Duckhunter

Final Project Presentation

Taylor Barton and Andrew Lisy

tbarton@mit.edu, alisy@mit.edu

6.111 - Digital Electronics Laboratory

Massachusetts Insitute of Technology

Outline of Talk

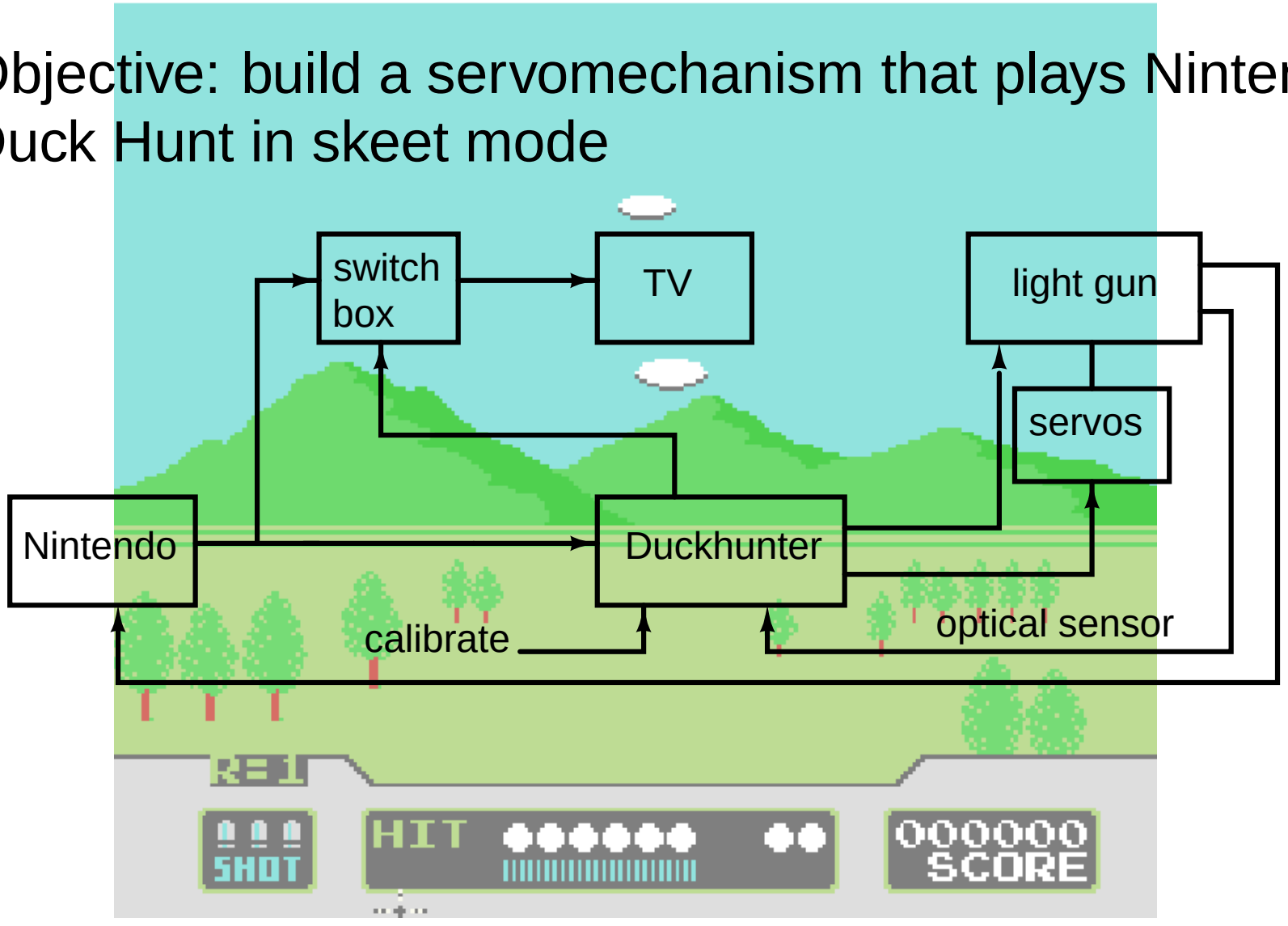
- Objectives and System Overview
- Block Diagram
- Input System
- Output Servomechanizm
- Calibration
- What's Next?
- Conclusions

Taylor Barton

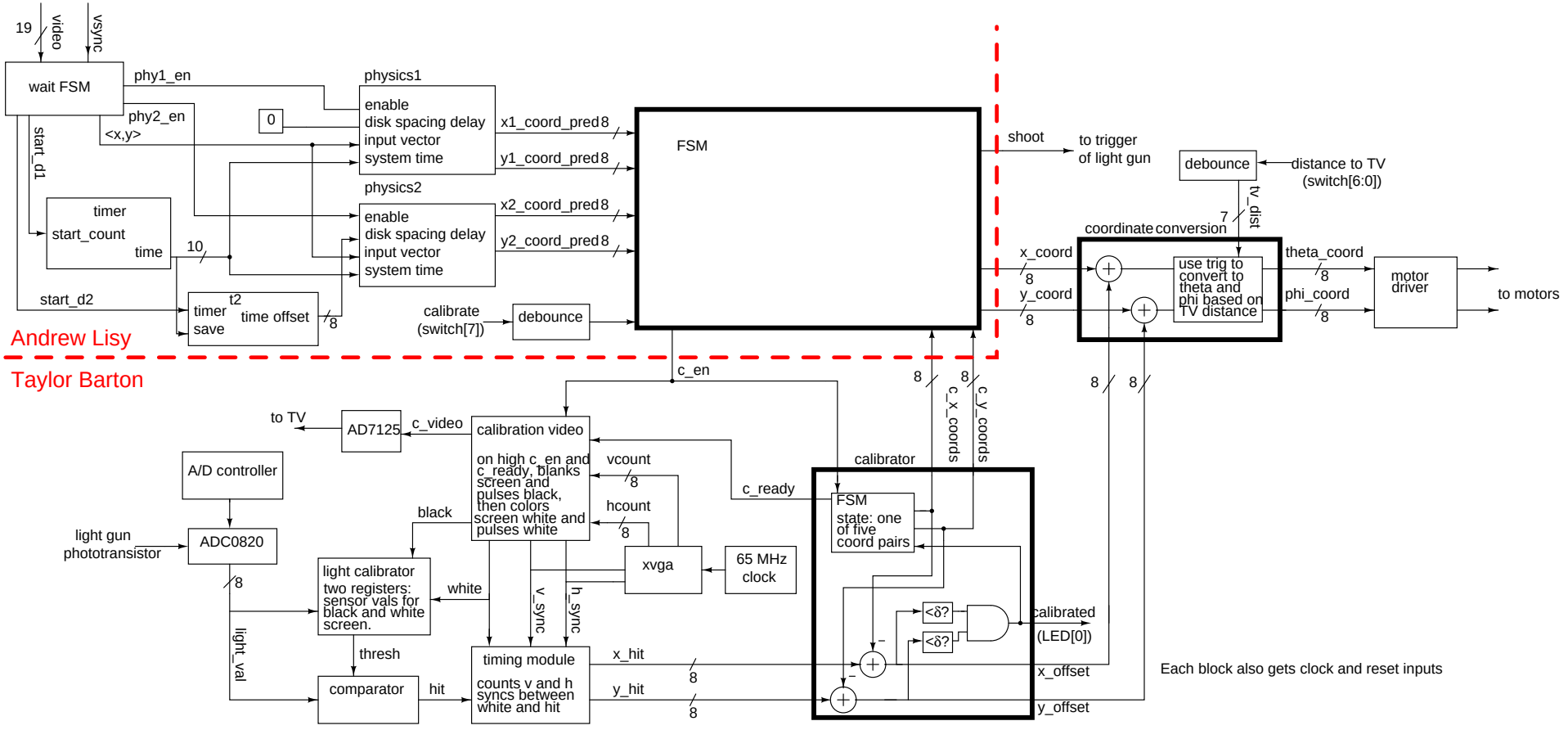
Andrew Lisy

Objectives and System Overview

- Objective: build a servomechanism that plays Nintendo's Duck Hunt in skeet mode



Block Diagram



Input System

Function: Translate the video output of the Nintendo to time-adjusted coordinates for the gun.

Input System

Function: Translate the video output of the Nintendo to time-adjusted coordinates for the gun.

Subcomponents:

Input System

Function: Translate the video output of the Nintendo to time-adjusted coordinates for the gun.

Subcomponents:

- Vector Analyzer - from frame sequence, detect discs and determine initial velocity vector $< \overrightarrow{x_1}, \overrightarrow{y_1} >$ for each disc.

Input System

Function: Translate the video output of the Nintendo to time-adjusted coordinates for the gun.

Subcomponents:

- Vector Analyzer - from frame sequence, detect discs and determine initial velocity vector $< \overrightarrow{x_1}, \overrightarrow{y_1} >$ for each disc.
- Physics Engine - Using vectors, apply formulas:

$$x_t = \overrightarrow{x_0}t,\ y_t = \overrightarrow{y_0}t + \tfrac{1}{2}g_{nes}t^2$$

where $g_{nes} =$ experimentally determined gravity force of NES (in $\mathrm{pixels/frame}^2$).

Input System

Function: Translate the video output of the Nintendo to time-adjusted coordinates for the gun.

Subcomponents:

- Vector Analyzer - from frame sequence, detect discs and determine initial velocity vector $< \overrightarrow{x_1}, \overrightarrow{y_1} >$ for each disc.
- Physics Engine - Using vectors, apply formulas:

$$x_t = \overrightarrow{x_0}t,\ y_t = \overrightarrow{y_0}t + \tfrac{1}{2}g_{nes}t^2$$

  where $g_{nes} =$ experimentally determined gravity force of NES (in $\mathrm{pixels/frame}^2$).
- Timer - keeps track of delays in $1/10$ seconds.

Input System: Vector Analyzer

1. Wait for white ’blip’ to appear just above scoring panel.

Input System: Vector Analyzer

1. Wait for white ’blip’ to appear just above scoring panel.
2. When blip is found, record coordinates in register bank A1. Wait a frame, record new coordinates to to A2 and assert start_d1.

Input System: Vector Analyzer

1. Wait for white ’blip’ to appear just above scoring panel.
2. When blip is found, record coordinates in register bank A1. Wait a frame, record new coordinates to to A2 and assert start_d1.
3. Set ’disc_present’ register to high to indicate we have coordinates for one disc.

Input System: Vector Analyzer

1. Wait for white ’blip’ to appear just above scoring panel.
2. When blip is found, record coordinates in register bank A1. Wait a frame, record new coordinates to to A2 and assert start_d1.
3. Set ’disc_present’ register to high to indicate we have coordinates for one disc.
4. Output $< x_{A2} - x_{A1}, y_{A2} - y_{A1} >$ to vector output

Input System: Vector Analyzer

1. Wait for white 'blip' to appear just above scoring panel.
2. When blip is found, record coordinates in register bank A1. Wait a frame, record new coordinates to to A2 and assert start_d1.
3. Set 'disc_present' register to high to indicate we have coordinates for one disc.
4. Output $< x_{A2} - x_{A1}, y_{A2} - y_{A1} >$ to vector output
5. Repeat process for next disk (using register bank B1, B2). When blip for next disk is found, assert start_d2.

Input System: Physics Engine

Inputs: Vector from *Vector Analyzer*, system time t, gun aiming latency t_{aim}, shot spacing offset t_{sp}

Input System: Physics Engine

Inputs: Vector from *Vector Analyzer*, system time t, gun aiming latency t_{aim}, shot spacing offset t_{sp}

Output: Coordinates of the two discs, compensating for aiming latency and spacing offset, as a function of t, t_{aim}, t_{sp}

Input System: Physics Engine

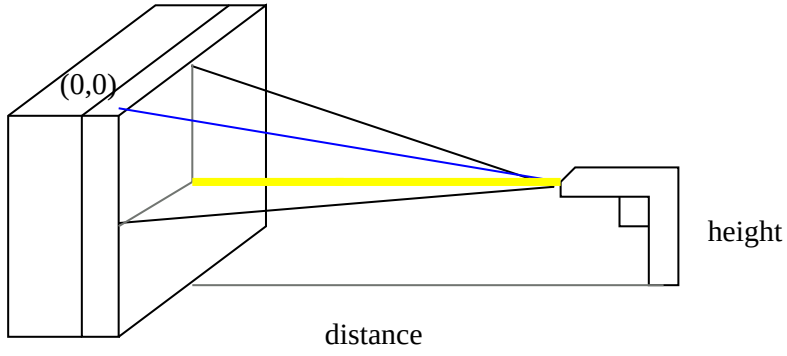
Inputs: Vector from *Vector Analyzer*, system time t, gun aiming latency t_{aim}, shot spacing offset t_{sp}

Output: Coordinates of the two discs, compensating for aiming latency and spacing offset, as a function of t, t_{aim}, t_{sp}

Implementation: The first physics engine receives input t from *timer*, t_{aim} from experimental values, and t_{sp} wired to ground (since it handles the first disc, there is no spacing delay). The second module receives t and t_{aim} from the timer and experimental values, as well as t_{sp} from the *t2* module.

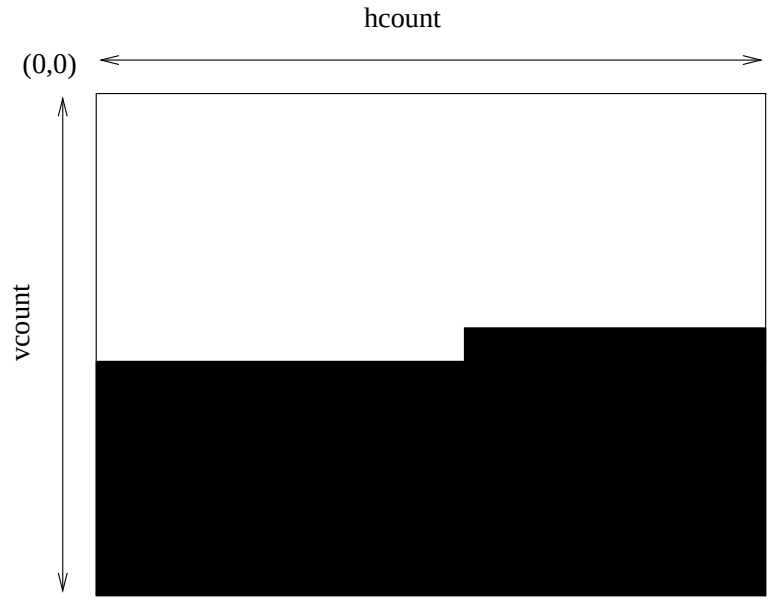
Servomechanism

- The output system takes the (x, y) coordinate from the physics engine and translates it into angles for the servomechanism.



- inputs: (x, y) coordinates, (x, y) offsets, distance from TV, height (constant)
- outputs: motor control signal
- Servo resolution ≈ 1 degree.

Servomechanism: Calibration



- Calibrator outputs video to TV: first black, then white screen.
- Timing module counts the hsync and vsync signals between the start of the white screen and the gun sensing a hit to determine position.

- System is calibrated by comparing intended and actual coordinates.

What's next?

1. Real-time Feedback: If we are able to find a version of the NES *Duck Hunt* game that uses a specific mode of hit detection (blanking then whiteing the entire screen), we can implement real-time calibration to our system, thus increasing the accuracy.

What's next?

1. Real-time Feedback: If we are able to find a version of the NES *Duck Hunt* game that uses a specific mode of hit detection (blanking then whiteing the entire screen), we can implement real-time calibration to our system, thus increasing the accuracy.
2. Miss Correction: If possible, we will implement a hit detection mechanism which sends a signal into the FSM after the gun is fired. If the signal is low, re-aim and fire. If high, move to the next state.

What's next?

1. Real-time Feedback: If we are able to find a version of the NES *Duck Hunt* game that uses a specific mode of hit detection (blanking then whiteing the entire screen), we can implement real-time calibration to our system, thus increasing the accuracy.
2. Miss Correction: If possible, we will implement a hit detection mechanism which sends a signal into the FSM after the gun is fired. If the signal is low, re-aim and fire. If high, move to the next state.
3. Duck Hunt Mode: Depending on the mechanical limitations of our tracking and firing mechanism, *Duckhunter* could be implemented to play the Duck Mode of *Duck Hunt*

Conclusion

- Mechanical considerations limit how fast and accurate the digital system can be.

Conclusion

- Mechanical considerations limit how fast and accurate the digital system can be.
- The calibration routine, while seemingly mundane, is a very critical part of the system, since the gun will be moved from station to station. Manual calibration would be tedious and unreliable.

Conclusion

- Mechanical considerations limit how fast and accurate the digital system can be.
- The calibration routine, while seemingly mundane, is a very critical part of the system, since the gun will be moved from station to station. Manual calibration would be tedious and unreliable.
- Determining the coefficients for the physics engine may require a separate system to track and analyze data from previous runs.

Conclusion

- Mechanical considerations limit how fast and accurate the digital system can be.
- The calibration routine, while seemingly mundane, is a very critical part of the system, since the gun will be moved from station to station. Manual calibration would be tedious and unreliable.
- Determining the coefficients for the physics engine may require a separate system to track and analyze data from previous runs.
- If we are successful, this system may succeed where we have failed: reaching the final level of *Duck Hunt*