AI Vision Security Guard: Automated CSV Log Engine in PictoBlox

By pavanhl in Teachers > University+

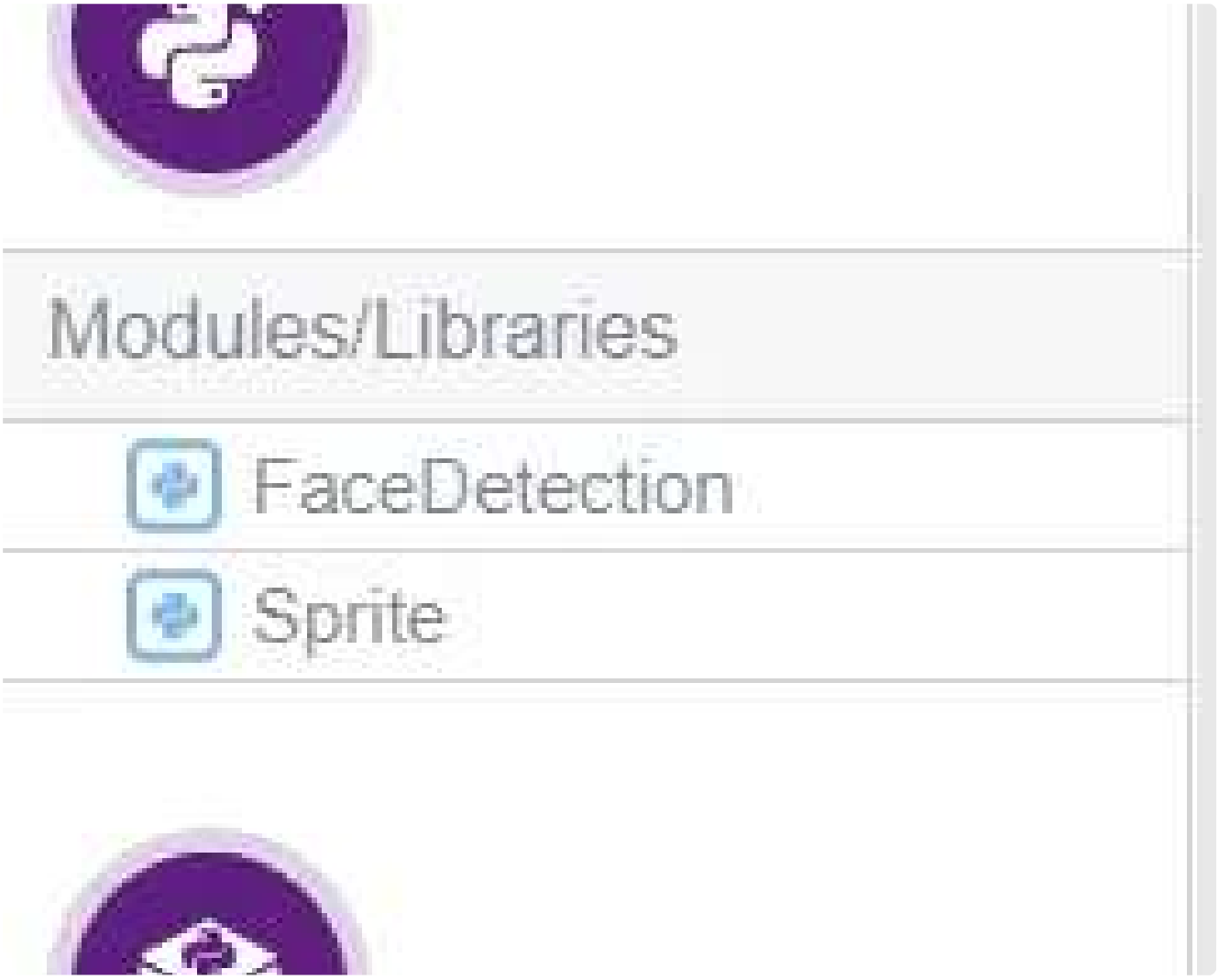
This project introduces an intelligent, edge-based security monitor built entirely within the PictoBlox Python environment. Conventional security systems often require massive continuous recording setups that generate gigabytes of stagnant, empty video files. This solution addresses that problem by introducing an intelligent, event-driven data automation framework.

By utilizing a local computer vision model via the Face Detection extension, the application actively scans a room. The instant a human face enters the frame, a multi-tier Python file automation pipeline initiates: it accesses the host operating system, checks for or dynamically creates an

optimized, dated directory structure, and appends a real-time, comma-separated event row (.csv) tracking structural entry details.

To prevent "file spamming" (writing to the disk hundreds of times while a person stands in front of the camera), this architecture uses a custom state latching mechanism. The program writes a single, precise entry upon target detection, locks further logging, and safely unlocks only when the environment returns to a completely clear baseline. This provides clean data telemetry logs suitable for instant parsing in Excel or Google Sheets.

Supplies



Main Platform: PictoBlox IDE (Downloaded and installed on Windows, macOS, or Linux). The workspace must be toggled from "Blocks" mode into the native **Python Coding Environment**.

Hardware Interface: A standard built-in laptop webcam or a plug-and-play external USB webcam.

PictoBlox Core Extension: The **Face Detection Extension**. This must be loaded using the yellow "Extensions" button in the bottom-left corner of the PictoBlox window. This extension imports the custom FaceDetection hardware/AI libraries into the environment.

Python Libraries Used (Built-in & Platform Specific):

The script utilizes a mix of Python's standard library modules and PictoBlox's proprietary machine learning wrappers:

1. **os (Python Standard Library):** Used for interacting with your computer's operating system. It handles the automated file directory scanning (os.path.exists) and directory compilation (os.makedirs).
2. **time (Python Standard Library):** Used to interface with your computer's hardware clock. It pulls live timestamps for data logging (time.strftime) and manages processing delays (time.sleep) to optimize CPU efficiency.
3. **FaceDetection (PictoBlox AI Module):** Loaded automatically via the UI extension. This wrapper interfaces with the computer vision models that slice live camera frames, render on-screen bounding boxes (fd.enablebox), and count human face vector hits (fd.count).

Step 1: Steps to Set Up the Program

Step 1: Setting Up the Workspace and Core Python Script

Step Title: Step 1: Setting Up the Workspace and Core Python Script

Step Body Text:

Open PictoBlox, ensure your workspace is switched to the **Python Environment**, and make sure you have loaded the **Face Detection** extension. Copy and paste the following complete, Object-Oriented code into your main script window:

Python

```
import os
import time

class IntelligentSecurityLogger:
def __init__(self):
"""Initializes the file automation paths and the AI Face Detection module."""
# 1. Automated File Directory Setup
self.folder_name = f"Security_Logs_{time.strftime('%Y_%m_%d')}"
self.csv_path = f"{self.folder_name}/activity_log.csv"
```

```
self._setup_file_system()

# 2. PictoBlox AI Camera Setup
self.fd = FaceDetection()
self.fd.video("on", 0) # Camera on, 0% transparency
self.fd.enablebox() # Draw bounding box around faces
self.fd.setthreshold(0.5) # 50% confidence threshold
print("AI Camera Initialized. Scanning room...")

# 3. State Tracking Variable (Prevents duplicating logs)
self.person_already_logged = False

def _setup_file_system(self):
"""Internal helper method to handle folder and CSV creation."""
if not os.path.exists(self.folder_name):
os.makedirs(self.folder_name)
print(f"Created folder: {self.folder_name}")

if not os.path.exists(self.csv_path):
with open(self.csv_path, "w") as file:
file.write("Timestamp,Event_Type,Status\n")
print("Initialized new CSV log file.")

def log_event(self):
"""Appends a new security alert line into the CSV file."""
timestamp = time.strftime('%Y-%m-%d %H:%M:%S')
with open(self.csv_path, "a") as file:
```

```
file.write(f"{timestamp},Human Detected,ALERT\n")

def monitor(self):
"""Main execution loop that keeps the security scanner running."""
try:
while True:
self.fd.analysecamera() # Process current camera frame
# Check if a face is in the frame
if self.fd.count() > 0:
if not self.person_already_logged:
current_time = time.strftime('%H:%M:%S')
print(f"[{current_time}] Alert: Human detected in frame!")
self.log_event() # Trigger file automation log
self.person_already_logged = True
else:
# Reset the logging latch when the room becomes empty
if self.person_already_logged:
print("Room is clear again. Resetting monitor.")
self.person_already_logged = False
time.sleep(0.2) # Maintain stable CPU performance

except KeyboardInterrupt:
print("\nSecurity System safely deactivated.")

# Execution Block
if __name__ == "__main__":
security_system = IntelligentSecurityLogger()
```

```
security_system.monitor()
```

Step 2: How the Code Works (The Logic Breakdown)

*Click **"Add Step"** again to create this block.*

Step Title: Step 2: Understanding the Automation & AI Architecture

Step Body Text:

Here is how the Python backend communicates with your computer's storage system and the AI model:

1. **Dynamic File System Setup (_setup_file_system):** The code pulls your computer's live internal date and structures a custom string (e.g., Security_Logs_2026_05_22). Using the os library, it verifies if that directory exists. If it doesn't, it generates it natively. It then opens a stream to build a custom spreadsheet .csv template with headers if one isn't present.
2. **AI Computational Slicing (fd.analysecamera):** During runtime processing loops, PictoBlox intercepts video streaming frames. The fd.analysecamera() function passes frame arrays directly to spatial processing models. If a face coordinates over 50% match probability, it flags a hit to fd.count().
3. **The State Machine Latch (self.person_already_logged):** If someone stands in front of the camera, a raw loop would write to the spreadsheet dozens of times a second. Our logic uses a boolean tracking flag to "lock" writing after the first frame hit. It safely unlocks

only when the room goes completely clear again, ensuring your logs stay clean and compact.

Step 3: Running and Testing Your Project

*Click **"Add Step"** one last time to finish.*

Step Title: Step 3: Running and Validating the System

Step Body Text:

Now it's time to run your project! Follow these steps to verify your system:

1. Click the green **Run** button inside the PictoBlox Python interface. Your webcam will light up instantly, showing a live bounding-box preview window on your stage canvas.
2. Step out of the frame initially to establish a clear baseline.
3. Step back into the camera's focus window. Look directly at your Python console terminal window inside PictoBlox; you will see it output the live alert string printout right as it spots you!
4. Terminate the script run. Navigate locally into your computer files where your PictoBlox project workspace saves assets.
5. Open your newly created, self-named dated folder, and launch activity_log.csv. Your precise human-detection timestamp logs are now cleanly cataloged across rows, perfectly ready for any spreadsheet application!