Use Case

With our system, it is possible to play card games over the internet with physical cards.

Use Case Requirements

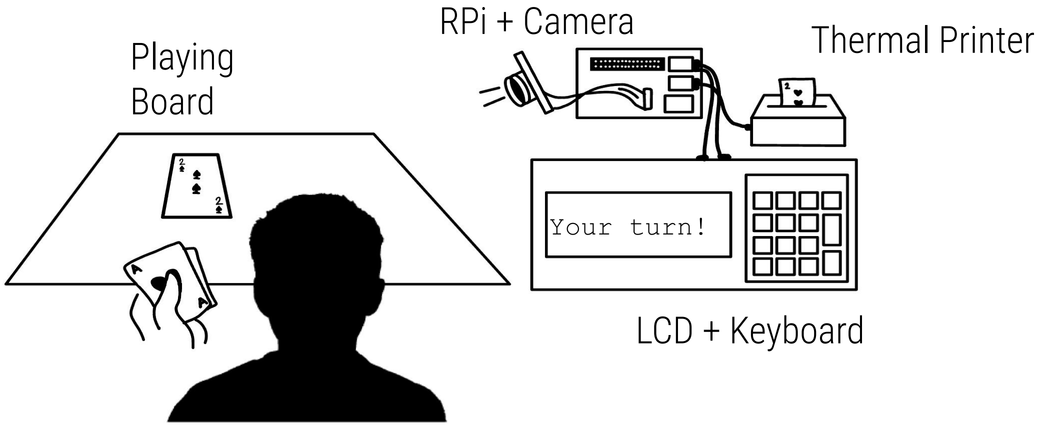
- Play with physical cards
- Plays the games such as Go Fish, Euchre, and Rummy
- Multiplayer support up to 5 players per game
- Be able to input any card for game logic
- Ability to have concurrent games

Quantitative Design Requirements

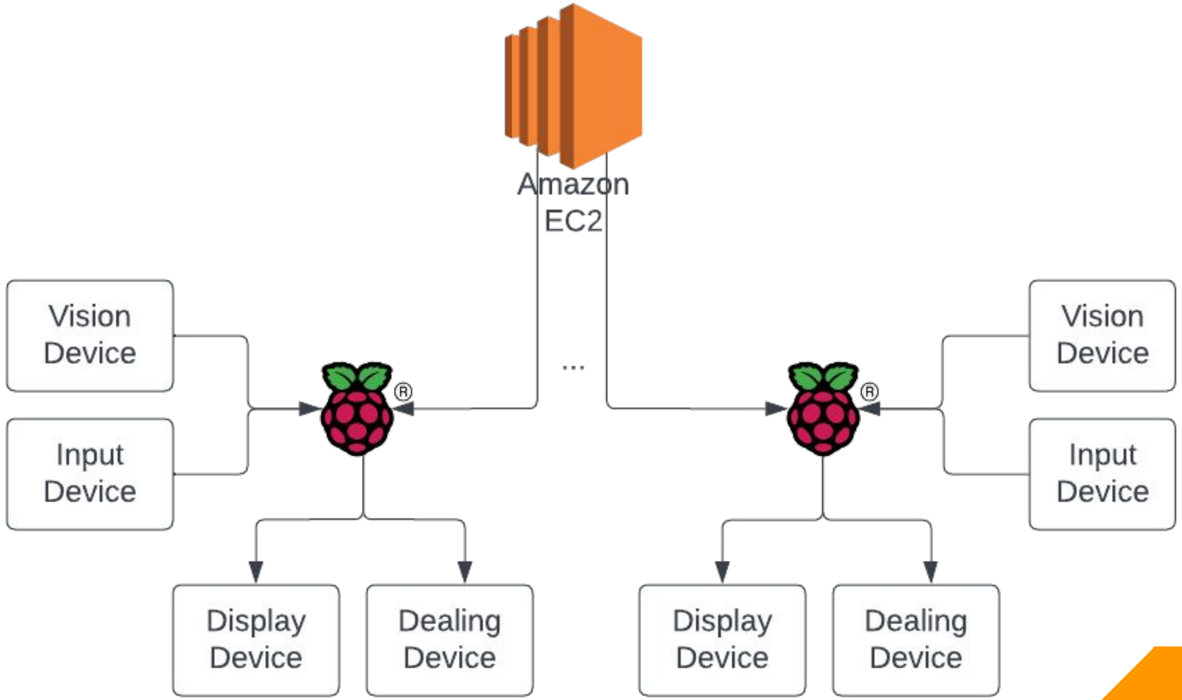
- A 18” x 24” playing/vision area
- Playing/vision area updates are done at least once per second
- When dealing cards are emitted at least once every 2 seconds
- The full physical device is smaller than a shoebox (14 in x 10 in x 5 in) and lighter than 10lbs

Solution Approach

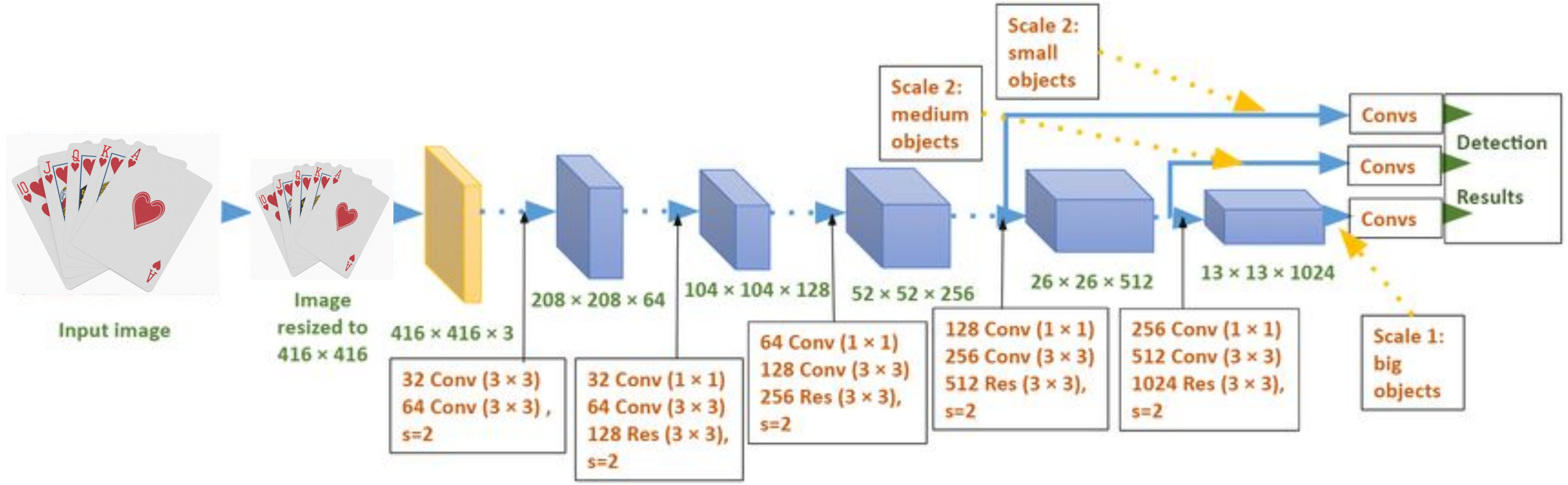
- Vision Device
  - Hardware: Raspberry Pi Camera Module
  - Software: Open CV, Tensorflow, YOLO algorithm
- Input Device
  - 10 key keyboard
- Output Device
  - 40x4 character LCD screen
  - Thermal Receipt Printer



System Block Diagram



YOLO Architecture



Device Block Diagram

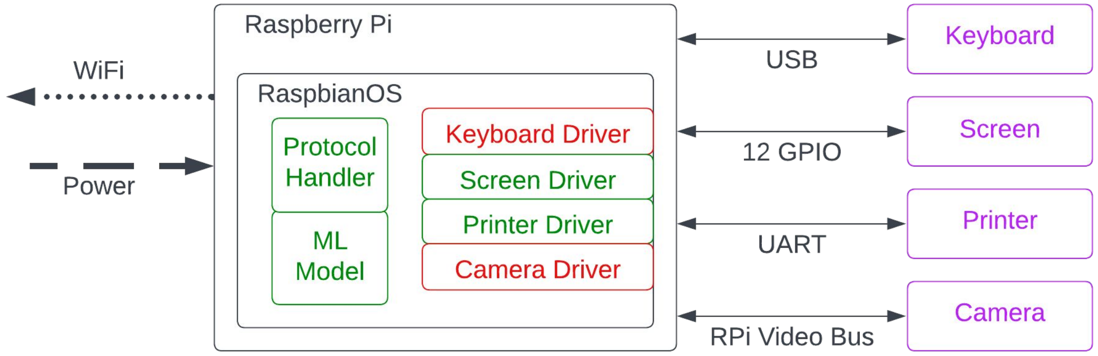


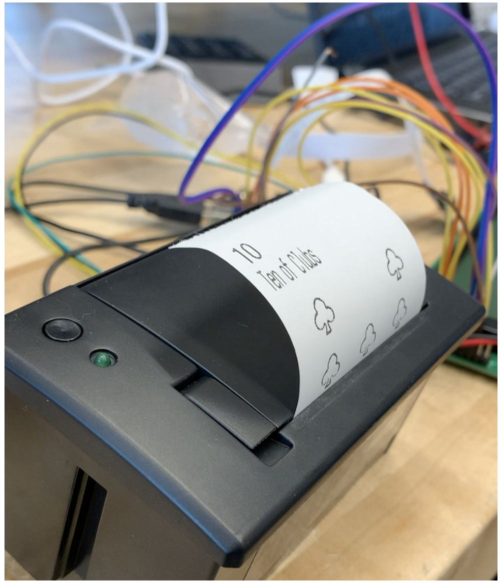
Diagram Key

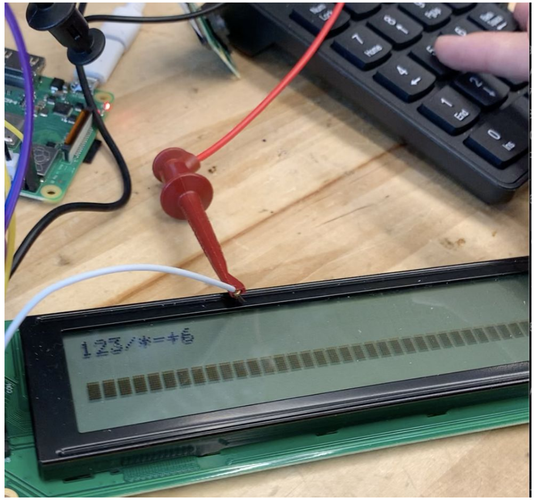
- 🟢 Custom Software
- 🔴 Raspbian Software
- 🟣 Hardware Peripheral

Implementation Plan

Device	Purpose	Implementation Actions/Software
Thermal Printer	Card dealing device	Custom driver for their TTL interface using RPi’s TX/RX pins
Raspberry Pi Camera Module	CV/scan cards	Picamera2 library, YOLO for object detection, implemented in TensorFlow
LCD Screen	Game state display	Custom driver for their custom protocol using GPIO pins
Keyboard	Bets/card requests	RaspbianOS keyboard driver

Complete Solution





Keyboard and LCD Screen

Thermal printing the cards

Unit Testing, Verification, Validation

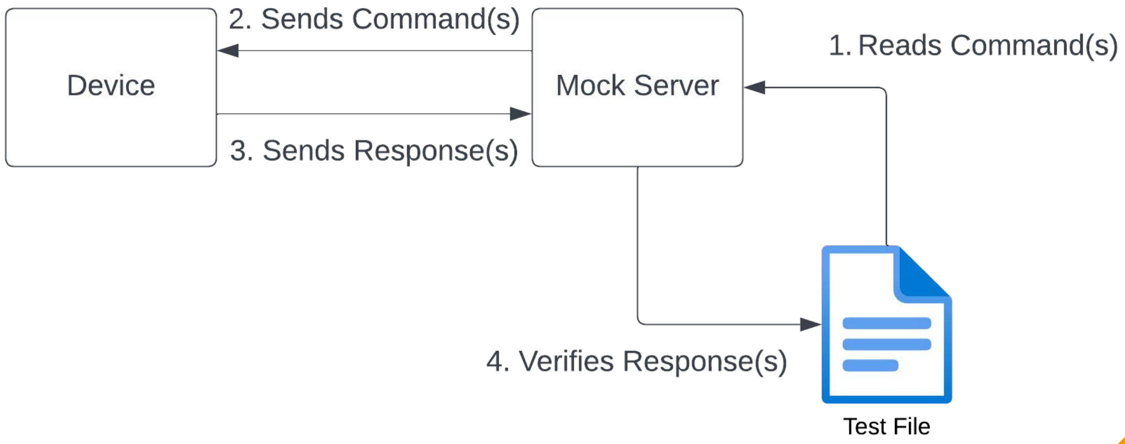
1. *Thermal printer:* Be able to print 3.25” x 2.25” cards with corresponding suit and number in a maximum of 1.5 seconds
2. *Camera/Computer Vision:* Properly identifies card(s) in <35 ms
3. *Small keyboard:* Inputs are properly received and buffered in <10 ms.
4. *LCD Screen:* Displays text, then special characters like suits in <1 ms
5. *Implementing game logic for different games:* Go fish, Euchre, Rummy
6. *EC2/Networking:* Concurrency and logic tests.

Integration Testing, Verification, Validation

1. *Software device-level supervisor:* Services interrupt from peripherals in a timely manner without dropping any signals.
2. *Keyboard/Screen Coupling:* Keypresses appear on screen within our latency targets.
3. *Server/device Network Protocol*: The device supervisor is able to send game state update messages to the server in a timely manner, and the server can reconstruct a matching local game state. The reverse is also true, the server can send commands to the device, which are serviced in a timely manner.

Testing through Mocking

Performance testing the device through writing a program that mocks the game server.



Testing through Mocking

Test File	**Purpose**
numpad_allkeys.test	Checks all keys
print_10_clubs.test	Prints the 10 of clubs
print_king_hearts.test	Prints the king of hearts
print_multiple_cards.test	Prints multiple cards in rapid succession
print_then_detect.test	Prints a card, then detects it
print_then_detect2.test	Prints two cards, then detects them
screen_basic.test	Writes “Hello World” to the lcd screen
screen_suits.test	Writes the suits to the lcd screen

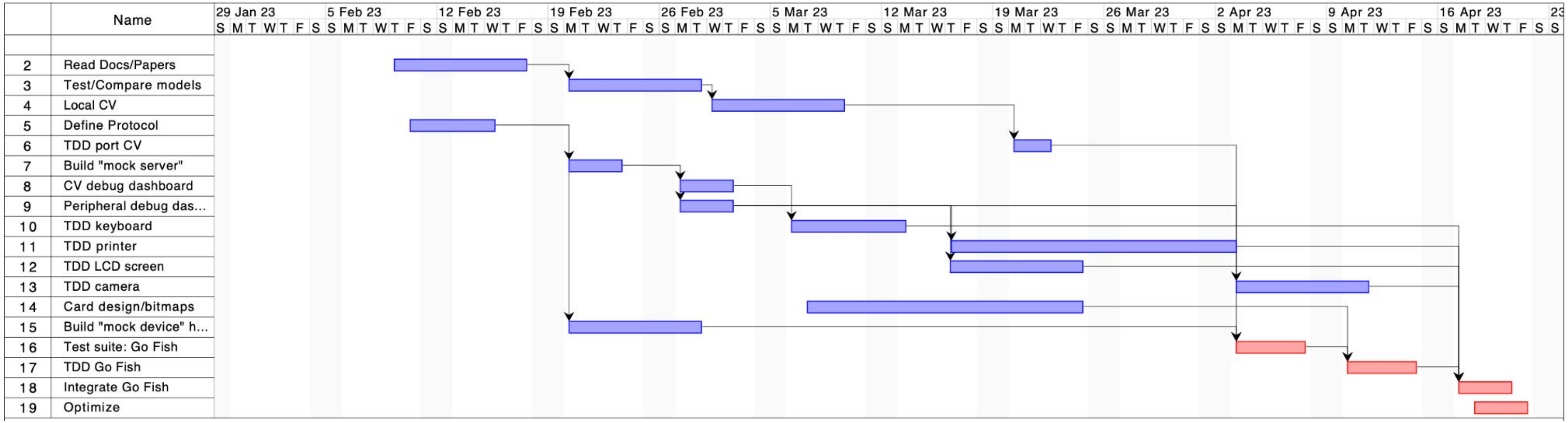
Performance

Performance Metric	Goal	Measured
Card printing speed	1.5 sec	4-7 sec
Keyboard/LCD latency	No visual lag (40ms)	No visual lag
Detection latency	35ms	~23ms
Detection accuracy	85%	97%

Trade-offs

- Single Card Detection vs Multi Card Detection:
  - Single: Faster to train, higher accuracy, would not align with the user requirements of our game
  - Multi Card: Takes a lot more data, more time to train, more epochs and batches to achieve high accuracy, aligns with the game requirements as originally planned
- Card designs vs. Printing speed
  - More authentic playing experience tradeoff with the time it takes to print full bitmaps

Project Management- Updated Schedule



	Name
2	Read Docs/Papers
3	Test/Compare models
4	Local CV
5	Define Protocol
6	TDD port CV
7	Build "mock server"
8	CV debug dashboard
9	Peripheral debug das...
10	TDD keyboard
11	TDD printer
12	TDD LCD screen
13	TDD camera
14	Card design/bitmaps
15	Build "mock device" h...
16	Test suite: Go Fish
17	TDD Go Fish
18	Integrate Go Fish
19	Optimize

Division of Labor:

ML Track: Rachel
Hardware Track: Mason & Miya
Software Track: Mason & Miya (& Rachel)

Lessons Learned

- Complications with the ordered parts (camera module)
- Working with new programming languages
- Getting the individual components to work in order to finish integration