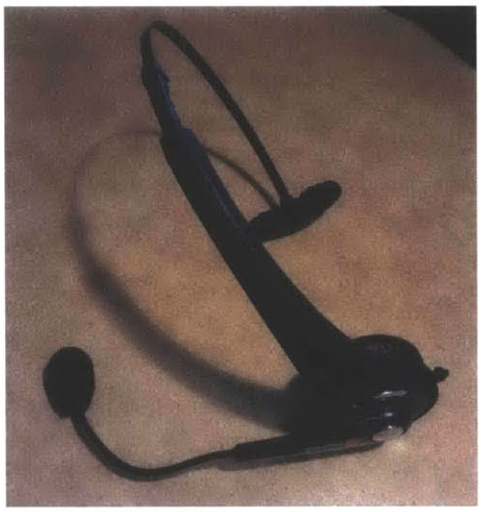


Figure 3-2: Wireless Bluetooth headset used to talk to the system.

3.2 Software Components

The deck simulation runs in a single application, which we call DeckAssistant. The software is written in Java and handles all functionality including graphics, speech recognition,

speech synthesis, and hand tracking. All processing is done internally without Internet communication. Table 3-1 lists open source libraries used in our application:

Library Name	Use
OpenNI	Hand tracking
SensorKinect	PrimeSense sensor module for OpenNI to interface with the Kinect sensor
OpenCV (JavaCV wrapper)	Geometric and image processing
CMU Sphinx 4	Speech recognition
FreeTTS	Speech synthesis
Processing (processing.org)	Graphics and underlying application framework

Table 3-1: Open source libraries used in DeckAssistant

3.3 Software Architecture Overview

3.3.1 Handling Multimodal Input and Output

We organize multimodal input/output into 3 software stacks, denoted as "multimodal stacks". The 3 stacks implement gesture recognition, speech recognition, and speech synthesis and are composed of modules that continuously monitor input while driving output. The 3 stacks work in parallel and provide APIs (application program interface) that allow any module within DeckAssistant to initiate multimodal interaction.

Figure 3-3 diagrams the components of each multimodal stack. Speech synthesis handles sentence construction and audio generation through the system's speakers. Speech recognition listens to the system's microphone to parse speech and relays the commands to the action framework. Gesture recognition interfaces with the Kinect for raw depth data to calculate hand and finger placement orientation over the table.

The action framework, which manipulates the deck based on user commands, leverages these interaction stacks for input/output. When the user speaks an initial command into the system microphone, the speech recognition stack parses the input to

create the appropriate action with the system. The logic in these actions logic for manipulating the deck and continue user interaction through the 3 stacks.

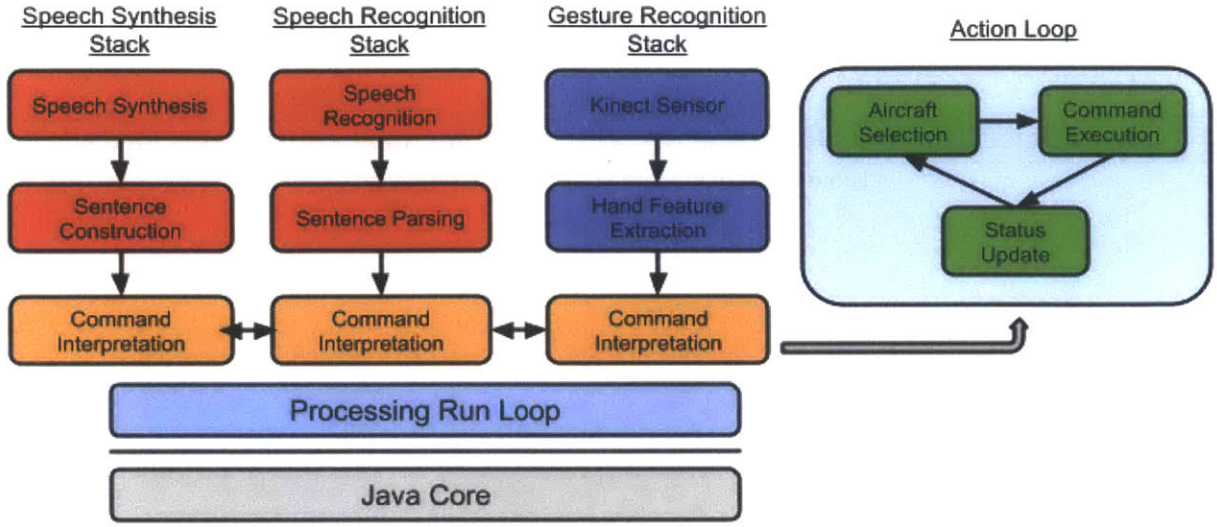


Figure 3-3: Each multimodal stack processes input/output in parallel while commands are handled jointly to drive further multimodal interaction.

3.3.2 Engines and Managers Overview

We divide the functionality within the multimodal stacks and the action framework into various modules within DeckAssistant. We refer to these modules as "Engines" or "Managers", depending on their functionality. While Engines and Managers share similarities in implementation, they differ in that:

- Engines primarily drive functionality within the system, such as speech recognition or hand tracking.
- Managers primarily track and update data structure state within the system.

Table 3-2 details the Engines and Managers that makeup DeckAssistant. We provide further details on their implementations in **DECKASSISTANT SOFTWARE GUIDE**. See the Appendix section for more info.

Module/Engine Name	Function
Speech Recognition Engine	Recognize spoken commands from the user
Speech Synthesis Engine	Generates speech responses to the user

Hand Tracking Engine	Track user's hands and pointing gestures over the tabletop
Selection Manager	Handles aircraft selection
Action Manager	Manages all actions used to update deck state
Flying Object (Aircraft) Manager	Manages all aircraft objects on deck

Table 3-2: Engine and Manager functionality.

Chapter 4

4 The Deck Environment and Objects

This chapter details objects that represent the deck, catapults, elevators, and aircraft in DeckAssistant.

4.1 Aircraft on Deck

Each aircraft on deck is represented by a unique instance in the code. Each aircraft instance stores the position, type of aircraft, status, and any other information relevant to the aircraft during the simulation. All aircraft are updated and rendered through the Flying Object (aircraft) Manager. The manager also provides an interface for querying aircraft based on position, status, or aircraft type.

4.2 Deck Environment

4.2.1 Deck Object

The Deck Object represents the entire deck. The Deck Object is responsible for updating and maintaining the parking regions and installations (catapults, elevators) on deck. Actions that move aircraft on the deck always check the Deck Object ensure the resulting deck layout is always valid.

4.2.2 Deck Regions and Parking Spaces

To simplify the understanding of aircraft placement on decks, deck handlers have a naming scheme for particular regions of the deck. Figure 4-1 shows map of common deck regions referred to by deck handlers during aircraft placement. When operating a traditional Ouija Board, deck handlers often receive aircraft placement information from deck spotters referring to these deck regions, which they'll use to update the placement of aircraft on deck.

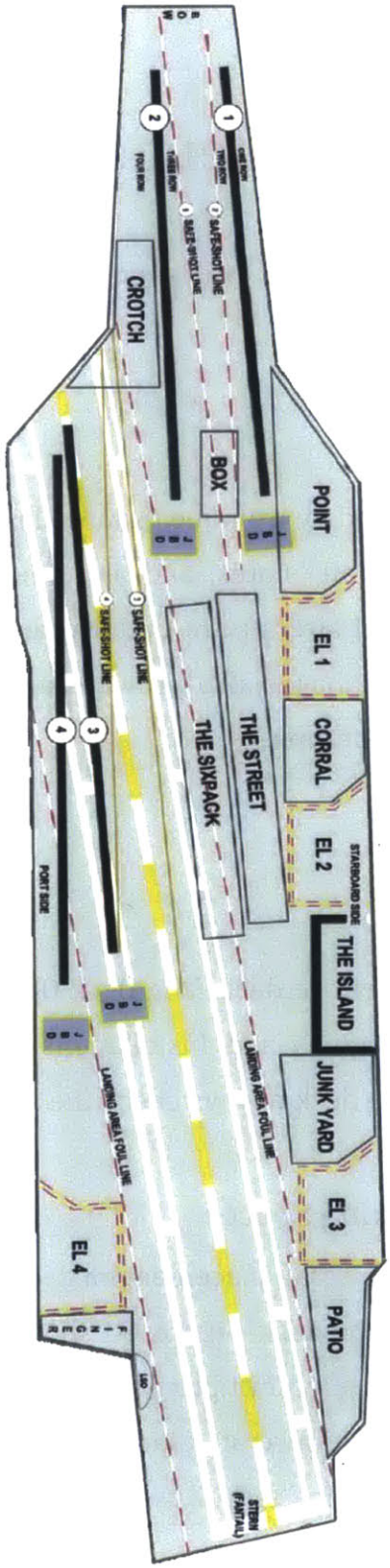


Figure 4-1: Map of well defined regions on deck.

Our system understands these deck regions by name as well. Each deck region maintains an array of parking spots, which define the arrangement of parked aircraft and track occupancy within the deck region. Using this information, our system can arrange aircraft in deck regions by command. For instance, if a deck handler points to a series of aircraft and give the command "*move these aircraft to the fantail*", the system can move the aircraft into a similar arrangement shown in Figure 4-2. Similarly, when rerouting aircraft or clearing blocked paths, the system uses parking spots to arrange all moved aircraft.

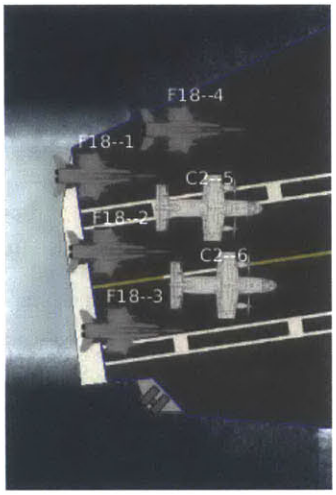


Figure 4-2: The "fantail" (sometimes referred to as "stern") is a deck region that contains 6 parking spots.

In addition to parking spots in deck regions, we maintain a separate type of spot for aircraft placement, known as a queue spot. Queue spots place aircraft for use on designated deck installations. We use queue spots for launching aircraft on the catapults and queuing additional aircraft in line behind catapults (see Figure 4-3). Each deck elevator also contains a corresponding queue spot. Queue spots are used with commands associated with deck elevators and launching aircraft.

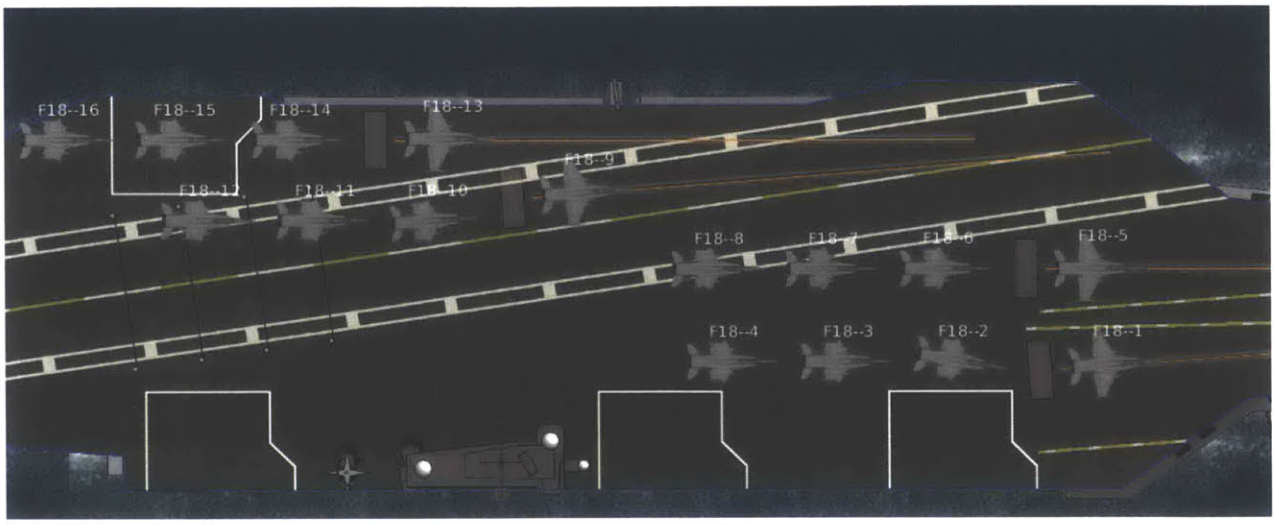
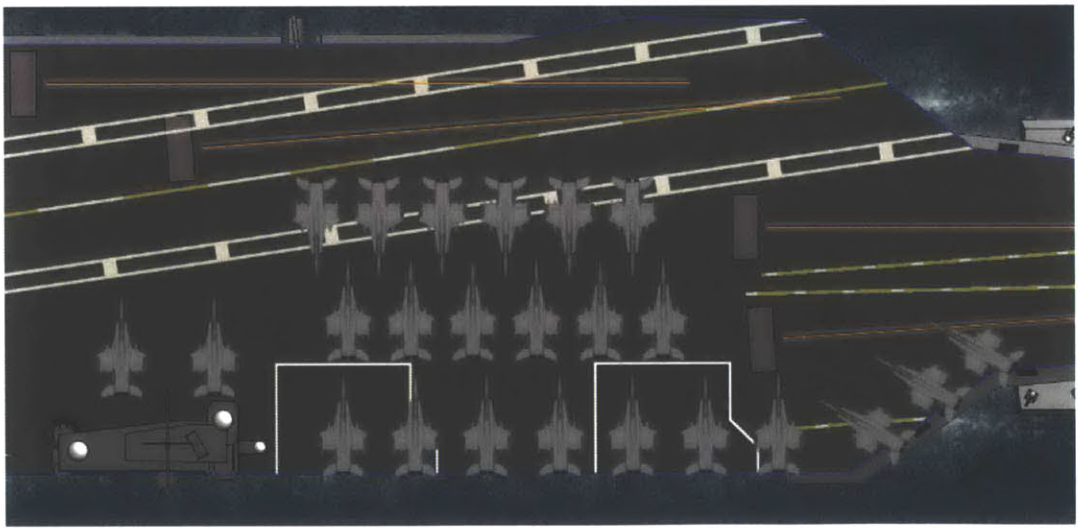
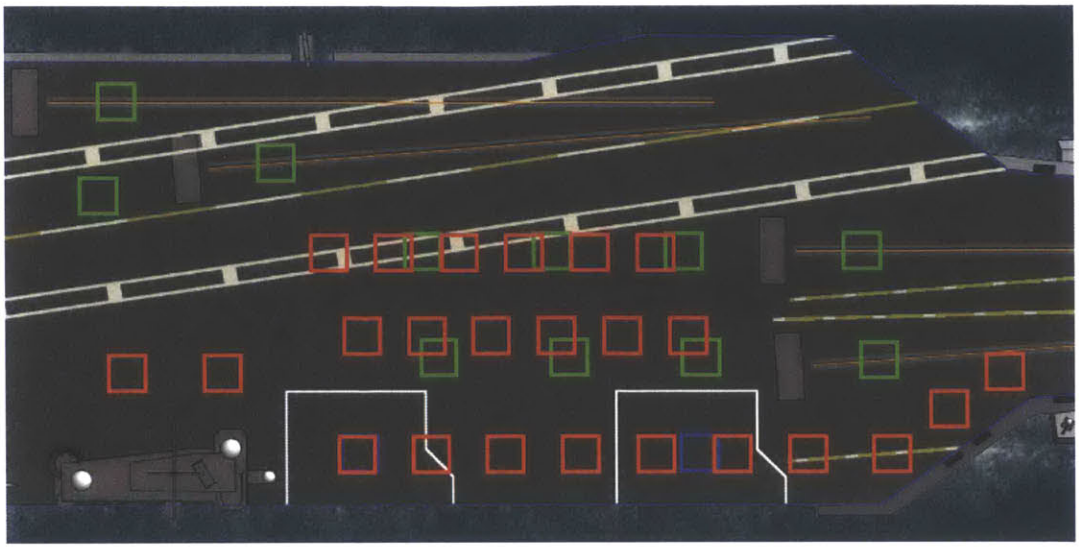


Figure 4-3: Aircraft queued behind catapults for launch (using queue spots).

Due to their different uses and arrangement, queue spots and parking spots overlap on several parts of the on deck. Because the system won't place aircraft in occupied spots, we can make use of either queue spots or parking spots under different situations. For instance, the deck handler can queue aircraft behind catapults 1 and 2, as shown in Figure 4-3. At another time, the deck handler can also use same region, known as the "street", for general aircraft parking (as shown in Figure 4-4(a)) which blocks the queue spots for lining up aircraft behind the catapults.



(a)



(b)

Figure 4-4: (Top) aircraft placed in parking spots in deck regions. (Bottom) Empty deck with parking spots and queue spots marked: parking spots (red), catapult queue spots (green), and elevator queue spots (blue).

In practice, the configuration and names of deck regions vary slightly between carriers and among personnel. For demonstration, we chose deck regions most representative across these variations. Both deck regions and their parking spot layouts are easily configurable to accommodate differences between aircraft carriers. In addition, deck handlers can always use gesture rather than names to indicate areas on deck (the system always indicates the potential placement of aircraft before a command is complete).

4.2.3 Paths and Space Calculations

Paths are the primary mechanism for planning routes on deck. Paths are composed of points and connecting lines. Path lines have width based on the wingspan of aircraft being moved (this accounts for folded and open winged aircraft). These widths give paths area on deck, which the system tests for blocking aircraft.

In the current iteration of DeckAssistant, path generation is limited to straight lines from one location to anther (see Figure 4-5), but the framework supports complex paths with many turns. A more advanced path planner could test multiple paths in searching for a route to a destination (see Figure 4-6).

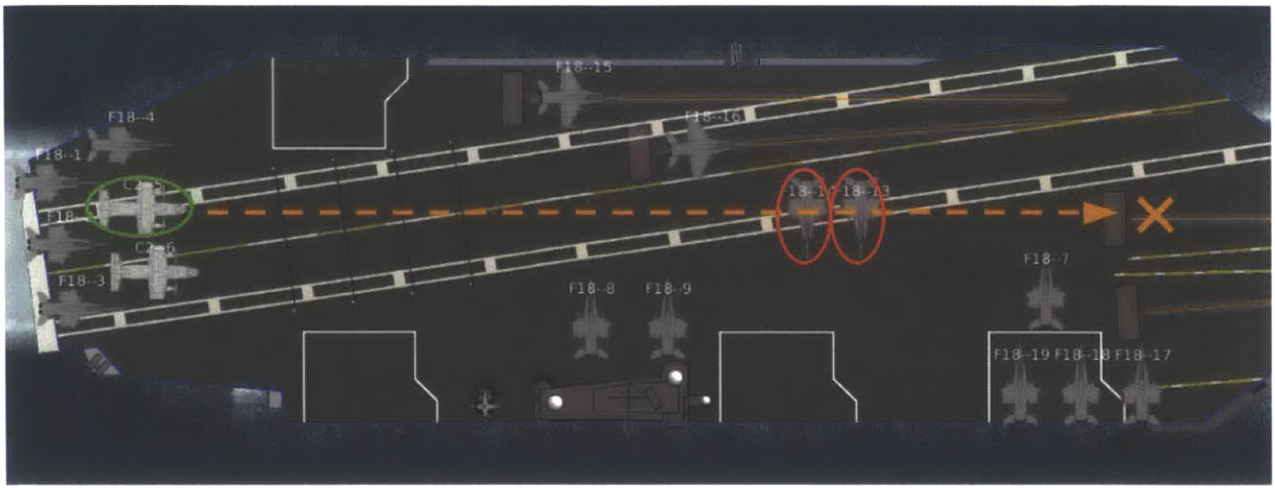


Figure 4-5: C2 aircraft (circled green) follows basic path to takeoff catapult. Basic path is a straight line that intersects two parked aircraft.

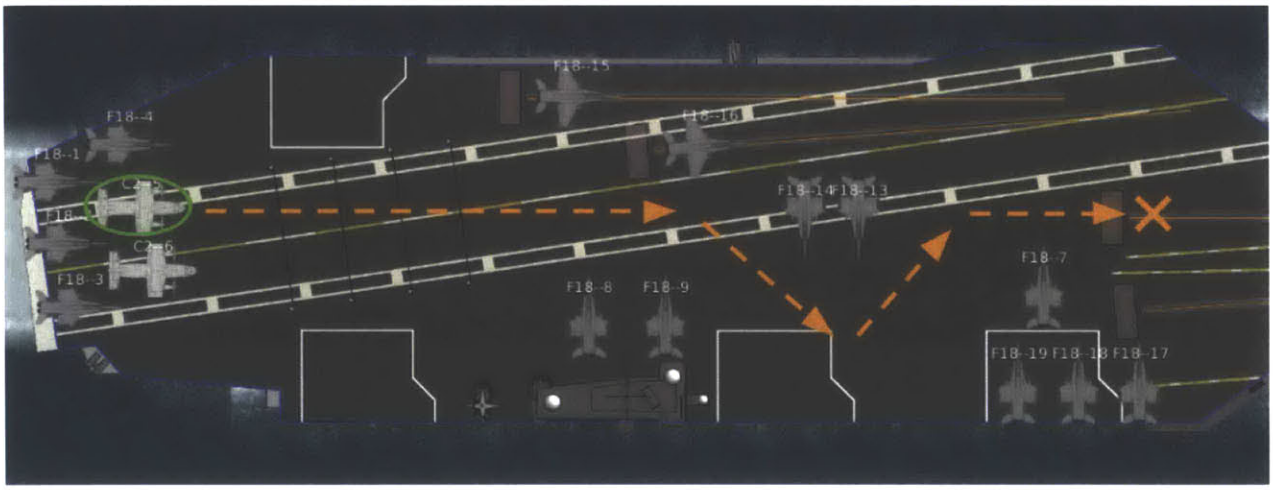


Figure 4-6: C2 aircraft (circled green) follows robust path to takeoff catapult. Path maneuvers obstructing aircraft.

Chapter 5

5 Hand Tracking

In this chapter, we discuss methods for tracking hands and interpreting pointing gestures. This consists of two main parts: identifying hands and fingers and interpreting pointing using tracking information.

5.1 Hand and Finger Tip Tracking Over Tabletop

We expand on work by Ying Yin et al [2] for tracking hands over a horizontal displays. Our system segments and identifies hands and fingertips.

We track the user's arms and hands with a Kinect sensor mounted over the tabletop. While the Kinect provides both an RBG image and a depth image, we use only the depth image for tracking. The depth image steam has a resolution of 640x480 pixels with 11-bit depth values at 30fps. We use Java bindings in the OpenNi framework to interface with the Kinect. We also use Java OpenCV wrappers (JavaCV) for geometric calculations.

We start by preforming a one-time calibration process map to map the Kinect depth image to pixels projected on our tabletop. During the deck simulation, the general process for hand and finger tracking has 3 steps:

1. Background subtraction
2. Upper limb and hand segmentation
3. Fingertip identification

We describe calibration first, then detail the general tracking method.

5.1.1 Kinect Calibration

We start by calibrating our display so we can align hand-tracking information with pixels on the tabletop. For calibration, we present a checkerboard image across the display. The user places 16 blocks on the checkerboard image, then marks a pixel location for each block (top left corner) on the checkerboard image and the corresponding pixel location in a

captured depth image from the Kinect (32 points total). We use the 16 pairs of points between the depth image and display pixels to compute a planar holography that maps the depth sensor image pixels to the digital display pixels.

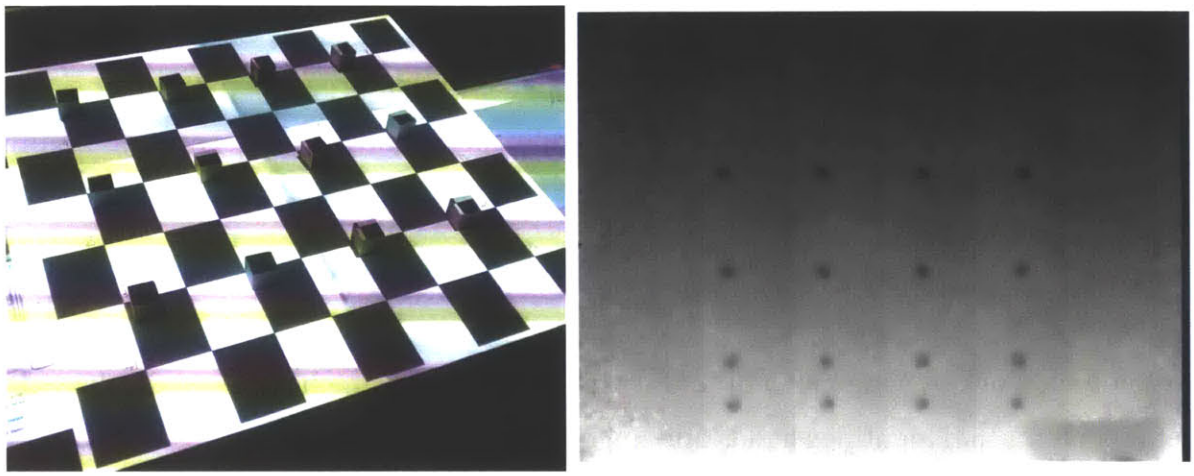


Figure 5-1: 16 blocks correlate specific pixels in the display and the depth image.

5.1.2 Background Subtraction

In order to follow hands over the tabletop, we separate objects in the foreground, presumed to be hands and limbs, from the static background seen by the Kinect. We identify the background by recording the first 2 seconds (i.e. 60 frames) of frames from the Kinect depth sensor before the user places their hands in the scene. We take a per-pixel average of these frames to get an estimate of the background depth. With each subsequent frame, we create a foreground/background mask that ignores pixels that are within a threshold of the estimated background. The resulting foreground separation still contains some noise; we clean the image using morphological opening operations [6].

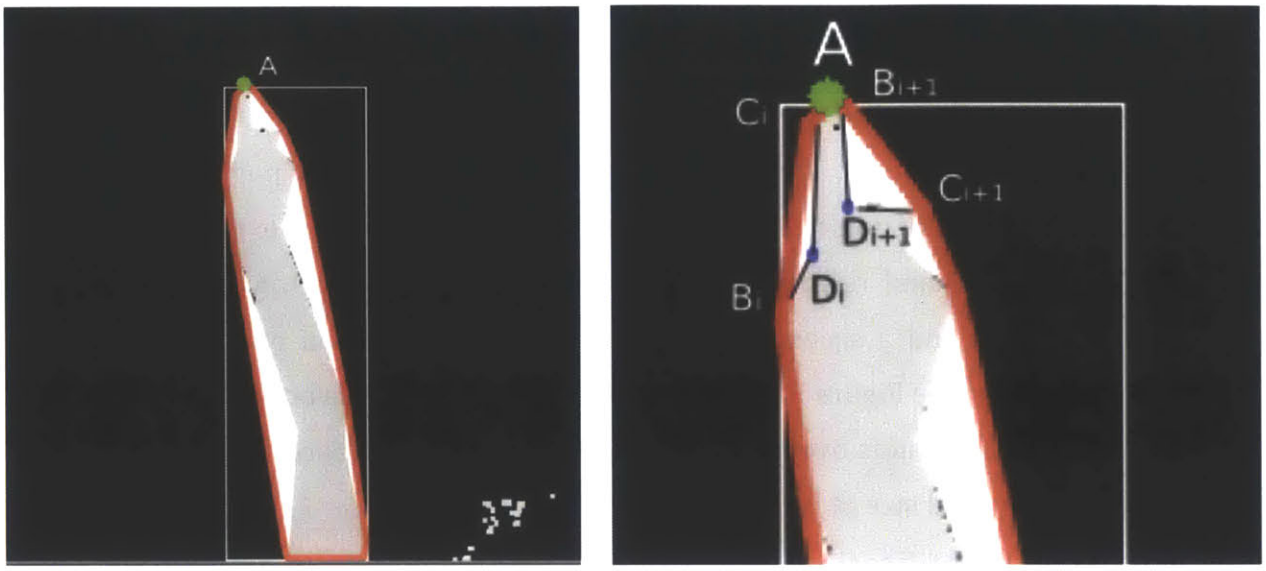
We have noticed that at time there is a slight drift in Kinect depth measurements over extended periods of use that lead to the background bleeding into the foreground. If at anytime, the user recognizes the system has trouble isolating their hands and limbs, they can easily repeat the background identification process.

5.1.3 Upper Limb and Hand Segmentation

We assume that limbs extended over the table enter our depth image from a single edge and extend some distance into the tabletop region. With our foreground/background mask, we find limbs by computing convex hulls and bounding boxes for each continuous foreground object seen. We set a minimum threshold on bounding box perimeter to screen out noise and limbs that don't extend far enough over the table. Next, we generate a bounding box for the hand region, which is taken to be the portion of the limb farthest away from the edge, and a second bounding box around a portion of the forearm that enters into the frame (see Figure 5-3). We estimate the sizes of these bounding boxes based on the expected size of hands over our tabletop (based on the distance of the Kinect above the tablet and the typical size of hands).

5.1.4 Fingertip Identification

We find fingertips using the technique in Ying Yin et al [2]. We use the depth pixels contained in the bounding box for the hand. This method identifies small, extended cylindrical portions of foreground depth as fingertips. From the convex hull, we compute the convexity defects to get a general sense of the hand shape (shown by the white areas in Figure 5-2(a)). Figure 5-2 shows how an extended finger exhibits a convexity defect on each side. Within the inner bounding box for the hand, we iterate through pairs of adjacent convexity defects, for example, $\Delta B_i C_i D_i$ and $\Delta B_{i+1} C_{i+1} D_{i+1}$ in the close-up view of Figure 5-4(b). We look for an acute angle between the two segments, $C_i D_i$ and $B_{i+1} D_{i+1}$ (less than 45°) and the distance between the points D_i and D_{i+1} to be greater than the finger width threshold (14mm). If these checks pass, we mark the midpoint of $C_i B_{i+1}$ as the fingertip. We then refine the fingertip position by searching the depth gradient along the finger (parallel to lines $D_i C_i$ and $D_{i+1} B_{i+1}$) for a sharp change in the gradient of the depth value and assign that point as the final calculated fingertip. This process repeats until we've searched all adjacent pairs or identified 5 fingertips.



(a) (b) Closeup view.

Figure 5-2: The white triangular areas are convexity defects and the red outline is the convex hull (Source: Ying Yin et al.).

Figure 5-3: Raw depth image shown on left (lighter pixels are closer to camera), final result after background subtraction and finger identification shown on right. The red box shows the hand regions, the blue box shows the forearm region that enters the frame. Green dots are calculated fingertips.

5.2 Pointing and Selection

From hand tracking information, we interpret pointing gestures from users. This process involves building a 3D model of the tabletop surface and intersecting it with rays extended from identified fingertips. We also apply additional filtering and processing to improve

pointing accuracy and responsiveness. The general process for object selection computed on each incoming depth frame has 3 steps:

1. Arm sampling and ray extension
2. Kalman filtering the intersection point
3. Object identification and selection

5.2.1 Table Surface Model

At the start of each run of DeckAssistant, we construct a 3D model of our tabletop surface using data from the background identification process, as in Ying Yin et al. [2]. We assume the tabletop surface is flat and fills the majority of the background. We take 10 evenly spaced depth samples along the center horizontal axis and another 10 evenly spaced depth samples along the vertical axis (20 samples total), as shown in Figure 5-4. We use a linear regression to fit a line to each axis of points (this relieves noise in the depth samples), then generate a 3D plane from our intersecting lines. Using the computed 3D plane and our planar homograph from Kinect calibration to the table surface, we can correlate any point on our 3D tabletop surface to a pixel location in the Kinect image, and subsequently to its corresponding pixel location on the tabletop display.

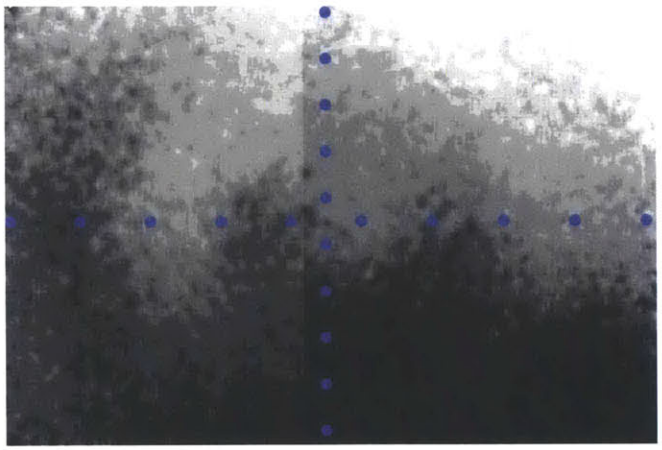


Figure 5-4: Depth image of the bare tabletop surface. 10 horizontal and 10 vertical depth samples are used for generating the table surface 3D model. Our tabletop is mounted at a slight angle (lighter pixels are closer to camera).

5.2.2 Arm Sampling and Ray Extension

We extend rays in 3D using tracked limbs over the tabletop to determine pointing targets on the table surface. First, we compute the centroid of all foreground depth points within

the inner forearm bounding box (generated during upper limb segmentation). This centroid serves as the forearm point; we extend a ray from this point through an outstretched figure tip in 3D space (see Figure 5-5).

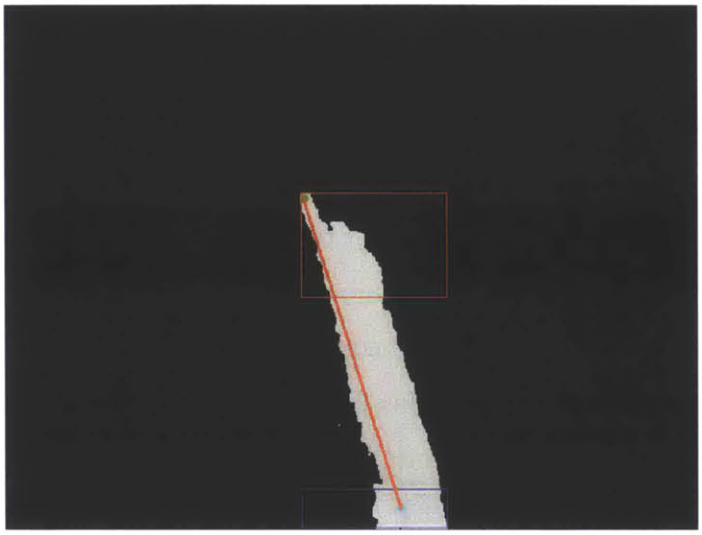


Figure 5-5: Pointing using the fingertip point (green) and the armjoint point (blue). Extended ray shown in red.

In practice, two points sampled from the Kinect are insufficient for reasonably accurate pointing interpretation. Figure 5-6 illustrates how small noise from measured depth at the fingertip and forearm creates a "seesaw" effect with large variations in the perceived pointing target. We sample additional points of depth along the arm using the pointing ray (typically 10 additional points are sufficient). The incline of the pointing ray is derived from fitting a linear regression through these points. The result is a much steadier estimation of pointing.

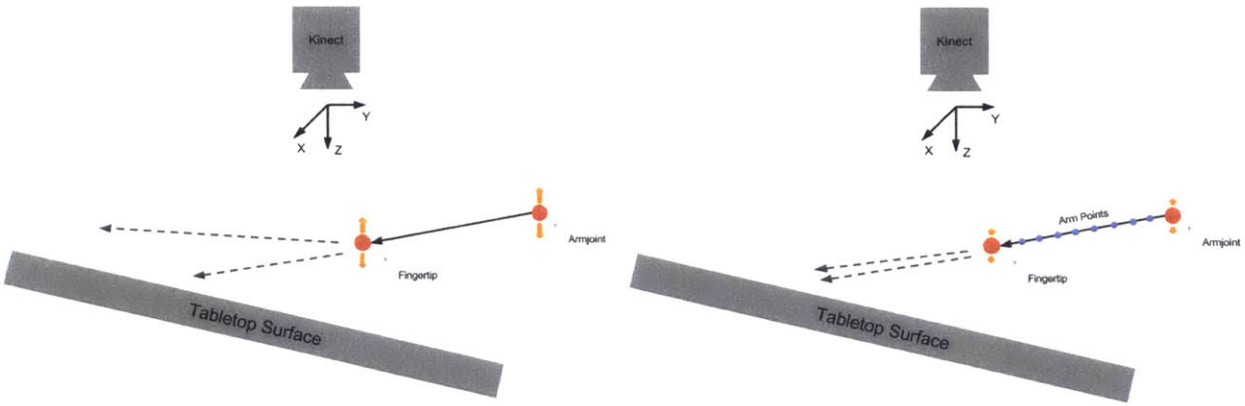


Figure 5-6: (On left) noise from just two sample points leads to noisy pointing interpretation along the extended ray. (On right) sampling additional depth along the arm provides better measurements.

Following the user's arm may still be perceived as slightly off center, since we often look from our eyes directly to our fingertip when pointing. To deal with this, we add a small angle rotation around the user's fingertip when extending our pointing ray (see Figure 5-7).

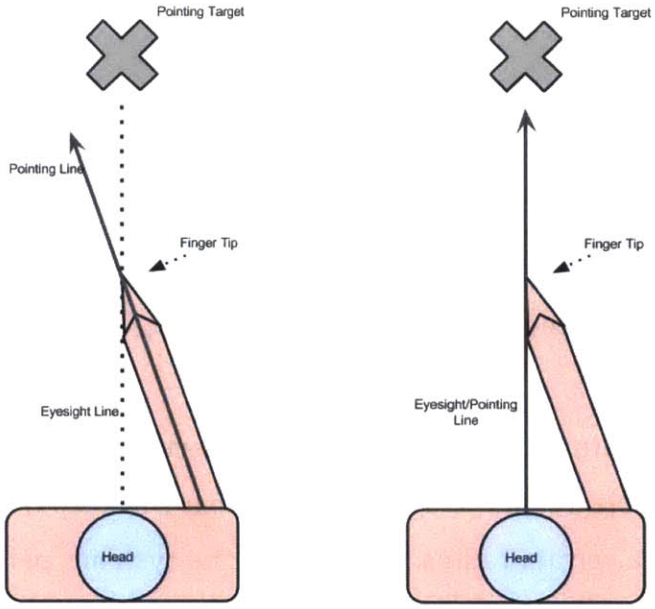


Figure 5-7: Pointing fix for offset between eyes and shoulder. The pointing ray is rotated around the fingertip.

5.2.3 Kalman Filtering

From our intersection point in 3D, we generate a 2D, pixel-mapped point on our tabletop display. We use a discrete Kalman Filter to smooth the motion of our pixel-mapped point

over time. Our Kalman filter is based on a constant-velocity model along 2D Cartesian coordinates. Assuming no external control, the a priori estimate of the state is modeled by:

$$x_t = Fx_{t-1} + w_t$$

The position and velocity of each intersection pixel are modeled accordingly in our state matrix:

$$x_t = \begin{bmatrix} x \\ y \\ v_x \\ v_y \end{bmatrix} t$$

F is our 4x4 state transition model that updates position based on velocity:

$$F = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

We assume the components of w_t have a Gaussian distribution $N(0,\Sigma_t)$, where Σ_t is the following covariance matrix:

$$\Sigma_t = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 10^{-2} & 0 \\ 0 & 0 & 0 & 10^{-2} \end{bmatrix}$$

The small variance in the last two values indicates a low level of uncertainty. This comes from the assumption that users generally point smoothly from one item to the next as pointing typically evolves some forethought.

5.2.4 Object Selection

We use the pixel coordinate from pointing to select objects based on their pixel locations on the screen. We also enable a small "magnetic" effect that will snap the pointing target to the closest aircraft within a certain radius. We indicate the system's perceived pointing target with a moving orange dot and indicate aircraft highlighted in real time as a visual aid (the pointing dot can be disabled at the user's discretion).

The end result is a system that feels natural and allows the user to point and select individual aircraft with the fine grain accuracy normally associated with using a mouse pointer. Furthermore, our filtering techniques make tracking pointing robust, given the difficulties exhibited from the overhead sensor and limited image resolution of the Kinect V1.

Chapter 6

6 Speech Recognition & Synthesis

This chapter details the speech technologies used to create interactive conversations between our system and the user.

6.1 Speech Synthesis

We chose the FreeTTS package as our speech synthesis for our fist iteration of DeckAssistant. FreeTTS provides a capable API and pure Java interface. In addition, FreeTTS is cross compatible with many operating systems (we developed DekcViewer on both Windows and Linux operating systems). We note that our open source synthesizer lacks the clarity and pronunciation quality of many proprietary speech packages, such as Microsoft's built in synthesizer and Dragon Naturally Speaking. However, DeckAssistant's design facilitates the future incorporation of proprietary packages for better speech.

The speech synthesis stack handles speech synthesis in DeckAssistant. Any module can access the speech stack through the Speech Synthesis Engine to generate speech. We use an event based programming model: speech events trigger speech to the user, which in turn generate speech completion events to notify the original source of speech events. If multiple speech events are sent simultaneously, the system queues and services them in the order they are received.

6.2 Speech Recognition

6.2.1 Speech to Text

Within DeckAssistant, all speech recognition is handled within the Speech Recognition multimodal stack, using the CMU Sphinx 4 framework. The framework uses a grammar (rules for specific phrase construction) to parse phrases we define for our application. Recognition is performed on each spoken phrase followed by a brief pause; when a phrase

in the grammar is recognized, a Speech Recognition Event that contains the speech text is passed to the Speech Recognition Stack.

6.2.2 Parsing Speech Commands

Each speech event contains text with the command from the user. Our system parses many commands in two parts. This 2-state command interpretation allows the system to recognize multiple gestures in tandem with a single speech command, while giving continuous feedback to the user. For instance, the command, "*Move this aircraft over here*", is parsed as, "*Move this aircraft...*, followed by, *"...over her."* before combined into a single command. While giving this command, the user first points to the desired aircraft, which is highlighted by the system as confirmation. The user then points towards the desired destination as they finish speaking the command. The full breakdown of speech commands is shown in Table 6-1.

Base Commands

Name	Function	Example(s)
Move Command	Select aircraft to be moved.	*"Move this F-18..."*
Location Command	Select destination of move.	*"...To the fantail."*
Launch Command	Select catapult(s) to launch aircraft on.	*"...To launch on Catapult 1."*
Yes/No Command	Respond to a question from the system.	*"Yes", "No", "Ok"*

Combined Commands

Name	Function	Combined From...
Move To Location Command	Move aircraft to a particular destination.	Move Command + Location Command
Launch Aircraft Command	Move aircraft to launch on one or more catapults.	Move Command + Launch Command

Table 6-1: Speech commands recognized by DeckAssistant. Combined commands use base commands for a more descriptive action.

Figure 6-1 shows the process for interpreting phrases and generating commands. First, the Speech Recognition Engine recognizes the speech. We parse text by assigning metadata that identifies the command type and any other relevant information. For combined commands, the system buffers the base commands parts until it can construct a complete command (see Figure 6-1).

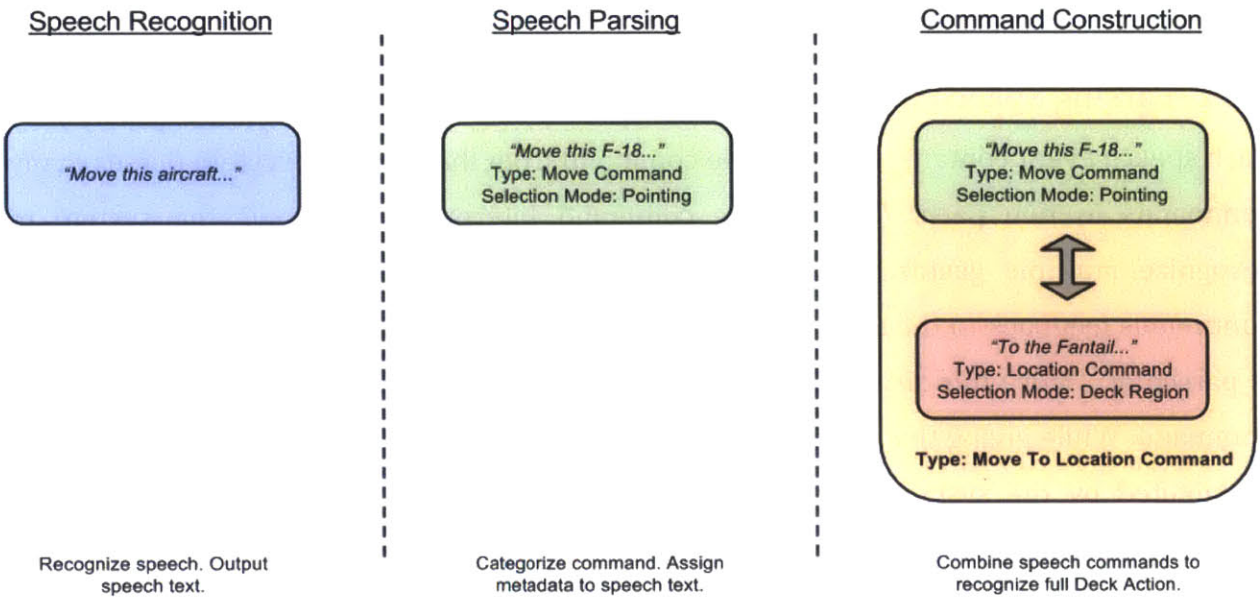


Figure 6-1: (From left to right) stages in recognizing commands from spoken speech. Note that two base commands are combined into one command.

6.2.3 Generating Deck Actions from Speech

We outline the complete process for creating Deck Actions from speech commands in Figure 6-2. We use a command for moving an aircraft to a region on deck as example. The user first speaks a command (a) that is interpreted by the Speech Recognition Engine. Speech parsing identifies the command and assigns metadata, which indicates this is a move command with pointing used for selection (b). The Action Manager receives the first base command and waits for additional input (c). We use metadata to select aircraft through the Selection Engine (d), allowing us to indicate selection as the user continues to speak to the system. The user continues by completing the rest of the command (e). Parsing assigns metadata to the second part of the command (f). With the second base command part, the Action Manager can complete the final command (g), and identifies the destined deck region through with the Deck Object (g). The complete command is used to create a Deck Action.

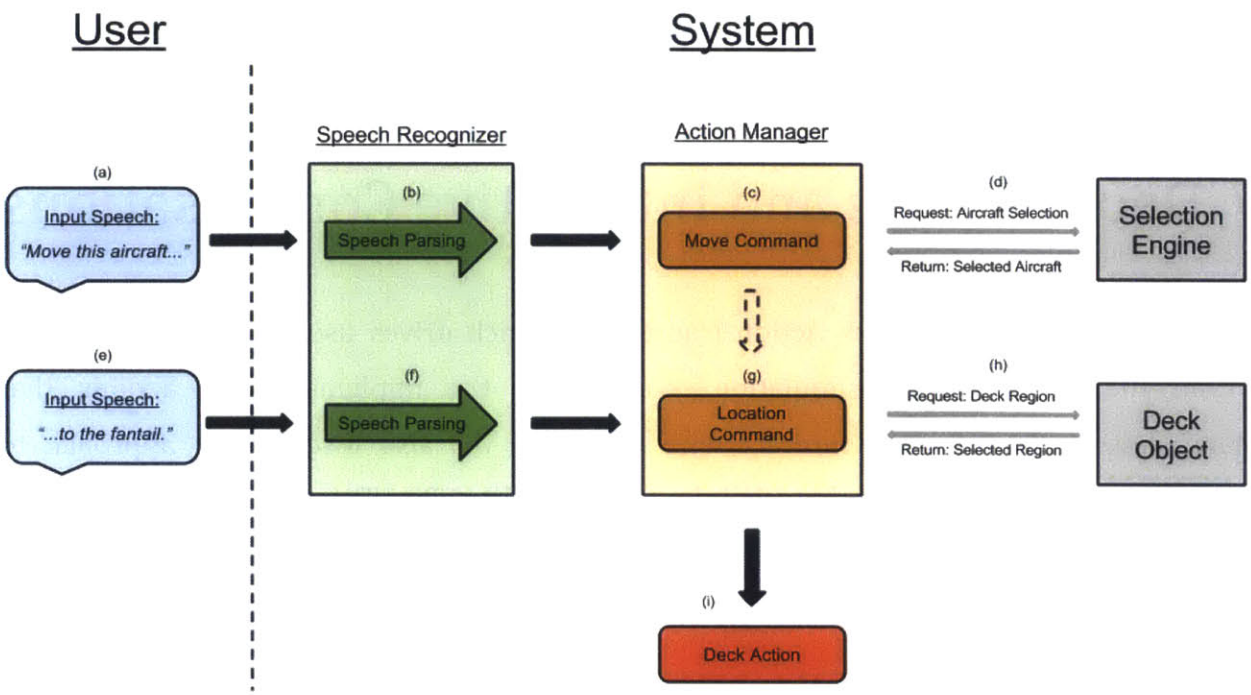


Figure 6-2: Process for parsing speech 2-step speech commands to create Deck Actions.

Chapter 7

7 Deck Actions and Interactive Conversations

This chapter details the Deck Action framework, which drives user-commanded actions within the DeckAssistant simulation. We explain the implementation actions that manipulate deck state and interact with the user. We also discuss our approach to combining multiple actions for complex functionality. Finally we detail the role of conversations in actions and our implementation of interactive conversations around task-oriented deck actions.

7.1 Overview of Deck Actions and Interactive Conversations

User commands initiate actions in the simulation, which we call Deck Actions. Deck Actions are the primary tool for planning and manipulating aircraft on our digital Ouija Board. Each Deck Action centers on a specific task communicated by the user. Deck Actions contain logic for manipulating the digital Ouija Board while initiating multimodal conversations with the user. Our system can combine multiple Deck Actions to create more complex functionality and handle a variety of situations when routing aircraft with meaningful feedback and potential options presented to the user.

7.1.1 Action Goals and Sub-Goals

Every user command indicates the desire for a particular situation on Deck. Within DeckAssistant, we organize actions around specific, well-defined goals. For instance, a user command to move an aircraft to a particular destination creates the appropriate Deck Action, called the Move Aircraft Action. The main goal of this action is to move an aircraft from start to destination while handling any subsequent details of the move.

For any given action, the main goal may not always be directly achievable. For instance, when attempting to move an aircraft, there may not be enough space at the specified destination or a clear path to reach the destination. In such cases, the system

recognizes this and initiates addition actions as needed, creating sub-goals that help complete the main goal (see Figure 7-1).

In DeckAssistant, we use additional Deck Actions for sub-goals. The result is users simply state the desired result on deck, and all additional actions are created automatically to achieve this result. As each action focuses on a specific goal, complex functionality results from combining multiple Deck Actions.

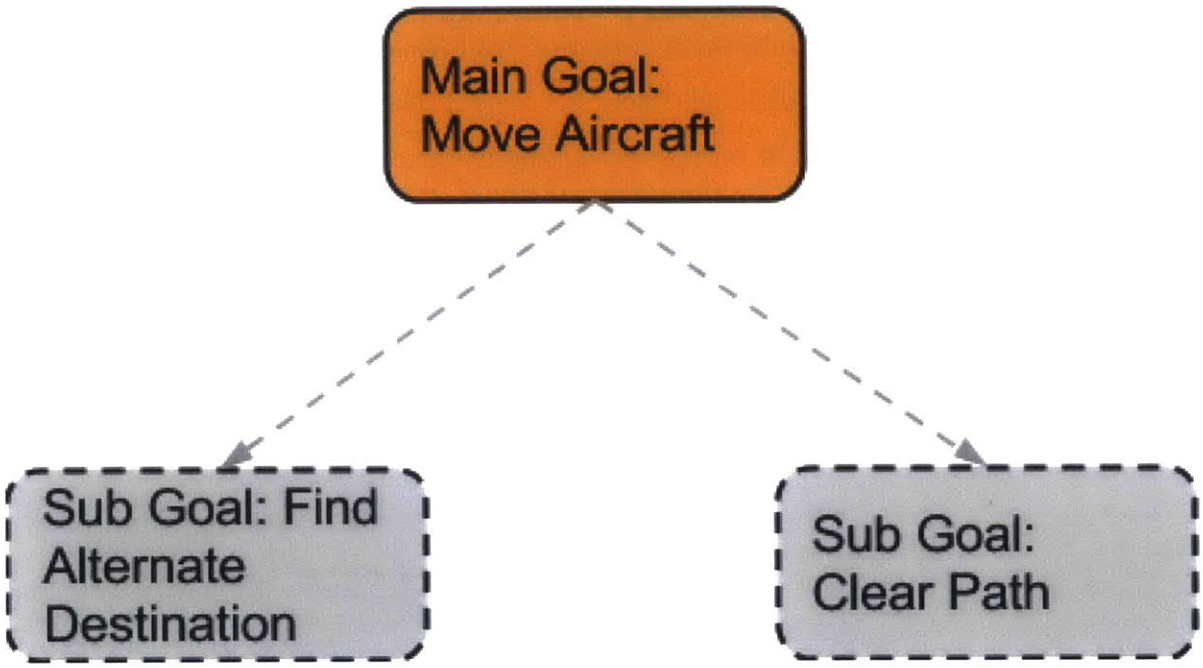


Figure 7-1: The main objective (commanded by the user) is to move the aircraft, however, the system may recognize the need to clear a path or find an alternate destination.

7.1.2 Interactive Conversation-Based Actions

When designing Deck Actions, we model their functionality around scripted, interactive conversations combined with logic for manipulating the deck. Within the conversation, a Deck Action engages the user for further input and makes decisions as it progresses. We model conversation as a graph of nodes, where each node is a specific element of user interaction or processing for the action. A Deck Action starts with the conversation at the root of the graph, which typically analyzes the state of the deck for the intended action. The system runs through the flow of the graph carrying out steps that typically involve:

- Accepting more speech or gesture input from the user.

- Producing speech or graphics output for the user.
- Processing the state of the deck, such as aircraft placement, path generation, etc. These can be considered as "hidden" parts of the conversation from the user.
- Dealing with points where the graph branches based on user input or deck state.

Figures 7-2 and 7-3 show example conversations for actions that move aircraft and find alternate destinations. The Move Aircraft Action starts by checking the validity of carrying out the move directly. If the path to the destination is blocked or the destination is full, the graph branches, leading to additional Deck Actions. If the move passes these validity checks, the deck is updated based on the move, and the user is notified of the end result. Similarly, the Find Alternate Target Action engages the user for additional input while processing deck state. Each Deck Action, or subsequent Deck Actions can have multiple outcomes, depending on the user's input or the state of the deck.

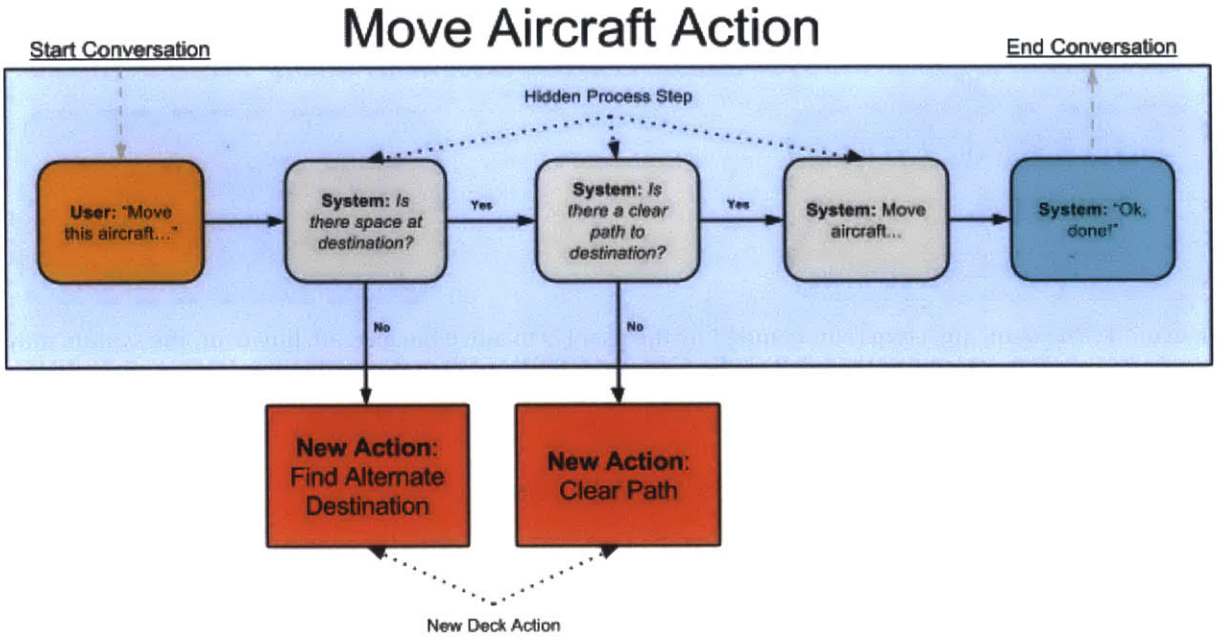


Figure 7-2: Conversation flow for moving an aircraft. We branch into new actions based on the user's input or, in this case, the situation on deck.

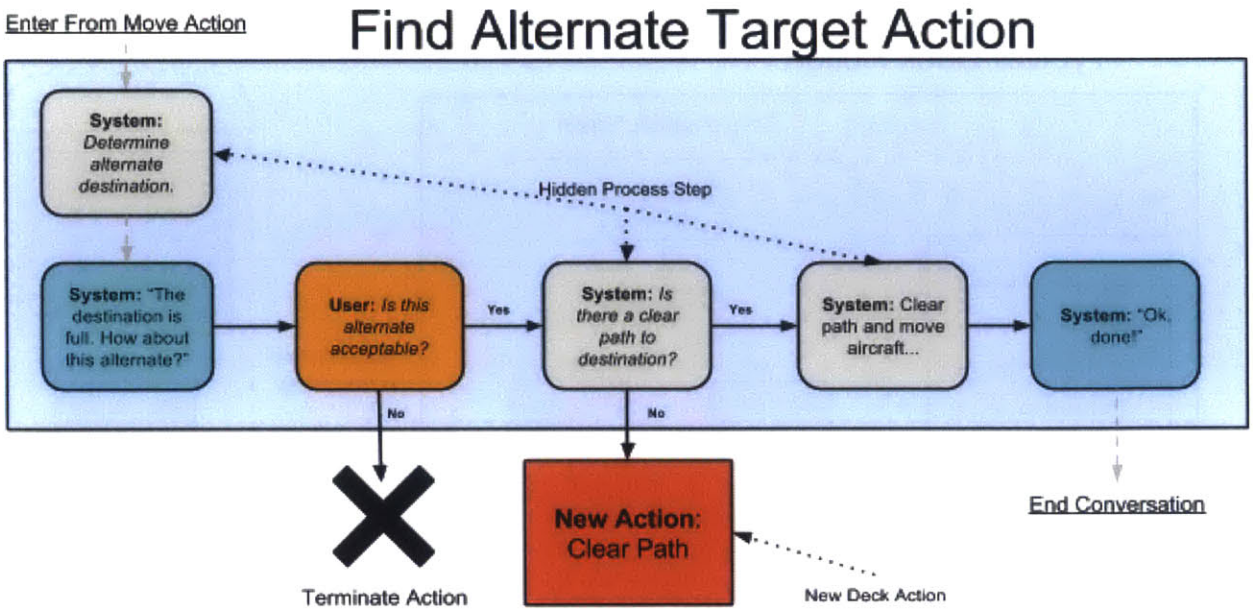


Figure 7-3: Conversation flow for finding an alternate destination. We branch into new actions or terminate the action based on the situation on deck, or in this case, the user's input.

7.2 Implementing the Deck Action Framework

7.2.1 Deck Actions

Figure 7-4 illustrates the key components of Deck Actions. Each Deck Action includes logic for manipulating the deck and a graph-like structure representing the predetermined conversation with the user, known as the Conversation Graph. The graph incorporates user input and output with the logic that makes up the Deck Action. Deck Actions execute by stepping through nodes of their Conversation Graphs. Nodes contain either conversation points for the user, logic for manipulating the deck, or a combination of both. Deck Actions use the multimodal stacks for interaction and make decisions by branching at different points in the graph. At any point, Deck Actions can create additional Deck Actions as necessary to complete the main action.

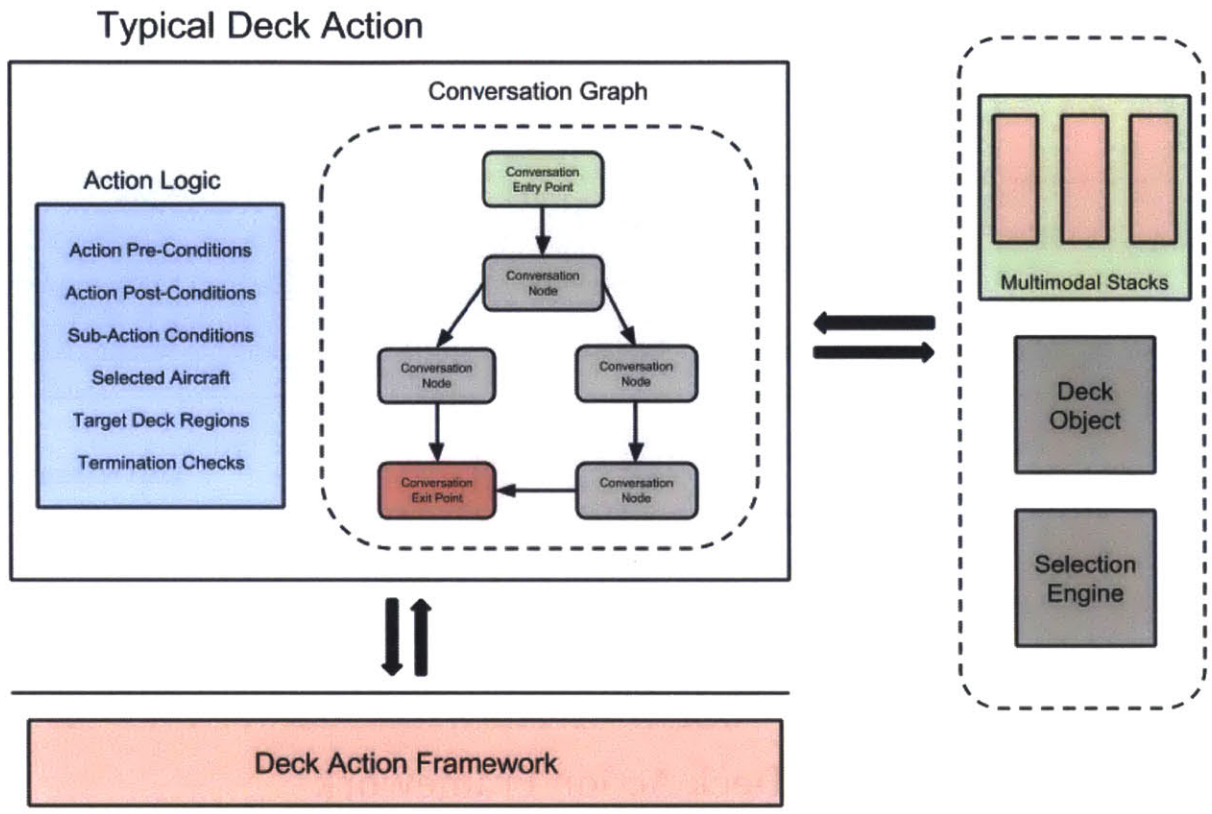


Figure 7-4: Internals for a typical Deck Action. Each Deck Action contains logic and corresponding conversation for the Deck Action. Actions can communicate with other modules in DeckAssistant, and initiate further actions through the Deck Action Framework.

7.2.2 Action Stack

We organize active Deck Actions on a stack-like structure, known as the Action Stack. Execution of Deck Actions on the Action stack is analogous to subroutines on a call stack. At the start of simulation, the Action Stack is empty. User initiated Deck Actions (though commands) are placed on the bottom of the Action Stack. Only the top-most Deck Action on the stack executes any given moment. At any moment, the executing Deck Action can create an additional Deck Action and add it to the top of the stack, pausing its exaction until the stack returns to it. This allows initial Deck Actions to delegate sub-tasks to additional Deck Actions when needed, as shown in Figure 7-5. A Deck Action finishes executing by notifying the Action Stack of its completion. When the top-most Deck Action is complete, the Action Stack pops items off the stack until it encounters an unfinished Deck Action, and resumes

its execution. Lower Deck Actions that paused for higher Deck Actions can make decisions based on results from completed Deck Actions on the stack, allowing decision-making to incorporate the outcomes of sub-actions. After the first user command, new Deck Actions initiated by user commands are queued at the bottom of the stack, allowing the currently executing Deck Actions and its sub actions finish before the system addresses new commands.

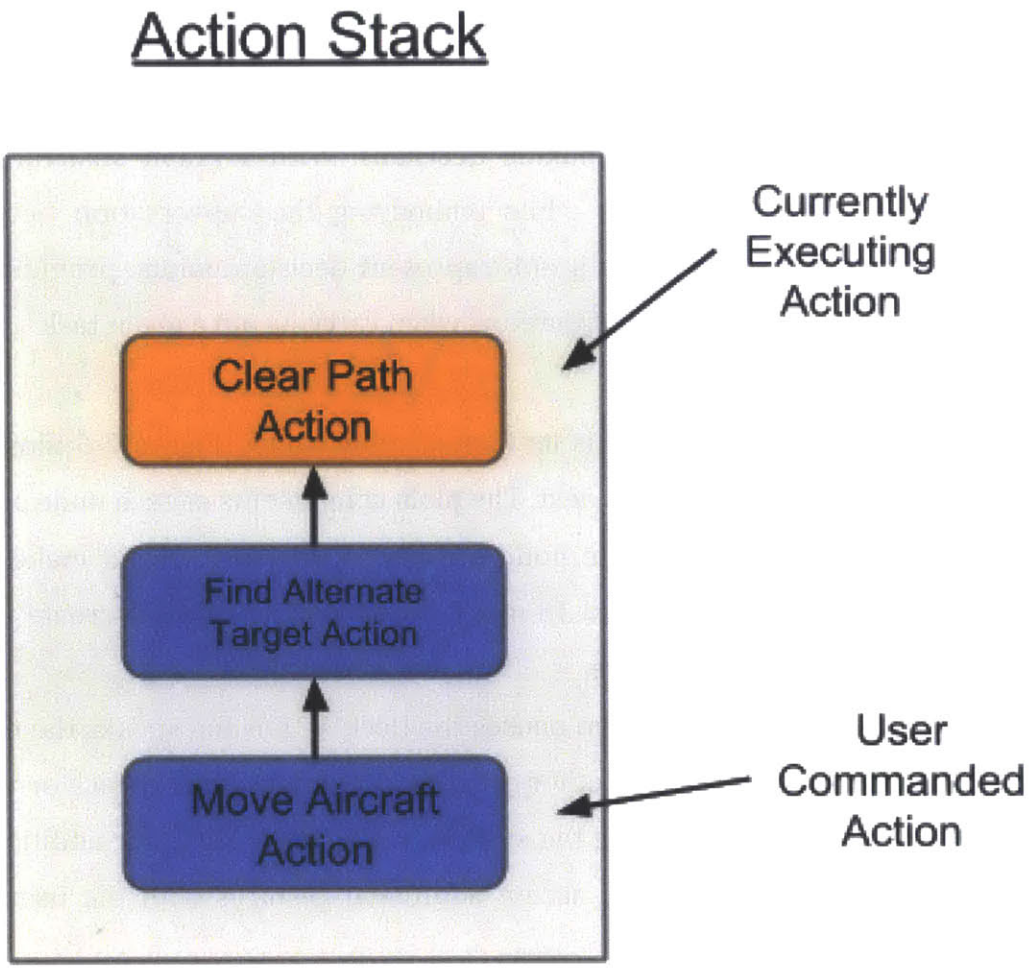


Figure 7-5: Deck Actions build upwards on the stack (similar to subroutines on a call stack). The top most action is the currently executing action. In this example, the Move Aircraft Action creates additional actions to find an alternate destination and clear a path.

7.3 Deck Action Logic and Interactive Conversations

In this section, we give an overview of the role of Conversation Graphs and Conversation Nodes in driving Deck Actions functionality within DeckAssistant. We provide further details on code implementation in the **DECKASSISTANT SOFTWARE GUIDE**. See the Appendix section for more info.

7.3.1 Conversation Graphs

The Conversation Graph embodies the logic and pre-determined interactions for each Deck Action in a graph structure built on nodes. As shown in Figure 7-2 and Figure 7-3 (see section 7.1.2.), each Deck Action requires multiple distinct steps, including processing deck state, interacting with the user, or making decisions. With a graph structure, we can represent these processes explicitly, while embodying the conversation between our system and the user. Branches in the graph represent decision points, providing actions with the flexibility to handle different situations when carrying out a given task.

7.3.2 Conversation Nodes

The makeup of a Conversation Graph is its Conversation Nodes. Figure 7-6 illustrates the flow within each conversation node to next. The main components of each node are:

- A Pre-Speech Process: Here the node can process deck state and make decisions before conversing with the user. In many nodes, we dynamically create the node's text for synthesized speech here.
- Speaking to the user: The system pauses the Deck Action and speaks the node's text to the user. A node chooses whether execution continues after speech or waits for a spoken user response, allowing the system to engage the user for additional input. While waiting, the system can accept additional gestures with the user's spoken response.
- A Post-Speech Process: Here, the node can incorporate user response after spoken text in decision-making.

We implement the flow through the Conversation Graph through each Conversation Node. After the Pre-Speech or Post Speech process, a node signals the next step to the Conversation Graph: either a to jump to the next Conversation Node, an end to the conversation, or the start a new Deck Action which is added to the top of the Action Stack. The Conversation Node contract requires a node to choose at least one of these actions

before it completes, and each node can process deck state or user response before deciding the next step in the conversation (branches in the Conversation Graph). When creating a new Deck Action, a Conversation Node still specifies the next node to jump to or the end of the current Deck Action, continuing the conversation when the Action Stack returns from higher Deck Actions.

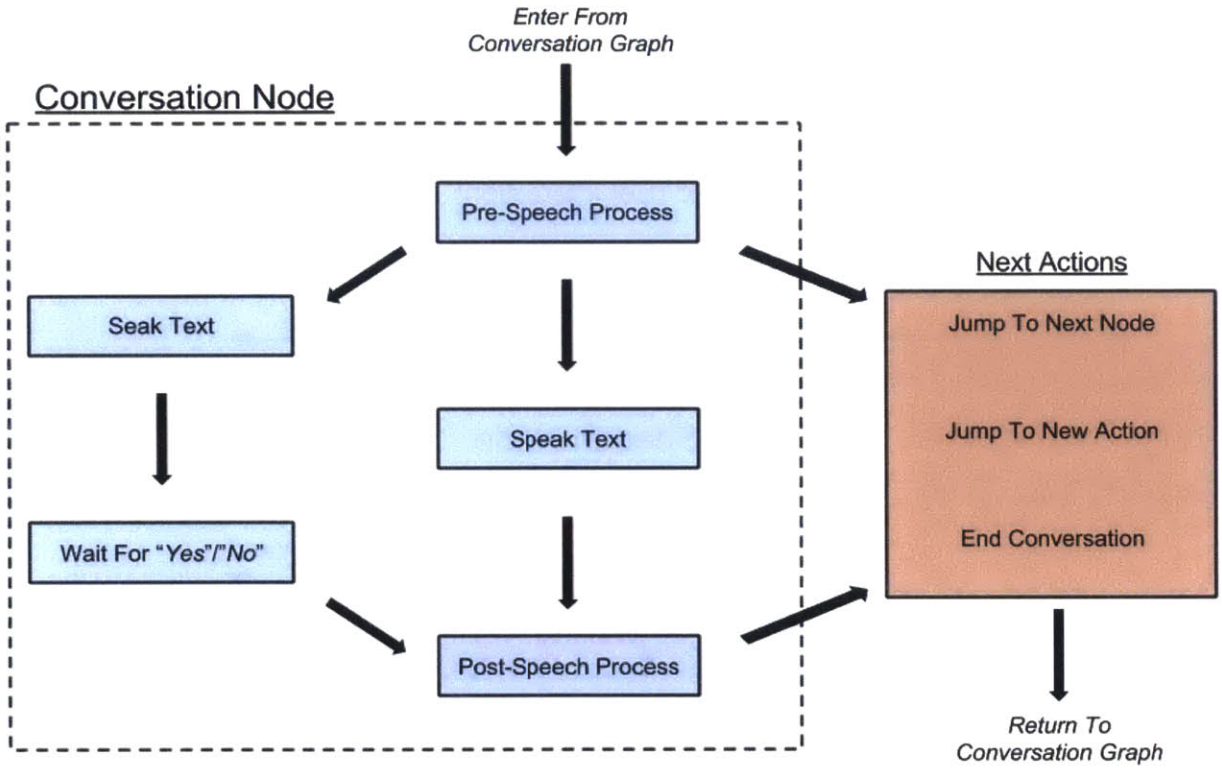


Figure 7-6: Process flow within a Conversation Node.

We create a variety of Conversation Node types based on the implementation of each node (see Figure 7-7). For instance, a node that runs its Pre-Speech Process before choosing the next step in the conversation does not interact with the user. These processing nodes act like hidden points in the conversation. Often, these nodes are used to preform pre-action checks, such as ensuring the user has selected appropriate aircraft or specified a valid destination. The most common nodes in our system speak text to the user. In addition, nodes can start and stop animations or change the deck before or after speech generation. Other nodes incorporate user feedback, which we use to for "Yes"/"No" responses. For instance, after clearing a blocked path or looking for an alternate

destination, the system asks the user if they're ok with the additional actions. Based on the users response, the system commits the results or reverts the deck to its original state. While our examples only demonstrate "Yes"/"No" responses, users can easily add additional predetermined responses to accept in question nodes. Future Deck Actions could leverage a variety of responses complex decisions.

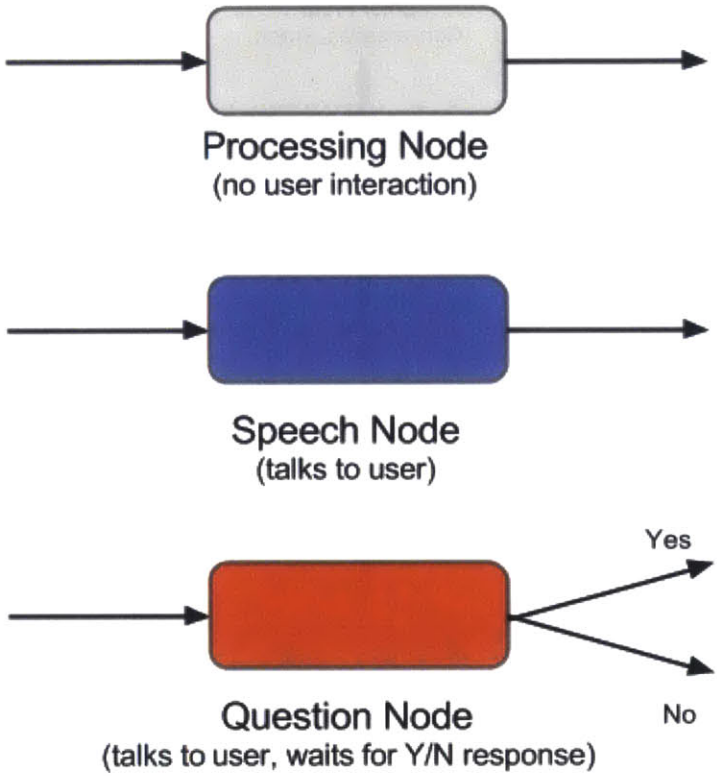


Figure 7-7: 3 main types of conversation nodes in DeckAssistant.

Chapter 8

8 Related Work

In this chapter, we discuss prior work related to or inspiring DeckAssistant.

8.1 Navy ADMACS

As previously mentioned, the Navy is moving towards digitally networked systems to replace current Ouija Board technology. They aim to have ADMACS standard on all carriers in the future.

8.2 Deck Heuristics Action Planner

Ryan et al. [1] developed a deck action heuristics planner with a simulation for aircraft operations on modern aircraft carrier decks. The planner utilizes an Integer Linear Programming (ILP) algorithm for optimizing a set of aircraft states and goals based on predetermined heuristics. Originally, it was developed for aiding deck handlers in operations and later extended to evaluate the use and control of UAVs on aircraft carrier decks. Overall, Ryan et al. cannot replicate the accuracy of human planning using a heuristic planner, but under certain scenarios human planners working in tandem with the automatic planner perform equivalently to human planners without. This is attributed largely to the difficulty in selecting the relevant heuristics in each situation. This planner demonstrates an earlier attempt at tools to aid planning.

8.3 Multimodal Gestures Presentations

Ying Yin et al. [2] developed gesture tracking for distinguishing both fixed and fluid gestures simultaneously using HMMs. These gestures can expand interaction with a variety of digital applications beyond conventional mouse and keyboard input. In this example, the speaker can use their gestures to transition and trigger interactive content on each slide.

This leads to a more natural conversation with a speaker's audience, without needing to stop and use a keyboard, mouse, or wireless controller. Yin et al. also developed an interface for pointing and gesturing over a tabletop surface which I will incorporate into my masters work.

Chapter 9

9 Conclusion

In this work, we presented DeckAssistant, a multimodal digital Ouija Board for planning aircraft carrier deck operations. Unlike traditional static Ouija Boards, DeckAssistant has a basic understanding of aircraft movement and space, and can aid the deck handler in planning. DeckAssistant uses symmetric multimodal interaction, allowing the deck handler to communicate actions with a combination of gestures and speech; the system responds with its own synthesized speech and graphics. The result is a conversation between the deck handler and the system as the two cooperate to accomplish tasks.

In our system, we implemented a novel approach to organizing and combing tasks. We built our system functionality into a set of scripted actions for carrying out specific tasks on deck. These actions are designed to be flexible in handling a variety of situations, with the ability delegate tasks with additional actions. We organize these actions around scripted conversations represented by graphs. These graphs capture both the flow of conversations and the logic for manipulating the deck and making decisions in actions.

We also expand on research by Ying Yin et al. for tracking hands over a horizontal display. This initial system uses depth-sensing technology to segment and identify hands and fingers over a tabletop and creates a model 3D model to interpret pointing gestures from the user. With additional sampling and filtering techniques, we've improved pointing accuracy and responsiveness, allowing users to comfortably point and differentiate objects and locations to our system.

9.1 Future Work

9.1.1 Supporting Additional Deck Operations

The initial iteration of our digital Ouija Board focuses on aircraft movement and placement operations. Future iterations should expand DeckAssistant's functionality to address

additional aspects of planning and simulation on carrier decks. Potential operations to address include the management of aircraft munitions, fuel, damage, and the repair of aircraft on deck. These additional operation require incorporating more detailed status information for each aircraft, giving DeckAssistant more information to present to the user and consider when making decisions during planning.

Many traditional Ouija Boards have two surfaces, a top surface that shows the deck and runways, and bottom surface that represents the carrier's bottom deck and aircraft hangar Adding this view to the interface would allow deck handlers to organize and view aircraft status both above and below deck. Other potential spaces could include the immediate area around the carrier, such as the Marshall Stack (the space behind the carrier where landing aircraft queue and approach the deck).

9.1.2 Improving Finger Tracking and Gesture Recognition

Our tracking method can only recognize outstretched fingers on hands that are held mostly horizontal, flat, and open with respect to the camera. Further improvements to our algorithms could enable our system to follow hands in a greater variety of poses, increasing potential recognizable gestures. For instance, while it is difficult to estimate finger positions occluded by the palm of the hand, there is still the potential for recognizing fingers extended towards the camera (e.g. a hand pointing up) using the depth information provided. Research by Ying Yin et al. demonstrates the ability to recognize complex fluid and static gestures simultaneously. Additional gestures, static or fluid, would give users more ways to express detail in Deck Actions, such as specially descriptive information that would normally be difficult to communicate with speech alone. For example, the deck handler could include illustrate motions or arrangements for aircraft on deck using fluid gestures. With additional gestures, we could add interface customizability to our system, allowing the user to pan and zoom the camera view into particular areas on deck.

We designed our hand tracking methods around the limitations of Kinect V1, introduced in 2010. After five years, Microsoft has made substantial improvements to the Kinect platform with the release of the Kinect V2. This sensor features a higher resolution depth image at twice the frame rate. Our initial experiments with this device have produced substantially cleaner depth images compared to the Kinect V1. With the Kinect

V2, we could track finer finger movement over a larger table surface, distinguishing more gestures.

9.1.3 Using Drawing Gestures

Apart from hand gestures, the ability to draw may prove very useful to our system. The table surface of our digital display is a large drawing digitizer capable of tracking movements of digital styluses above or on its surface. A digital pen could be used for additional input to Deck Actions, such as drawing aircraft movement and placement, or drawing symbols illustrate additional context. We could also create a note taking system, allowing users further illustrate actions or annotate the deck and aircraft. These notes could be particularly useful for both collaboration and keeping a history of important information related to deck operations.

9.1.4 Timeline History, and Review

Deck Actions give our system a way of understanding operations on deck. By keeping track of each action and its results, we could build a history of operations during our system's use. We could give users additional flexibility in planning operations, with the ability to apply then rewind multiple actions. This history could also serve as a valuable record during review of deck operations.

10 References

[1] Jason C. Ryan. *Evaluating Safety Protocols For Manned-Unmanned Environments Through Agent-Based Simulation*. Massachusetts Institute of Technology. PHD Thesis 2014.

[2] Ying Yin. *Real-time Continuous Gesture Recognition For Natural Multimodal Interaction.* Massachusetts Institute of Technology. PHD Thesis 2014.

[3] US Navy Air Systems Command. *Navy Training System Plan for Aviation Data Management and Control System*. US Navy 2002.

[4] Timothy Thate and Adam Michels. *Requirements for Digitized Aircraft Spotting (Ouija) Board for Use on U.S. Navy Aircraft Carriers.* Naval Postgraduate School 2002

[5] Philip Ewing. *Carrier 'Ouija Boards' Go Digital*. NavyTimes 2008. http://archive.navytimes.com/article/20080907/NEWS/809070307/Carrier-8216-Ouija-boards-go-digital

[6] Morphological Image Processing. https://www.cs.auckland.ac.nz/courses/compsci773s1c/lectures/ImageProcessing-html/topic4.htm

11 Appendix

11.1 Demo

A video demonstration of DeckAssistant is available at the project website: http://groups.csail.mit.edu/mug/projects/ouijaboard/

11.2 Additional Documents

Located on the DeckAssistant project website (link above) are additional documents including:

- A code-level, detailed software guide with on formation on running and modifying DeckAssistant.

11.3 Code Location

The entire codebase for DeckAssistant is located on GitHub: https://github.com/MUG-CSAIL/DeckAssistant