

SNAKES ~~~~~
ON A CAR
~~~~~ **OR:**  
**Overengineering a Toy**





# Kat Cosgrove

Developer Advocate @ JFrog

Twitter: Dixie3Flatline

Shownotes: [bit.ly/DIDevOps](https://bit.ly/DIDevOps)

# AGENDA



**01**

## COMPANY

Here you could  
describe the topic of  
the section

**02**

## MARKET

Here you could  
describe the topic of  
the section

**03**


## TEAM

Here you could  
describe the topic of  
the section

**04**

## FUTURE

Here you could  
describe the topic of  
the section





**So, what's this demo?**



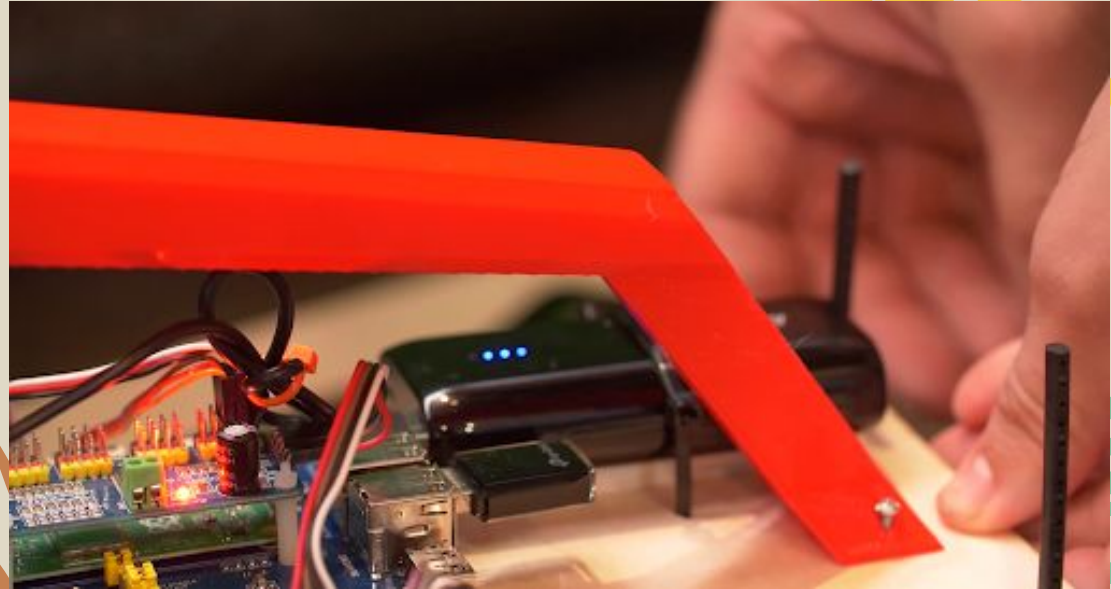


The background is a stylized tropical jungle scene. On the left, a tall palm tree with a green trunk and fronds stands against a light beige sky. The ground is a mix of green and brown patches, representing grass and soil. Various tropical plants are scattered throughout, including large green and yellow leaves, smaller green plants, and a red leaf on the left. The overall style is flat and modern.

# **Building the Basic Car**

# Donkey Cars

- About \$250 in Parts
- R/C Car
- Raspberry Pi 3B
- Pi Camera
- Battery Bank
- Train it
- Race it!

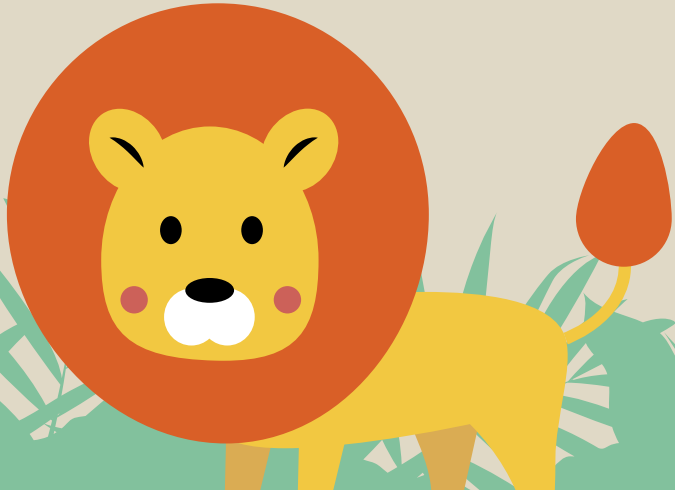








# Swapping the Controls



```
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)

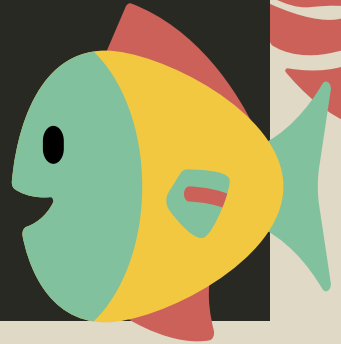
        topicfilter = b'donkeycar.racewheel'
        self.subscriber.setsockopt(zmq.SUBSCRIBE, topicfilter)
        self._lastimg = b''
```



```
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 frontend
- ZMQ Proxy
  - CI/CD
  - Racewheel data
  - Steering and Throttle
  - Image feed



# Adding Virtual Scenery





1. Read frame
2. Convert to HSV
3. Define HSV range
4. Create mask from range
5. Crop background
6. Merge them!
7. But wait...

```
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()
```

- Actually super 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 Training



```
class Trainer:
    def __init__(self):
        self.tub = None

    def new_writer(self, extra_path="default"):
        self.path = "./store/" + extra_path
        inputs = ["cam/image", "user/angle", "user/throttle", "user/mode", "timestamp"]
        types = ["image", "float", "float", "str", "str"]
        self.tub = TubWriter(self.path+"/tub", inputs=inputs, types=types)

    def store(self, data_set):
        if self.tub is None:
            self.new_writer()
        user_data = data_set["user"]
        img = Image.open(io.BytesIO(data_set["img"]))
        img = self._img_process(img)
        timestamp = str(datetime.datetime.utcnow())
        self.tub.run(img, user_data["angle"], user_data["throttle"], "user", timestamp)

    def publish(self, api_key):
        # REMOVE THIS NEXT LINE
        #self.path='store/8c77931f-968b-4be1-aa64-21fceb8ba8e'
        arc_path = shutil.make_archive(self.path + '/' + self.path.split('/')[-1], 'zip', self.path, 'tub')
        logger.info("Training archive created at %s", arc_path)
        upload_training(arc_path, api_key, 'trainer')
        logger.info('Training data uploaded')
```

# That's it!

## Your Speaker

Kat Cosgrove

## Win a Switch

[bit.ly/DIDevOps](https://bit.ly/DIDevOps)

## Twitter

Dixie3Flatline

## Credits

Freepik and Flaticon

