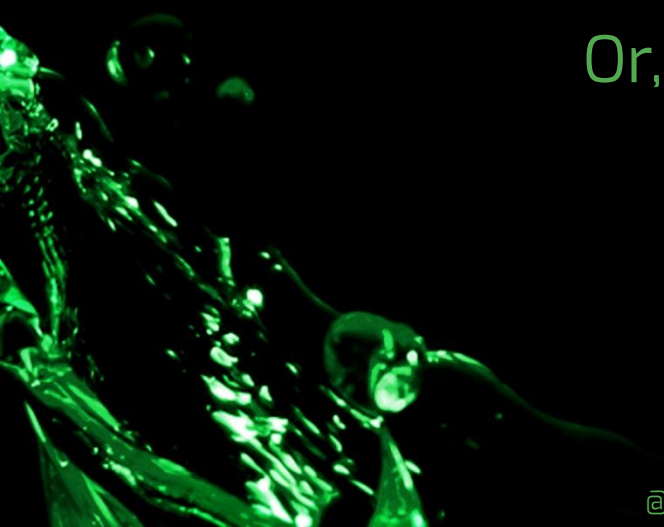
Snakes on a Car

Or, Overengineering a Toy



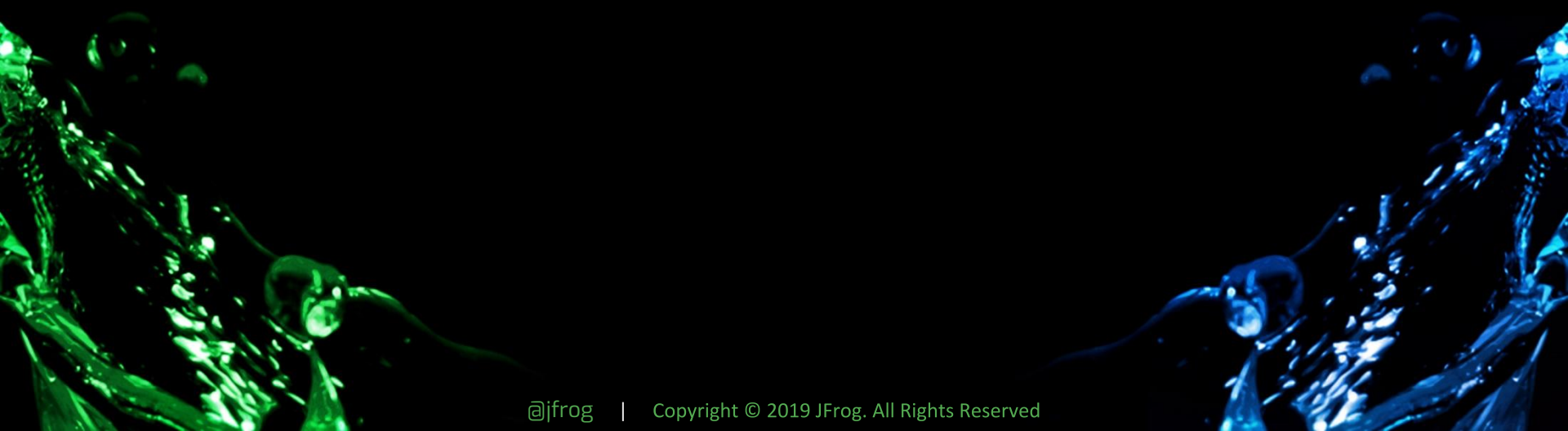
 |

Kat Cosgrove

Developer Advocate
IoT Engineer
Twitter: @Dixie3Flatline
Email: KatC@jfrog.com
bit.ly/SDPythonJFrog

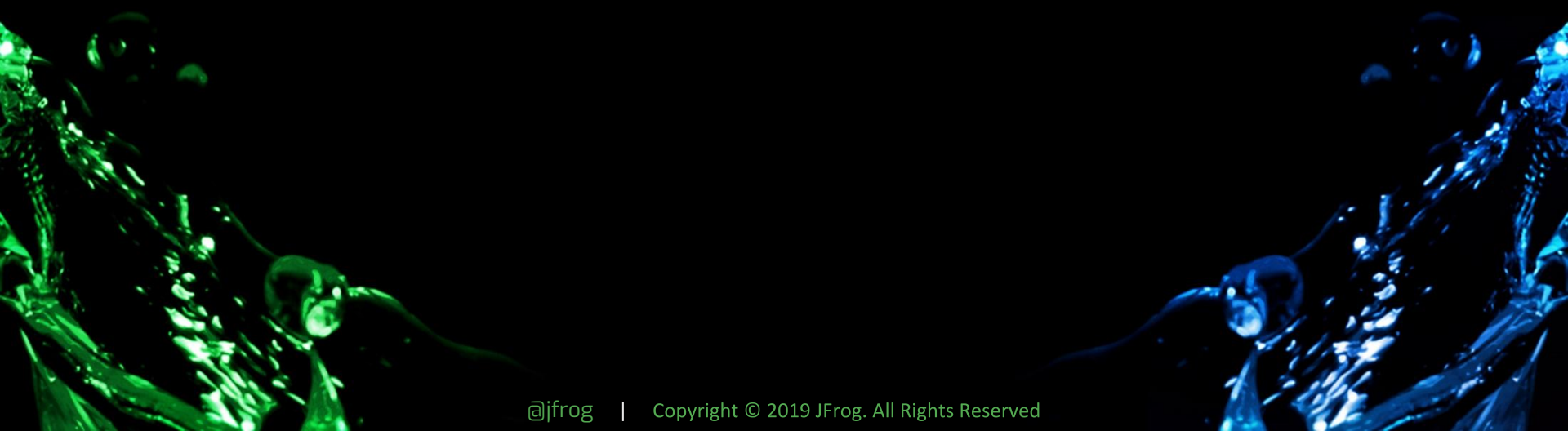
 |

So, what's this demo?



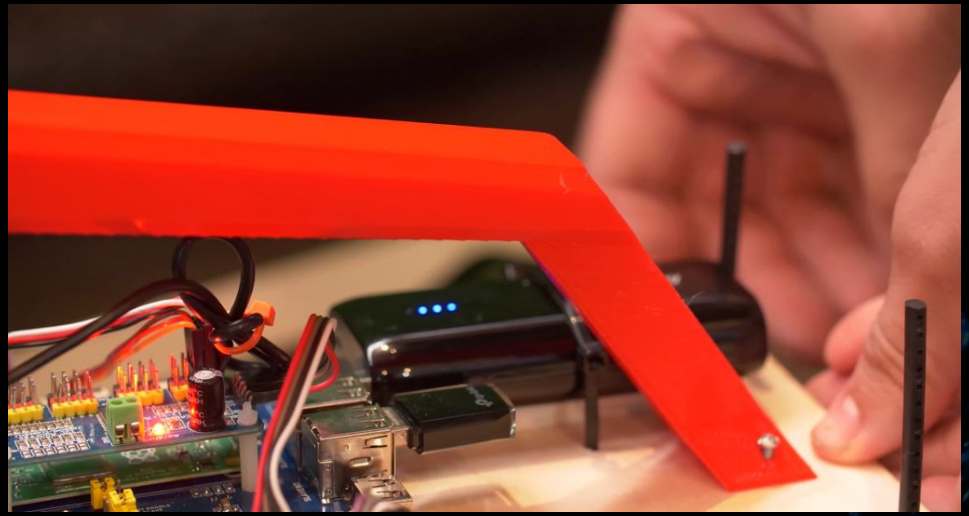
 |

A basic, self-driving miniature car



Donkey Cars

- About $250
- R/C Car
- Raspberry Pi 3B
- Pi Camera
- Race them!



Now make it cooler

- Control it with a USB race wheel + pedals instead
- Automate training new models
- Move camera feed to Driver's screen
- Add a green screen for some flair

Swapping the Controls

```
class Wheel(object):
    def __init__(self, cfg):
        self.state = {
            'angle': 0.0,
            'throttle': 0.0,
            'throttle_offset': 0.0,
            'mode': 'user',
            'recording': False,
        }
        resolution = cfg.CAMERA_RESOLUTION
        self.resolution = (resolution[1], resolution[0])

        context = zmq.Context()

        self.subscriber = context.socket(zmq.SUB)
        self.publisher = context.socket(zmq.PUB)
        self.publisher.set_hwm(10)

        self.subscriber.connect(cfg.ZMQ_PROXY_SUB)
        self.publisher.connect(cfg.ZMQ_PROXY_PUB)

        topicfilter = b'donkeycar.racewheel'
        self.subscriber.setsockopt(zmq.SUBSCRIBE, topicfilter)
        self._lastimg = b''
```


Magic

```
def steering_magic(self, value):
    full_scale = 8.0 * self.steering_scale
    return self.clip(
        self.center(value, 2 ** 16) * full_scale + self.steering_veer,
        -self.steering_range, self.steering_range,
    )

def throttle_magic(self, value):
    real = float((2 ** 8) - value) / 2 ** 8

    # Apply a negative expo scale function
    base = 0.03
    scale = self.throttle_scale
    real = (
        (1.0 - base ** real) * (1.0 / (1.0 - base)) * scale
    )
    top_speed = (self.top_speed/80.0)
    return self.clip(real, 0.0, top_speed)
```

 |

The Driver's Seat

- Managed by Intel NUC
- Sanic Webserver
- VueJS front-end
- ZMQ proxy
  - CI/CD
  - Image feed
  - Racewheel data

 |

Writing a Green Screen

- Read frame with OpenCV
- Convert to HSV
- Set HSV range
- Create mask from range
- Crop background
- Merge them

```
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

mask = cv2.inRange(hsv, config["lower_green"], config["upper_green"])
mask = cv2.erode(mask, kernel, iterations=2)
mask = cv2.dilate(mask, kernel, iterations=2)

blur = cv2.GaussianBlur(mask, (11, 11), 0)
smooth = cv2.addWeighted(blur, 2.0, mask, -.5, 0)

img[smooth != 0] = [0, 0, 0]

crop_background = np.copy(
    config["background"][
        config["y"] : config["y"] + config["height"],
        config["x"] : config["x"] + config["width"],
    ]
)

crop_background[smooth == 0] = [0, 0, 0]

final_frame = crop_background + cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
final_frame = imgfilter(final_frame)

success, img = cv2.imencode('.jpg', final_frame)
return img.tostring()
```

 |

Green Screen, but better

- Actually really disorienting with a static background
- Angle of steering used to calculate how far to move crop position vs previous frame
- Scale variable to change perceived speed of panning

```
if (config["x"] + (data["user"]["angle"] * config["scale"])) < 2:
    config["x"] = int(config["background"].shape[1] / 2)
elif (config["x"] + (data["user"]["angle"] * config["scale"])) >= (int(config["background"].shape[1] - config["width"])):
    print(data["user"]["angle"] * config["scale"])
    config["x"] = int(config["background"].shape[1] / 2)
else:
    config["x"] = int(config["x"] + (data["user"]["angle"] * config["scale"]))
```

 |

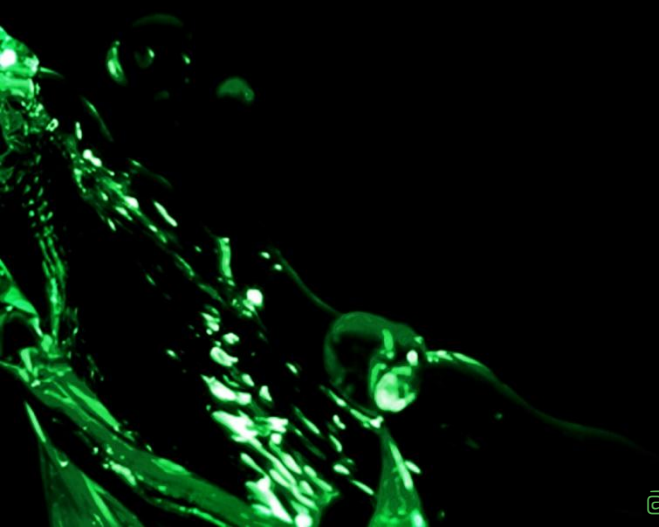
Automating the Training

- ZMQ proxy already has images + steering data
- TubWriter utility on NUC processes the data into usable format
- Data passed up to TensorFlow on GCP for training
- Around 10 minutes to train new model and make it available for the driver
- Still slow, but faster and way easier than manual

 |

And that's it!

ez clap



 |

JFROG IS A PROUD 2020 SD PYTHON SPONSOR

We are raffling off 2 Circuit Playground Expresses and 2 JFrog T-Shirts

SCAN

https://bit.ly/SDPYTHON

JFrog