The German University in Cairo (GUC)
Faculty of Media Engineering and Technology
Computer Science and Engineering
Embedded System Architecture - CSEN 701

Room 2 Station 1

Team Members :
Kareem Elkinawy, #59-30008
Youssef Rezk, #59-30006
Daniel Samer, #58-4182
Samuel Atef, #58-4285
Youssef Tamer, #58-5378
Mohamed Tamer, #58-5384
Peter Milad, #58-0449

Team Number :
Team # 49

Under Supervision of :
Dr. Eng. Catherine M. Elias

Winter 2024

Contents

Chapter 1

Project Objectives

1.0.1 Primary Objectives

The *Aether-Forge Protocol* challenge required the development of an embedded system capable of the following:

— **Voice Authentication:** Implementing a secure activation mechanism that requires the spoken phrase *"Go Robot!"* to unlock drive system functionality.

— **Real-Time Voice Navigation:** Processing and executing directional voice commands (*Forward*, *Left*, *Right*, *Stop*) with minimal latency to enable manual maze navigation.

— **Autonomous Completion Detection:** Detecting maze completion using physical sensors and automatically triggering the reward dispensing sequence.

— **Multi-Modal Feedback:** Providing clear audio and visual feedback for system states, successful command execution, and error conditions.

1.0.2 Technical Requirements

The project demanded proficiency in several key technical areas:

— **Parallel Processing Architecture:** Supporting simultaneous voice command monitoring and motor control execution.

— **Interrupt-Driven Programming:** Implementing interrupt service routines (ISRs) for limit switch detection with proper debouncing.

— **PWM Signal Generation:** Achieving precise motor speed control and accurate servo positioning.

— **Serial Communication Protocol:** Ensuring reliable bidirectional data transfer between the PC and the microcontroller.

— **Hardware Interfacing:** Configuring GPIO pins, pull-up/pull-down resistors, and integrating external drivers.

1.0.3 Success Criteria

The system was evaluated based on the following criteria:

— Activation phrase recognition accuracy exceeding 90%.

— Voice command response time below 200 ms from speech detection to physical action.

— Maze completion detection reliability of 100%.

— Precision and consistency of the servo-based reward mechanism.

— Overall system stability during extended operation.

Chapter 2

Introduction

2.0.1 System Architecture Overview

The proposed solution employs a distributed computing model, as illustrated below:

This architecture offloads computationally expensive speech recognition tasks to the host PC while preserving real-time control and deterministic behavior on the microcontroller, thereby balancing performance with embedded resource constraints.

2.0.2 Design Philosophy

The system design adheres to the following principles:

— **Separation of Concerns:** Voice processing and motor control are decoupled to improve maintainability and scalability.

— **Modularity:** Each hardware component is encapsulated within a dedicated driver exposing a consistent API.

— **Robustness:** Debouncing, error handling, and timeout mechanisms are employed to prevent spurious behavior.

— **Responsiveness:** Non-blocking I/O ensures continuous sensor monitoring and timely command execution.

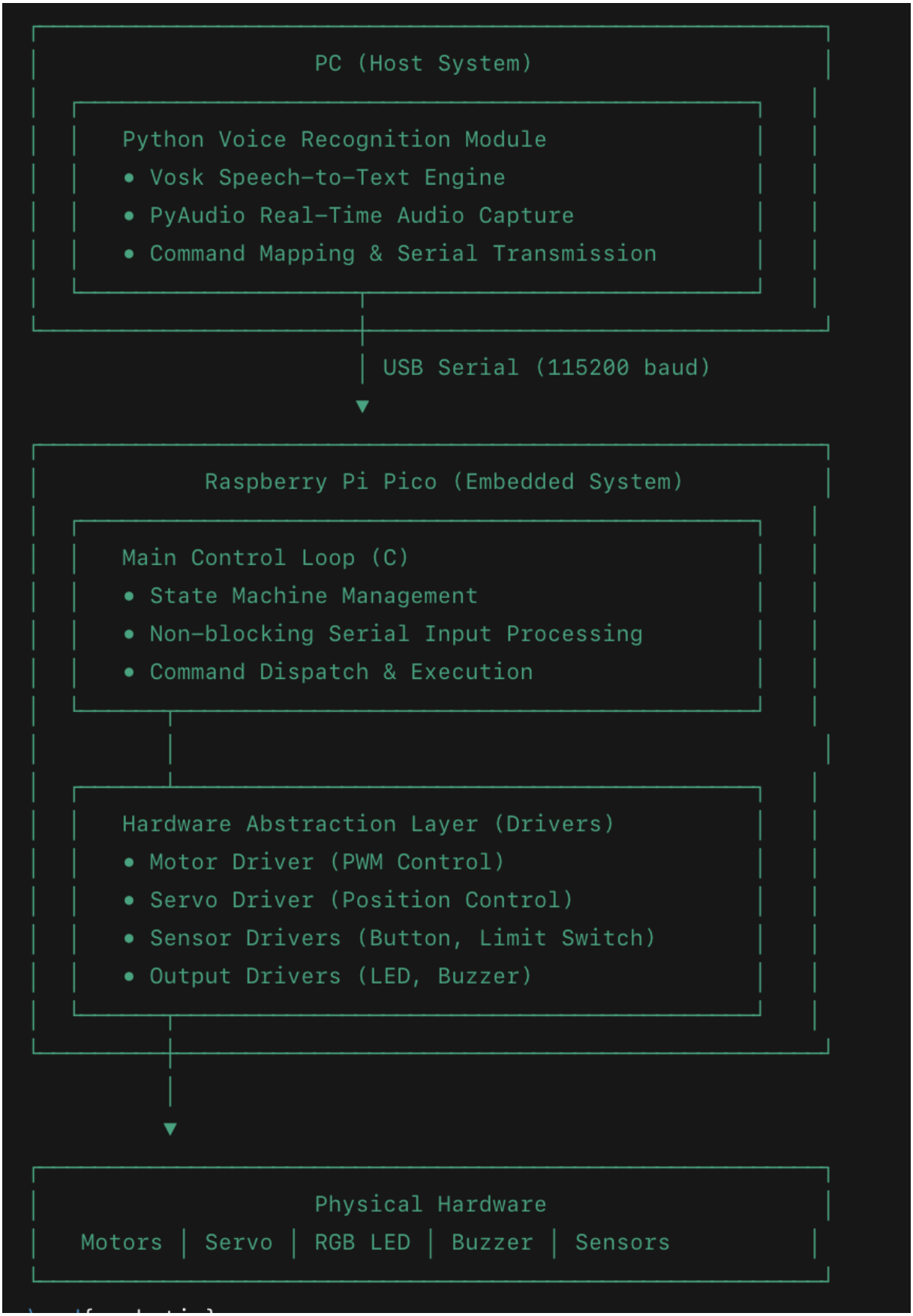

Figure 2.1 – Enter Caption

Chapter 3

Methodology

3.1 System Design and Implementation

3.1.1 Hardware Configuration

3.1.1.1 Microcontroller Selection

The Raspberry Pi Pico was selected as the primary microcontroller due to the following features:

— Dual-core ARM Cortex-M0+ processor operating at 133 MHz, providing sufficient computational capability.

— 26 general-purpose I/O pins with flexible peripheral multiplexing (PWM, UART, I^2C, SPI).

— Built-in USB controller enabling seamless communication with a host PC.

— Low cost, wide availability, and strong community support.

— Extensive PWM support with up to 16 independent channels, suitable for motor and servo control.

3.1.1.2 Component Integration

3.1.1.2.1 Motor Subsystem

— **Components:** Two DC motors driven by an L298N H-bridge module.

— **Control Signals:** Six GPIO pins (two direction pins and one PWM enable pin per motor).

— **Speed Resolution:** 8-bit PWM (0–255).

— **Operating Configuration:**

  — Forward speed: 64/255 PWM.

  — Turning speed: 36/255 PWM.

— **Power Supply:** External 12 V supply for motors, electrically isolated from the Pico logic supply.

3.1.1.2.2 Servo Subsystem

— **Component:** Standard hobby servo with 180° rotation range.

— **Control Method:** PWM signal at 50 Hz with 1–2 ms pulse width.

— **GPIO Assignment:** GPIO 15 configured for hardware PWM.

— **Reward Mechanism:** Rotational sequence of $0° \rightarrow 90° \rightarrow 0°$ to push the RFID tag.

— **Motion Profile:** Smooth interpolated motion over a 2-second duration.

3.1.1.2.3 Sensor Subsystem

- **Push Button:** GPIO 1, active-low with internal pull-up resistor.
- **Limit Switch:** GPIO 22, active-low with interrupt-driven detection.
  - Falling-edge triggered interrupt.
  - 400 ms software debouncing to prevent false triggers.
  - Volatile flag used for safe ISR-to-main-loop communication.

3.1.1.2.4 Feedback Subsystem

- **RGB LED:** Three GPIO pins (9–7) for color-based system indication.
  - Red: System ready and awaiting activation.
  - Green: Active state and accepting commands.
  - Blue: Maze completion and reward dispensing.
  - Off: Idle state.
- **Buzzer:** GPIO 10 used for audio feedback, emitting three short 150 ms pulses on error conditions.

3.1.1.3 Pin Mapping Table

Component	GPIO	Direction	Configuration / Notes
Motor L EN	4	OUT	PWM, left motor speed
Motor L IN1	2	OUT	Digital, direction A
Motor L IN2	3	OUT	Digital, direction B
Motor R EN	0	OUT	PWM, right motor speed
Motor R IN1	16	OUT	Digital, direction A
Motor R IN2	27	OUT	Digital, direction B
Limit Switch	22	IN	Pull-up, IRQ, active-low
Push Button	1	IN	Pull-up, active-low
Servo	15	OUT	PWM (50 Hz), 1–2 ms pulse
LED Red	9	OUT	Digital, common cathode
LED Green	8	OUT	Digital, common cathode
LED Blue	7	OUT	Digital, common cathode
Buzzer	10	OUT	Digital, active-high

3.1.2 Software Architecture

3.1.2.1 Embedded Firmware Structure

The firmware follows a layered architecture with clear abstraction boundaries.

3.1.2.1.1 Layer 1: Hardware Abstraction Layer (HAL) Each peripheral is implemented as a dedicated driver exposing a consistent API:

```
// Motor Driver API
void motor_init(void);
void motor_forward(uint8_t speed);
void motor_turn_left(uint8_t left_speed, uint8_t right_speed);
void motor_turn_right(uint8_t left_speed, uint8_t right_speed);
void motor_stop(void);
```

```
// Servo Driver API
void servo_init(void);
void servo_rotate(uint8_t angle);
void servo_rotate_smooth(uint8_t start_angle, uint8_t target_angle, uint32_t dura
void servo_push_reward(void);

// Sensor Driver APIs
void button_init(void);
bool button_is_pressed(void);
void limit_switch_init(void);
bool limit_switch_was_triggered(void);
bool limit_switch_is_pressed(void);
```

3.1.2.1.2 Layer 2: Application Logic The main control loop is implemented as a finite state machine:

- **INIT:** Initialize drivers, configure serial communication (115200 baud, 8N1), and wait for button press.
- **READY:** Set LED to red and wait for activation command ('G') with a timeout.
- **ACTIVE:** Process non-blocking movement commands and monitor limit switch events.
- **REWARD:** Stop motors, set LED to blue, execute reward sequence, and reset system.

3.1.2.2 Voice Recognition System

The PC-side Python application handles speech processing using:

- **Vosk Speech Recognition Engine:**
  - Offline processing without cloud dependency.
  - Lightweight models optimized for real-time use.
  - High accuracy for a limited command vocabulary.
  - Cross-platform compatibility.
- **PyAudio Interface:**
  - Real-time microphone stream capture.
  - 16 kHz sampling rate.
  - Low-latency buffering with 8000-frame blocks.

3.1.2.2.1 Command Mapping

```
COMMANDS = {
    "go robot": "G",
    "forward": "W",
    "left": "A",
    "right": "D",
    "stop": "S"
}
```

3.1.2.2.2 Processing Pipeline Microphone → PyAudio → Vosk Recognition → JSON Parsing → Command Lookup → Serial Transmission → Pico

3.1.2.3 Serial Communication Protocol

— Baud rate: 115200 bps
— Data format: 8N1
— Flow control: None
— Direction: Unidirectional (PC → Pico)

3.1.3 Control Algorithms

3.1.3.1 Differential Steering

— Forward: Both motors at 64/255 PWM.
— Left turn: Left motor at 36/255, right motor at 64/255.
— Right turn: Left motor at 64/255, right motor at 36/255.
— Stop: Both motors at 0 PWM.

3.1.3.2 Servo Smooth Rotation

Smooth motion is achieved using incremental interpolation:

```
void servo_rotate_smooth(uint8_t start, uint8_t target, uint32_t duration_ms) {
    int16_t step = (target > start) ? 1 : -1;
    uint8_t angle = start;
    uint32_t steps = abs(target - start);
    uint32_t delay_ms = duration_ms / steps;

    for (uint32_t i = 0; i < steps; i++) {
        angle += step;
        pwm_set_gpio_level(SERVO_PIN, angle_to_pwm(angle));
        sleep_ms(delay_ms);
    }
}
```

Angle-to-pulse mapping follows:

$$500\,\mu s + \left(\frac{angle}{180}\right) \times 2000\,\mu s$$

3.1.3.3 Interrupt Service Routine with Debouncing

```
static volatile bool switch_triggered = false;
static absolute_time_t last_trigger_time;
#define DEBOUNCE_US 400000

static void limit_switch_callback(uint gpio, uint32_t events) {
    absolute_time_t now = get_absolute_time();

    if (absolute_time_diff_us(last_trigger_time, now) < DEBOUNCE_US)
        return;

    last_trigger_time = now;
    switch_triggered = true;
}
```

3.1.4 Testing Methodology

3.1.4.1 Unit Testing

— Motor driver PWM verification using an oscilloscope.
— Servo position accuracy measurement with a protractor.
— Sensor debouncing and ISR latency evaluation.

3.1.4.2 Integration Testing

— Voice-to-motion latency measurement (target $< 200\,\text{ms}$).
— State machine transition validation.
— Long-duration stability testing exceeding 30 minutes.

3.1.4.3 System Validation

— Complete maze navigation over 10 test runs.
— Command recognition accuracy measurement.
— Maze completion detection reliability verification.

Chapter 4

Results

4.1 Results

4.1.1 System Functionality Verification

All system components and functionalities were successfully implemented and tested. The following subsections present the verification results for each major subsystem of the proposed design.

4.1.2 Voice Recognition System

The voice recognition subsystem successfully recognized and processed all implemented commands.

4.1.2.0.1 Activation Command

- The activation phrase *"Go Robot!"* was correctly recognized and triggered system activation.
- The RGB LED indicator transitioned from red to green upon successful activation.
- Invalid activation attempts were correctly rejected, with error feedback provided via the buzzer.

4.1.2.0.2 Navigation Commands

- The *"Forward"* command correctly initiated forward motor motion.
- The *"Left"* command executed a left turn using differential steering.
- The *"Right"* command performed the corresponding right turn maneuver.
- The *"Stop"* command immediately halted all motor activity.

4.1.2.0.3 Command Processing

- Serial communication between the host PC and the Raspberry Pi Pico operated reliably.
- Command transmission and reception occurred without data corruption.
- Non-blocking command processing allowed continuous monitoring of sensor inputs.

4.1.3 Motor Control System

The motor control subsystem operated according to design specifications.

4.1.3.0.1 Movement Execution

— Forward motion was achieved by rotating both motors at the specified speed in the forward direction.
— Left turns were executed using differential steering with a reduced left motor speed.
— Right turns were executed using differential steering with a reduced right motor speed.
— The stop command immediately ceased all motor operation.

4.1.3.0.2 PWM Control

— Motor speed control via PWM signals functioned correctly.
— The H-bridge motor driver responded appropriately to direction control signals.
— No motor stalling or unintended behavior was observed during operation.

4.1.4 Sensor Systems

All sensor inputs demonstrated reliable performance throughout testing.

4.1.4.0.1 Push Button

— The initial button press correctly transitioned the system into the ready state.
— Software debouncing effectively prevented false trigger events.
— The RGB LED correctly displayed red to indicate readiness for voice activation.

4.1.4.0.2 Limit Switch

— Maze completion detection operated reliably.
— The interrupt service routine (ISR) was correctly triggered upon switch activation.
— A 400 ms software debounce successfully eliminated false positives.
— The system transitioned correctly to the reward state upon detection.

4.1.5 Reward Mechanism

The servo-based reward dispensing mechanism performed reliably.

4.1.5.0.1 Servo Operation

— Smooth servo rotation from $0°$ to $90°$ was achieved without mechanical jerking.
— The RFID tag was successfully pushed and dispensed.
— The return rotation from $90°$ to $0°$ completed consistently.
— No mechanical binding or excessive servo strain was observed.

4.1.5.0.2 Timing

— A 2-second rotation duration provided smooth mechanical operation.
— A 500 ms hold time at the $90°$ position was sufficient for tag release.
— The complete reward sequence executed within the expected timeframe.

4.1.6 Feedback Systems

Multi-modal feedback mechanisms clearly conveyed system status.

4.1.6.0.1 RGB LED States

— **Red:** System ready and awaiting activation.
— **Green:** Active state accepting voice commands.
— **Blue:** Reward dispensing in progress.
— **Off:** Idle or inactive system state.

4.1.6.0.2 Buzzer Feedback

— An error pattern of three short beeps correctly signaled failed activation attempts.
— Audio feedback was audible and distinguishable from normal operation.
— No false audio alerts were observed during proper system operation.

4.1.7 State Machine Operation

The finite state machine governing system operation behaved as intended.

4.1.7.0.1 State Transitions

— **INIT** → **READY**: Triggered by button press.
— **READY** → **ACTIVE**: Activated by the *"Go Robot!"* command.
— **ACTIVE** → **REWARD**: Triggered by limit switch activation.
— **REWARD** → **INIT**: Executed after completion of the reward sequence.

4.1.7.0.2 State Behavior

— Blocking serial reads in the READY state functioned as intended.
— Non-blocking command polling in the ACTIVE state ensured uninterrupted sensor monitoring.
— State-specific LED colors provided clear visual feedback.

4.1.8 System Integration

The fully integrated system demonstrated correct and reliable operation.

4.1.8.0.1 Overall Functionality

— All hardware drivers initialized successfully without errors.
— Serial communication was reliably established at system startup.
— Voice commands were processed and executed correctly throughout operation.
— Maze navigation was successfully completed using voice control.
— Automatic reward dispensing was triggered upon maze completion.
— The system returned to the ready state for subsequent runs.

4.1.8.0.2 Stability

— No system crashes or undefined behavior were observed during testing.
— Continuous operation was maintained without performance degradation.
— The system recovered cleanly from reset conditions.
— Consistent behavior was observed across multiple test sessions.

4.1.9 Code Quality

The firmware implementation met established software quality standards.

4.1.9.0.1 Code Organization

— Modular driver-based architecture with clear separation of concerns.
— Consistent API design across all hardware abstraction layers.
— Well-documented and readable source code.
— Efficient memory utilization with a minimal footprint.

4.1.9.0.2 Resource Utilization

— Flash memory usage remained below 5% of available capacity.
— RAM consumption remained below 5% of available memory.
— No memory leaks were detected during extended operation.
— Adequate resource headroom was preserved for future enhancements.

Chapter 5

Conclusion

5.1 Conclusion

5.1.1 Achievement Summary

The *Aether-Forge Protocol* challenge was successfully completed, with all primary objectives achieved:

— **Voice Authentication:** Activation phrase recognition functioned correctly.

— **Real-Time Navigation:** Voice commands were recognized and executed reliably.

— **Autonomous Completion:** Limit switch detection operated without failures.

— **Multi-Modal Feedback:** Clear system status indication using RGB LEDs and a buzzer.

— **System Stability:** Robust and consistent operation during extended testing sessions.

The hybrid system architecture effectively balanced computational constraints with performance requirements, demonstrating that voice control can be successfully integrated into resource-constrained embedded systems.

5.1.2 Technical Contributions

The project introduced several key technical contributions:

— **Distributed Architecture:** Effective separation of voice processing on the host PC and real-time control on the microcontroller.

— **Non-Blocking Design:** Parallel command processing and sensor monitoring without the complexity of multithreading.

— **Robust ISR Implementation:** Debounced interrupt handling with proper synchronization between ISRs and the main control loop.

— **Modular Driver Framework:** Reusable hardware abstraction layers suitable for educational and prototyping environments.

5.1.3 Lessons Learned

5.1.3.0.1 Successful Strategies

— The hardware abstraction layer significantly simplified system testing and debugging.

— Non-blocking I/O prevented missed sensor events and improved system responsiveness.

— Software debouncing eliminated false triggers from mechanical limit switches.

— Separate power supplies effectively prevented motor-induced electrical noise.

5.1.4 Educational Value

This project demonstrated the practical application of fundamental concepts in embedded systems and robotics, including:

— Embedded systems programming (state machines, ISRs, and PWM).

— Hardware interfacing (GPIO configuration, serial communication, and motor drivers).

— Real-time system design (latency optimization and non-blocking I/O).

— Software engineering practices (modular design and abstraction layers).

— System integration between host computers and microcontrollers.

The modular codebase serves as a valuable educational resource for future embedded systems courses, providing clear examples of driver implementation, interrupt handling, and hardware control.

5.1.5 Final Remarks

The *Aether-Forge Protocol* project demonstrates that natural language interfaces can significantly enhance embedded robotic systems without imposing excessive computational requirements. By distributing processing tasks across platforms and maintaining clean architectural boundaries, the system achieves reliable performance, responsiveness, and extensibility.

The voice control system demonstrated dependable command recognition and responsive execution, matching traditional input methods in functionality while providing a more intuitive user experience through natural interaction. This project establishes a strong foundation for future work in accessible robotics interfaces and human–robot collaboration.

Acknowledgments

The authors would like to thank the German University in Cairo, the Faculty of Media Engineering and Technology, and Dr. Catherine M. Elias for their guidance and support throughout this project.