AI-RPi for Scalable User-Level Device Automation

Etienne DEGUINE - test.ai - etienne@test.ai
2021-10-11
PNSQC 2021
Problem statement

The core of our issue is user application automation without OS provided automation driver/API.

A way to frame the problem is to define the requirements as

- **Observe**: this is the input from which deduce the state of the application
- **Control**: this is the output that we use to control the application
- **Program**: this is the intelligence and logic of your test
- **Scale**: we are tackling the case of device farms with 30+ devices
Example 1 - chromebooks device farm

At test.ai we specifically needed to automate a device farm of chromebooks running ChromeOS and we needed it at the base level (managing login screen, rolling updates, reboots) no solution in the market
Example 2 - video game testing
Our solution

Observe: HDMI capture card (B101)
Control: USB input emulation (usb_gadget)
Program: web APIs (flask)
Scale: create an OS image and flash
HDMI capture

e.g. `gst-launch-1.0 -e -v v4l2src device=/dev/video0 ! v4l2convert !
video/x-raw,width=1280,height=720,framerate=20/1 ! omxh264enc ! h264parse ! matroskamux ! filesink location={video_name}`
USB emulation

We use a feature of Linux called usb_gadget

Basically define a USB report descriptor file exposing the USB communication format with the host, explained in the USB human interface device (HID) standard. We create a descriptor that encodes both keyboard and mouse info in a single packet. We also got it to work with game controllers. Snippet below:

```
INPUT (Const,Var,Abs)

0x05, 0x01,    // USAGE_PAGE (Generic Desktop)
0x09, 0x30,    // USAGE (X)
0x09, 0x31,    // USAGE (Y)
0x09, 0x38,    // USAGE (Wheel)
0x15, 0x01,    // LOGICAL_MINIMUM (-127)
0x25, 0x01,    // LOGICAL_MAXIMUM (127)
```
The USB data we send to the host contains relative coordinate movements e.g. (+24, -125) but those are not actual pixels, they depend on screen resolution and host quirks.

So we have to calibrate the system so that we can write something like `click(1337, 121)`

Calibration process:

1. run a full screen click listener on the host
2. sample coordinates and deduce how they match up. Essentially we build a lookup dictionary
3. decompose movements because the mouse input caps at (-127, 127) in our implementation.
The final setup is quite compact.
Fleet management - quality of life

On top of the HDMI and USB web server exposing our observe and control APIs, we also created a simple ‘OS server’ that basically allows us to run routine fleet management tasks via a web API:

- pull the code and restart the servers with latest
- restart the RPI
- deactivate / reactivate the HDMI input or the USB emulation
- grep the logs
- make a grid screenshot of all the machines in the fleet to quickly see the state of all the fleet

It’s really all about convenience and managing a lot of devices
Summary functional diagram
the “borg” view - screenshot grid
Fun extra feature - remote desktop

As you might have guessed, our implementation also allows us to create what is essentially a poor man’s KVM = remote desktop implemented in hardware.

For fun, we created it as follow:

- a pyQT UI which connects to the HDMI stream with the gstreamer library and displays it live in the UI (we observe latency around 200ms on LAN, could be better)
- click on the screen to emulate mouse clicks
- type in a text box to send keyboard events (text or shortcuts)

The solution is generic enough that you could create a remote desktop environment in hardware for any system with HDMI + keyboard + mouse.
Demo: https://watch.screencastify.com/v/uMqkV81Z822aUI74HV7M
Summary of the tech stack

HDMI capture:
- B101 chip (by Auvidea)
- v4l2 (video 4 linux)
- gstreamer

USB emulation: Linux usb_gadget

Web apis: flask

Remote desktop: python gstreamer library, pyQt
test.ai - our platform

HUMAN INTERFACE
- BOT TRAINING: Labeling Interface
- TEST SPECIFICATION: Authoring Interface
- BOT CONFIGURATION: Execution Interface
- ANALYTICS: Reporting Interface

APPLICATION INTERACTION
- EXPLORATION: Interactive, Autonomous
- TEST EXECUTION: Scripted, Goal-Based

CORTEX: MACHINE LEARNING BRAIN
- OBJECT LOCALIZATION: Chopping, Template Matching
- OBJECT CLASSIFICATION: Screens, Elements, Widgets
- LEARNING: Supervised, Reinforcement

DEVICE INTERACTION
- SENSORS: Object Model, Image Capture, Video Capture
- ACTUATORS: Mouse, Keyboard, Touch, Game Controller

SUPPORTED DEVICES AND PLATFORMS
- Laptop, Desktop, Mobile
- iOS
- Android
- PS4
- XBOX
- SWITCH
Closing words

With our approach, any system with input / output in the form of HDMI and USB can be automated.

When combined with a testing solution that uses computer vision and works in visual only mode, the sky is the limit.

If this fits your bill of requirements contact us, our AI can work in visual only mode using our chopper and test automation platform.

Example work we did using this platform:
- ChromeOS at scale app testing
- Game controller + video game testing as well as input latency measurement in streaming
- Android new phone sign-up flow automation
Q & A
Team

Coders

Etienne DEGUINE
Alexander WU

Leadership

Chris NAVRIDES