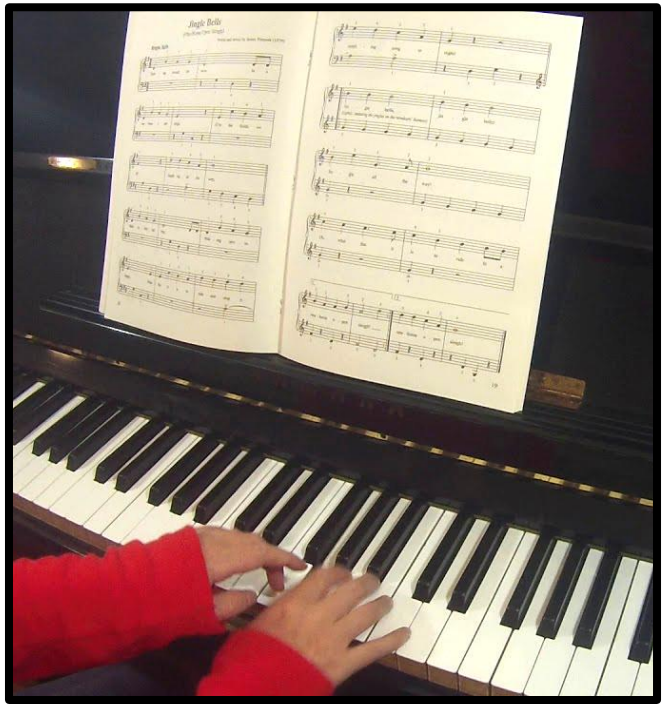
Sheet Vision

Members: Ricardo Faure, Bryan Fung, Garrett Greenfield, Trevin Nance, Walter Svenddal

Faculty Advisor: Alexander Stoytchev

Project plan

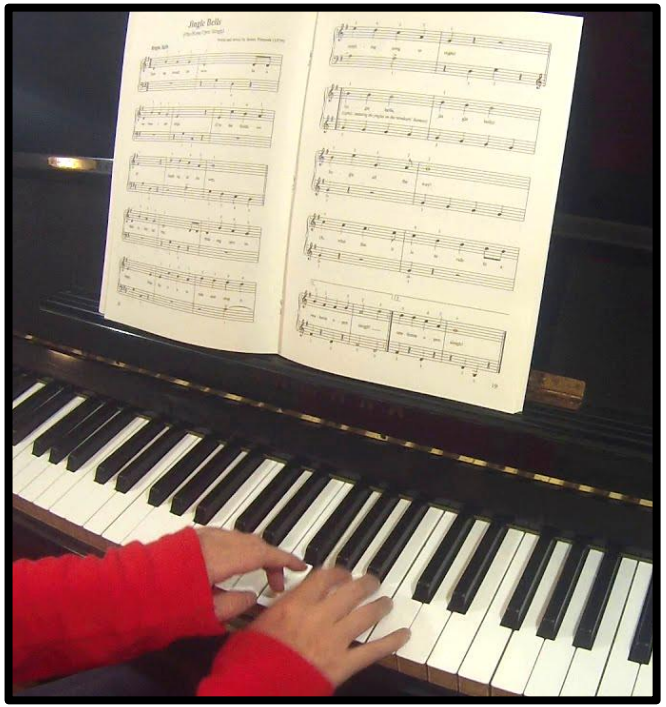
Problem Statement



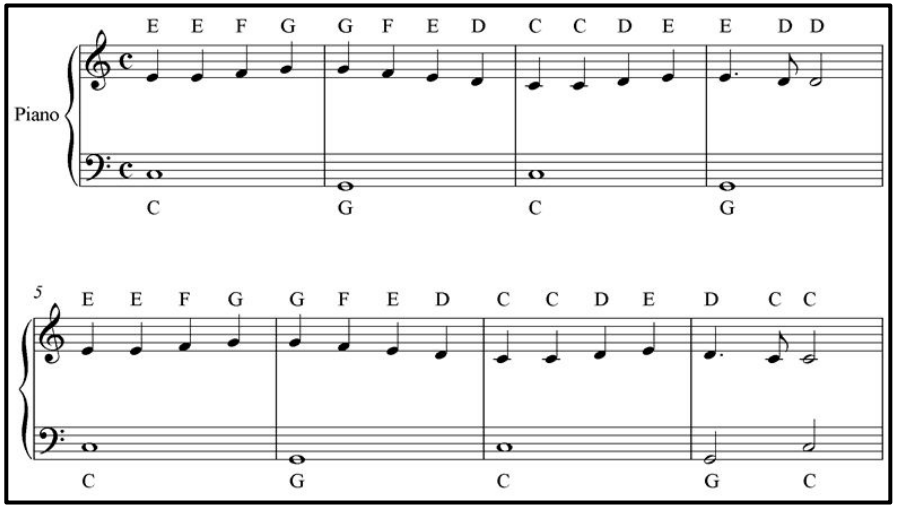
Problem Statement

Sheet music is helpful, but it's:

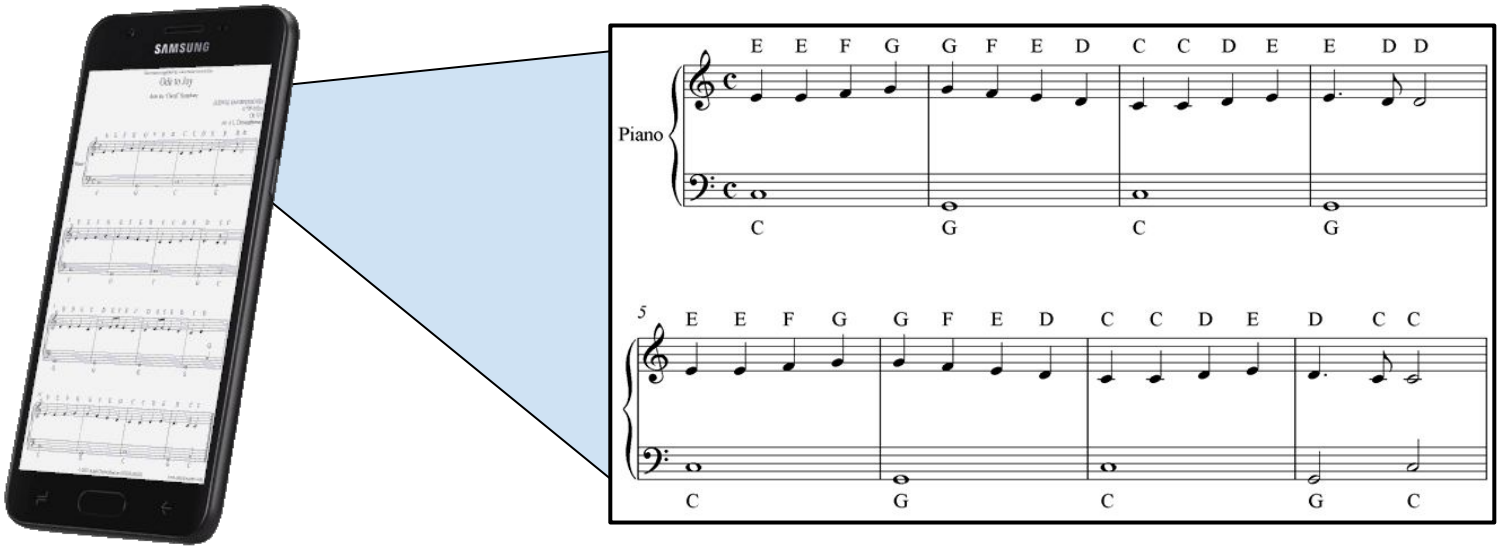
- Hard to read
- Hard to learn from
- Hard to "hear the music"



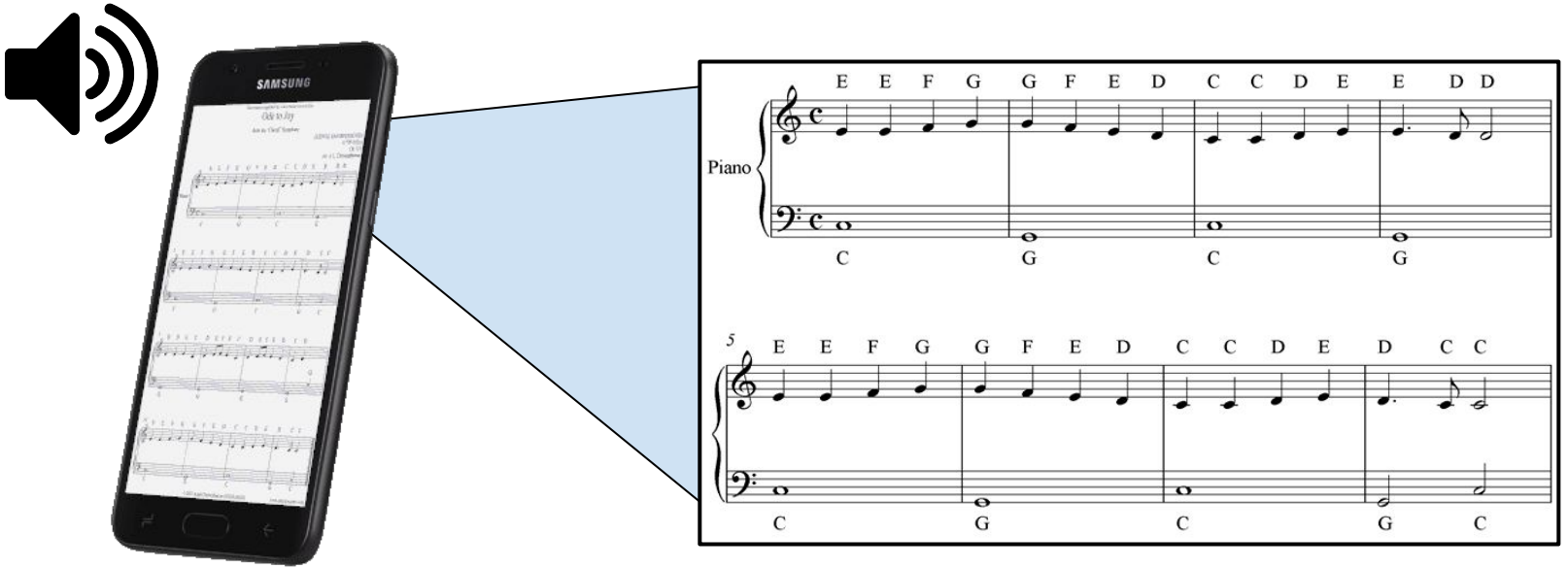
Problem Statement



Problem Statement



Problem Statement



Functional Requirements

- App can take picture & send it to backend CV algorithm.
- CV algorithm identifies sheet music components.
- CV algorithm translates symbols to a music data structure.
- Data sent back to app, and app plays music based on data.
- App shows a piano being played while music is played.

Non-Functional Requirements

- Computer vision algorithm should be able to analyze the sheet music in less than 15 seconds.
- Algorithm should detect most beginner music symbols.
- Application’s music playback should “sound as accurately as possible”.

Constraints and Considerations

- Multi platform application.
- Different styles of sheet music.
- Many different symbols to interpret in sheet music.
  - What is “Beginner sheet music”?
- Operating conditions (Lighting, Camera quality).
- Computer vision may be computationally expensive.

Risks

- Encrypting requests to the server.
- Legality (Photography/Storing of copyrighted music).

Market survey

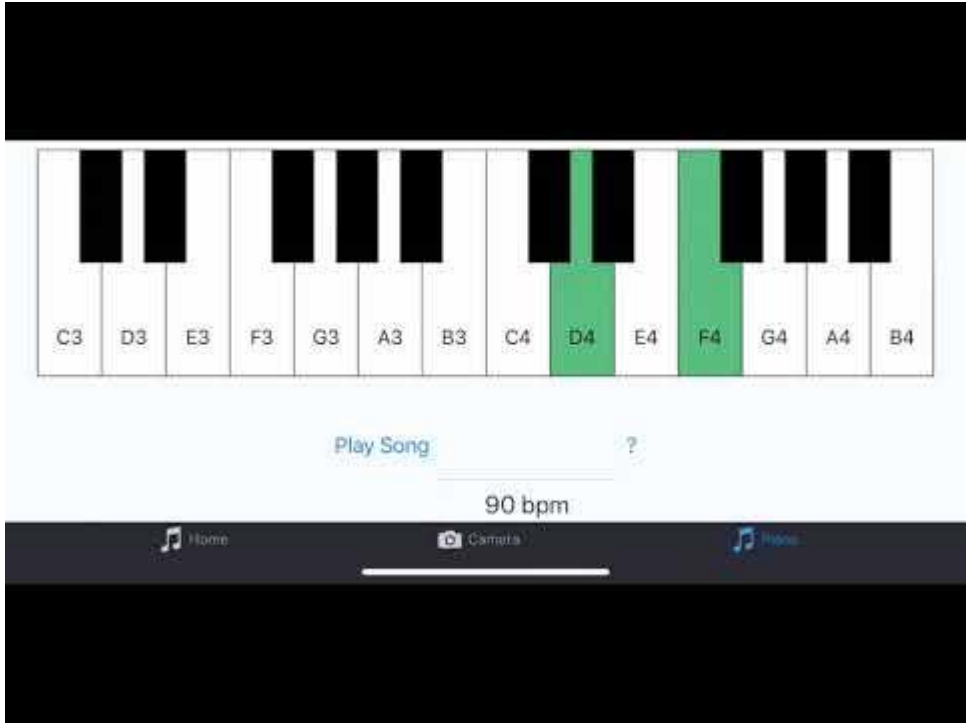
- Sheet Vision
  - Single line only.
  - Only creates MIDI file.
- Musescore
  - Existing created sheet music.
  - Does not show how to play.
- Playscore
  - Poor UI.
  - Misaligned progress bar.
  - Does not show how to play.

System Design

Functional Decomposition

- User scans or selects a sheet music from their gallery and sends it to the server.
- User watches a preview of how to play in accordance to the sheet music.
- User plays the piano to simulate the sheet music.

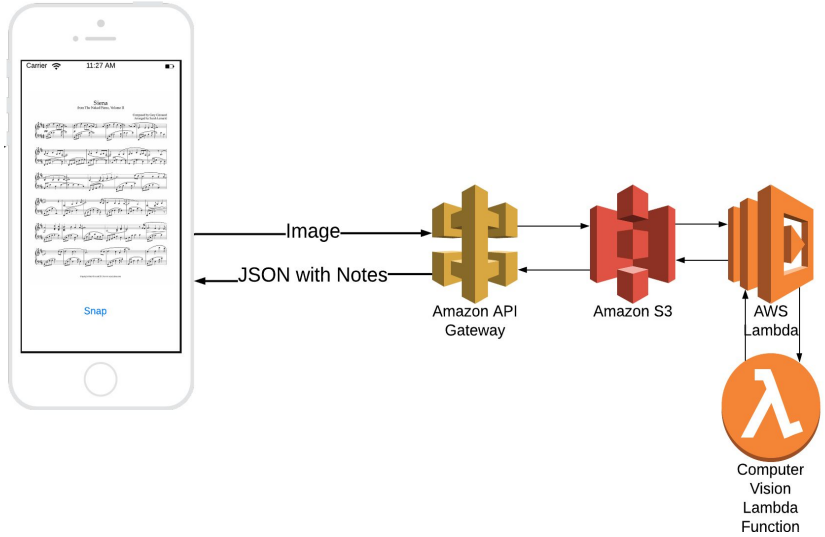
Video Demo



General Architecture

Frontend:
React Native/Expo

- Modular
- Multiplatform
- Javascript

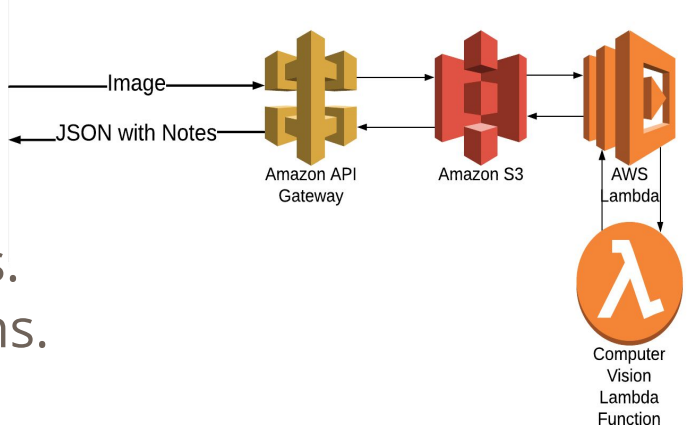


Backend:
AWS

- Lambda, S3, API Gateway
- OpenCV 4.0
- Python 3.6

How does AWS work?

- AWS API Gateway
  - REST API, sends/receives data to different Amazon Web Services.
- AWS S3
  - Buckets contains application files.
  - Files can trigger Lambda functions.
- AWS Lambda
  - Write serverless code in any language.
  - Return data back in a response, or post back data into S3.

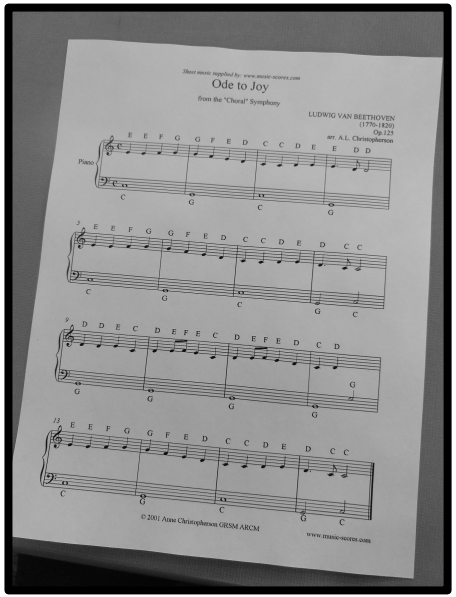


Computer Vision Algorithm

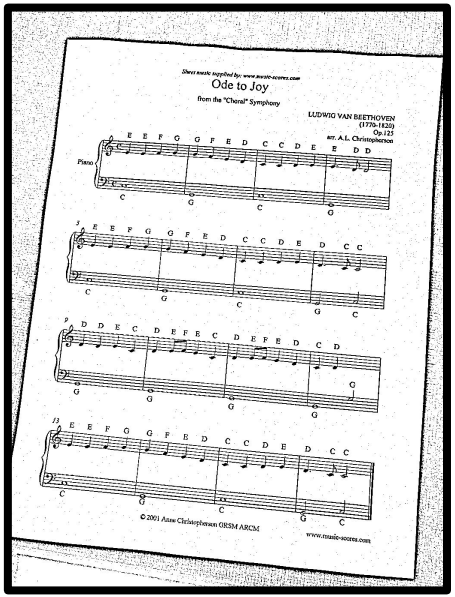
Runs on AWS in four stages:

1. Image sectioning .
2. Staff/Bar identification.
3. Note extraction.
4. Note mapping and JSON construction.

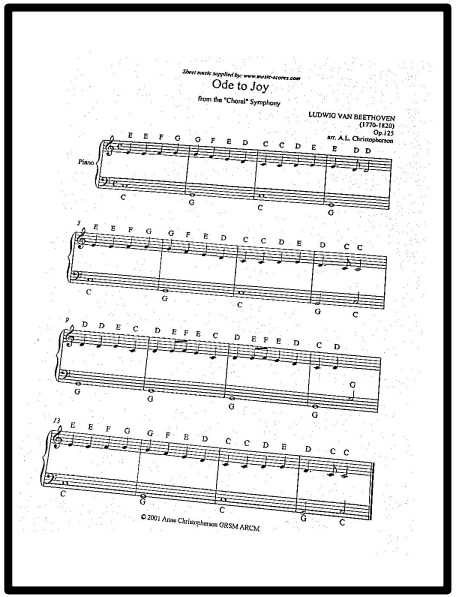
1. Image Sectioning



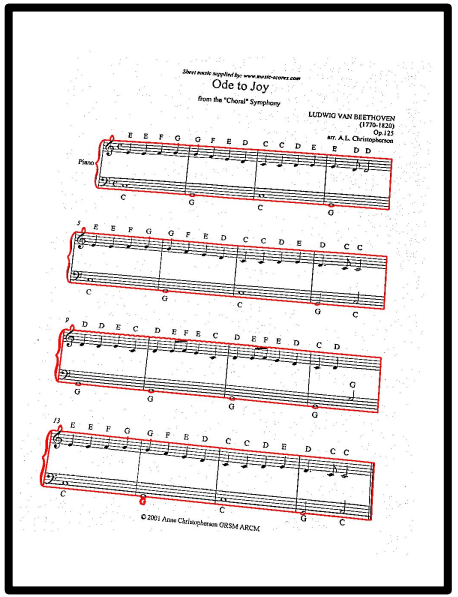
Convert to Grayscale



Gaussian Threshold
(Fixes lighting differences)



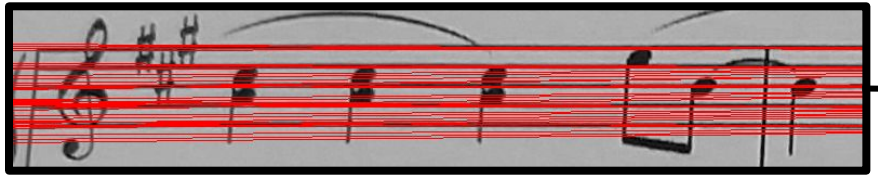
Find Contours
(Find largest white object,
erase all else)



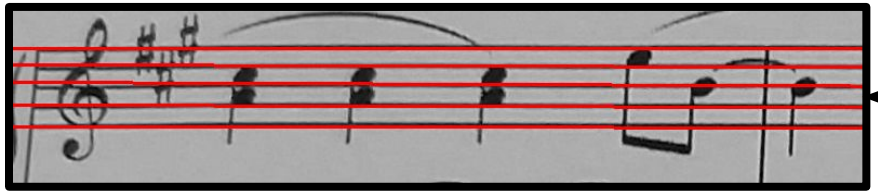
Find Contours
(Find largest black objects)

2. Staff/Bar Identification

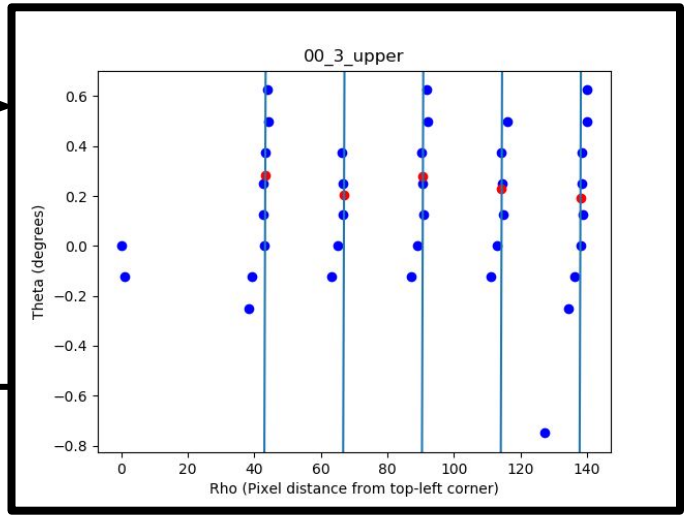
Hough lines (Find all fitting lines in the image)



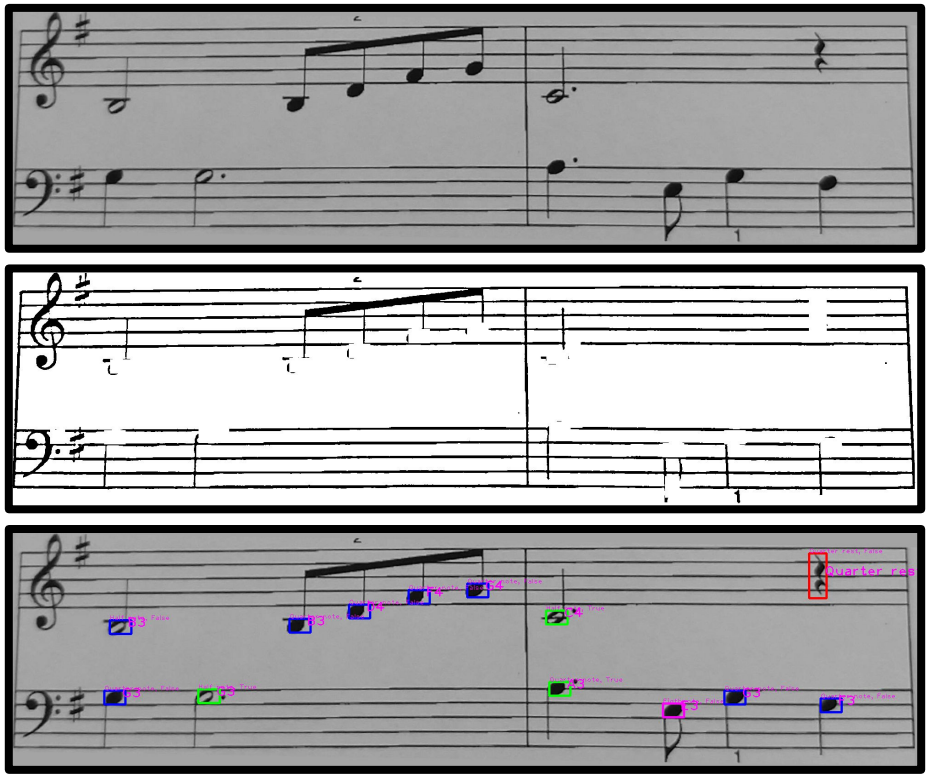
Five custom lines found from Hough lines



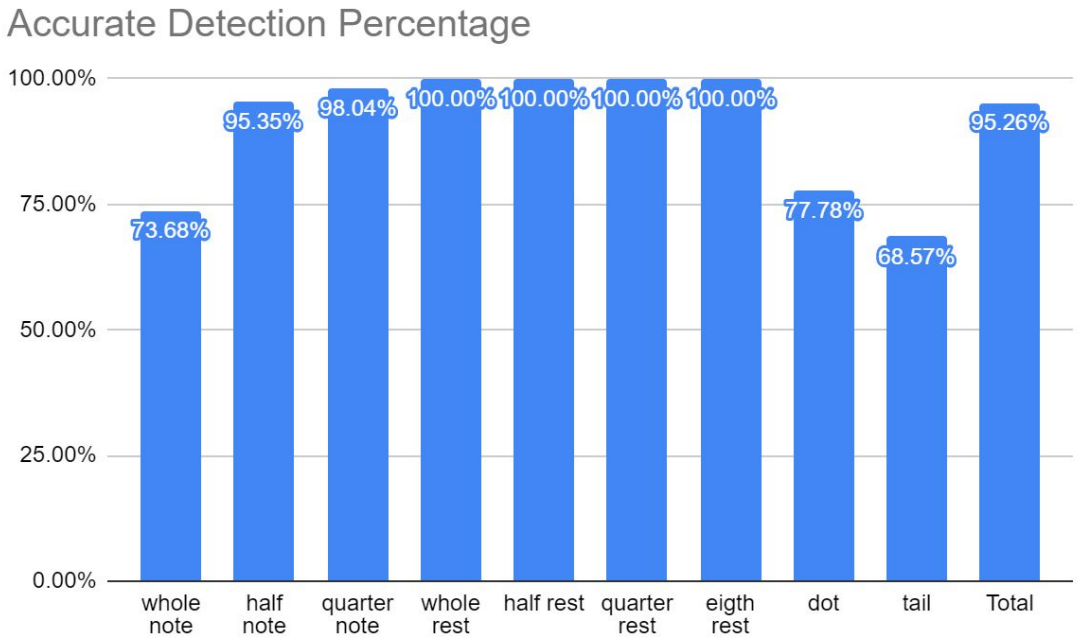
Hough lines dual space (Each blue dot is a line)



3. Note Detection



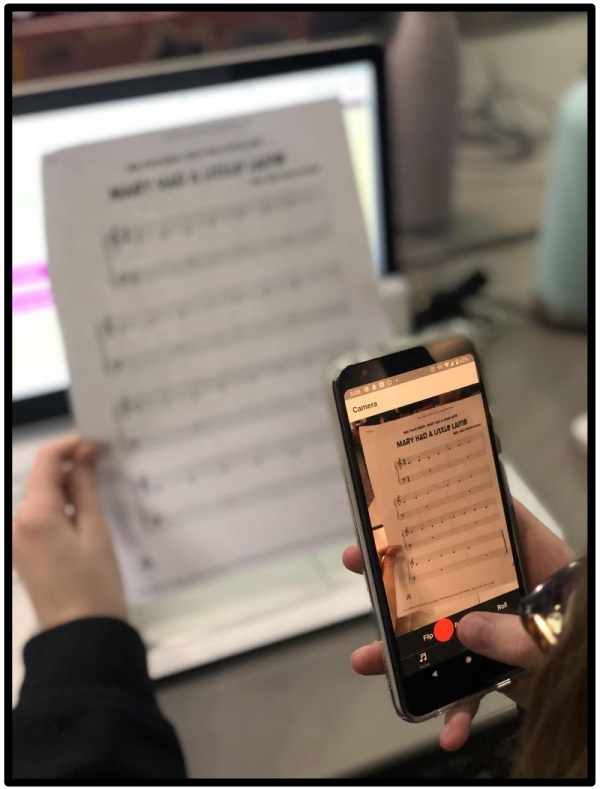
4. Notes To Music Data Structure

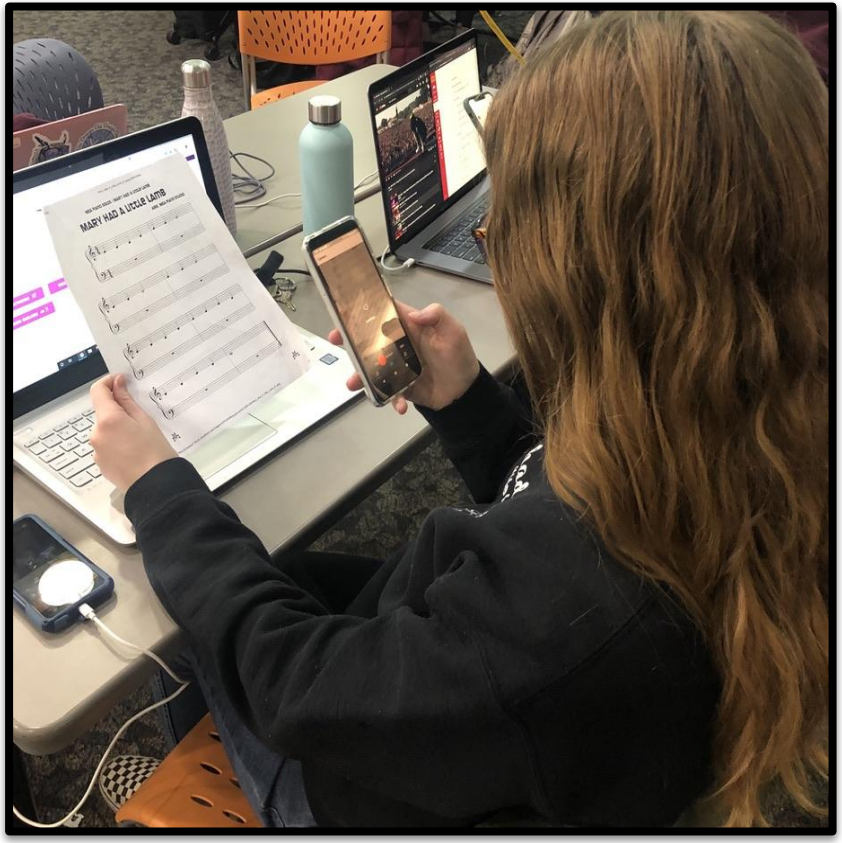


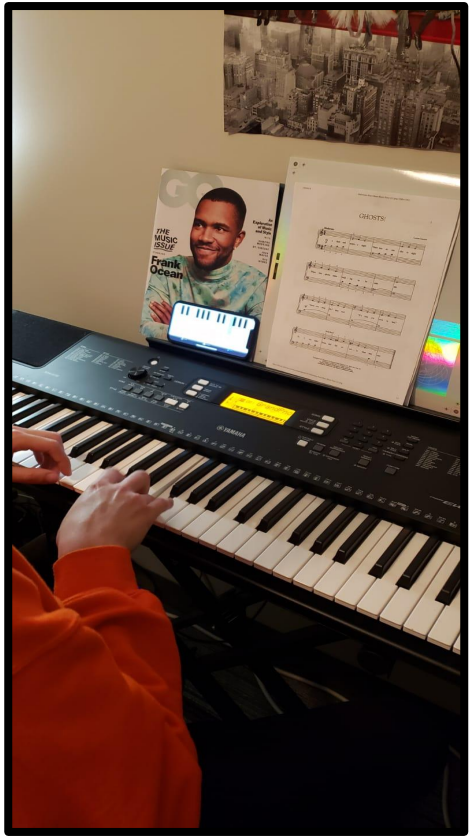
* there were very few whole, half and eighth rests

Test Plan

- AWS Lambda Testing Suite
  - Request Tests
- Functional Testing
  - Application usability
  - Algorithm accuracy
  - Audio “smell test”
- Community Testing




MARY HAD A LITTLE LAMB


THE MUSIC ISSUE
Frank Ocean

Risks Mitigations

- Requests sent are unencrypted.
  - No user data is ever collected or sent to our databases.
  - No risk to our users data.
  - PUT and GETS to AWS lambda only allowed by our client.
  - Data being posted to our S3 buckets and processed is ignored if not of the correct format.
- Legality
  - Sheet music stored only temporarily on app/backend.
  - Not distributing copyrighted material.

Challenges -Backend

- React-Native doesn’t allow requests to and from pages with no Authorized SSL certificates.
- No way to add our EC2 server to Certificate Authorities on Mobile, especially since it’s self-signed.
- AWS Lambda request sizes are limited to 10 MB (images can be larger sometimes).
- Machine Vision algorithm requires clear images (compression wasn’t an option.)
- AWS S3 triggers don’t support POST.

Innovations

Strange request structure to accommodate for limitations.

- PUT request to S3 (base64 string of image).
- S3 triggers lambda function on upload.
- Store result back in S3.
- Get average of how long a request returns.
- Add on the standard deviation.
- Have client GET the output from S3 after given time.

Challenges -Computer Vision

- Image sizing issues
  - Broke symbol detection.
  - Solved by dynamically sizing symbol templates.
- Accuracy issues
  - Still an issue.
  - Reduced by using dynamically sized templates.
- Reducing runtimes

Challenges -Frontend

- Expo libraries
  - Lack of documentation & examples.
  - Cross Platform diversity.
  - Constraints between Expo and React Native.
- Technologies
  - React Native.

Conclusion

Current Project Status

- App main functionalities work as described.
- Machine vision model completed.
- AWS connected with every component.
- Communication between server and app fully functional.

Future Implementations -Frontend/Backend

- Use more well-known development platform
  - Android Studio, XCode, React Native.
- Multiplatform code very inconsistent. Could be easier to work with platforms indepently.
- Continue with AWS
  - Fantastic platform for hosting applications
  - Research more about limitations and interactions with client platforms before finalizing our choices.

Future Implementations -Machine Vision

- Improve performance.
- Improve accuracy.

Task Responsibility/Contributions

Name	Responsibility
Bryan Fung	Frontend Developer
Garrett Greenfield	Frontend Developer
Ricardo Faure	Frontend/Backend Architect
Trevin Nance	Machine vision Engineer
Walter Svenddal	Machine vision Engineer

Acknowledgement

Thank you ISU Engineering Department and Dr. Daniels for providing us with this opportunity.

We would like to thank our faculty advisor Dr. Stoytchev for his support and great feedback.

Questions?