

# DREAM CATCHER

An Autonomous Robot that Dreams



---

## *Introduction*

### *Abstract*

A segment of Robotic Vision emulates processes that occur in the human brain. Creating and programing networks we are able to teach robots to identify simple shapes to more complex figures and see. Our neural networks are one of the most complex systems that we know of and our perception of reality is informed through what we see and experience everyday. Naturally inclined to see patterns in everyday the human mind warps reality to make sense of an abundance of stimuli. Processing during sleep, the mind dreams and conjures surreal images of constructs of everyday.

## *Concept*

We propose to run grant the robot the ability to not only navigate its environment but create its own interpretation of the visual information it is experiencing. Letting it run and then Dream. Finalizing its day (loop) by communicating its vision by displaying it in an interesting way before it goes and loops again.

We plan to achieve this visualization using DeepDream. DD is a computer vision program which uses a convolutional neural network to find and enhance patterns in images via algorithmic pareidolia.

We intend to explore a rudimentary process of visual processing and interpretation similar to the type of processing humans do which similar to DeepDream are informed through our experiences and learned information.

---


The Dream Catcher will enable robots to cope with the complexity of being an information-processing entity in domains that are open-ended both in terms of space and time. It paves the way for a new generation of robots whose existence and purpose goes far beyond the mere execution of dull tasks.



---

## Implementation Process

The Dream Catcher utilizes two main components; a Zumo 32U4 robot, which is based on the Arduino ATmega32U4 microcontroller, and a Raspberry-Pi with a camera module. The Zumo robot is programmed to use its infrared sensors and line sensor array to stay within the boundaries of tape placed on the ground meanwhile the Raspberry-Pi uses its camera to look for faces that resemble humans.



```
arduino_part | Arduino 1.6.0

/**
 * This example uses line sensors and infraed sensors for autonomous walking
 * in a restricted area with obstacles.
 * Rodger (Jieliang) Luo
 * Feb 9th, 2017
 */
#include <avr/pgmspace.h>
#include <Wire.h>
#include <Zumo32U4.h>

#define DIST_THRESHOLD 9 // brightness level, for infraed sensors
#define QTR_THRESHOLD 500 // microseconds, for line sensors

// These might need to be tuned for different motor types.
#define REVERSE_SPEED 100 // 0 is stopped, 400 is full speed
#define TURN_SPEED 200
#define FORWARD_SPEED 125
#define REVERSE_DURATION 300 // ms
#define TURN_DURATION 500 // ms

// Accelerometer Settings
#define RA_SIZE 3 // number of readings to include in running average of accelerometer readings
#define XY_ACCELERATION_THRESHOLD 2400 // for detection of contact (~16000 = magnitude of accelerati

Zumo32U4LCD lcd;
Zumo32U4ButtonA buttonA;
Zumo32U4Buzzer buzzer;
Zumo32U4Motors motors;
Zumo32U4LineSensors lineSensors;
Zumo32U4ProximitySensors proxSensors;

// Store this song in program space using the PROGMEM macro.
// Later we will play it directly from program space, bypassing
// the need to load it all into RAM first.
const char fugue[] PROGMEM =
```

```
python_part.py -- ~/Desktop/robotic-vision-master/Face_Sound/5_face_to_sound

python_part.py
8
9 import time
10 import io
11 import os
12 import sys
13 from cStringIO import StringIO
14 import random
15 import pipan
16 import picamera
17 import cv2
18 import numpy
19 import base64
20 import serial
21
22 ser = serial.Serial('/dev/ttyACM0', 9600)
23
24 find_face = False
25 serach_time = 0
26
27 def looking_face():
28
29     #Create a memory stream so photos doesn't need to be saved in a file
30     stream = io.BytesIO()
31
32     with picamera.PiCamera() as camera:
33         camera.resolution = (1280, 720)
34         camera.hflip = True
35         camera.vflip = True
36         camera.capture(stream, format='jpeg')
37
38     #Convert the picture into a numpy array
39     buff = numpy.fromstring(stream.getvalue(), dtype=numpy.uint8)
40
```

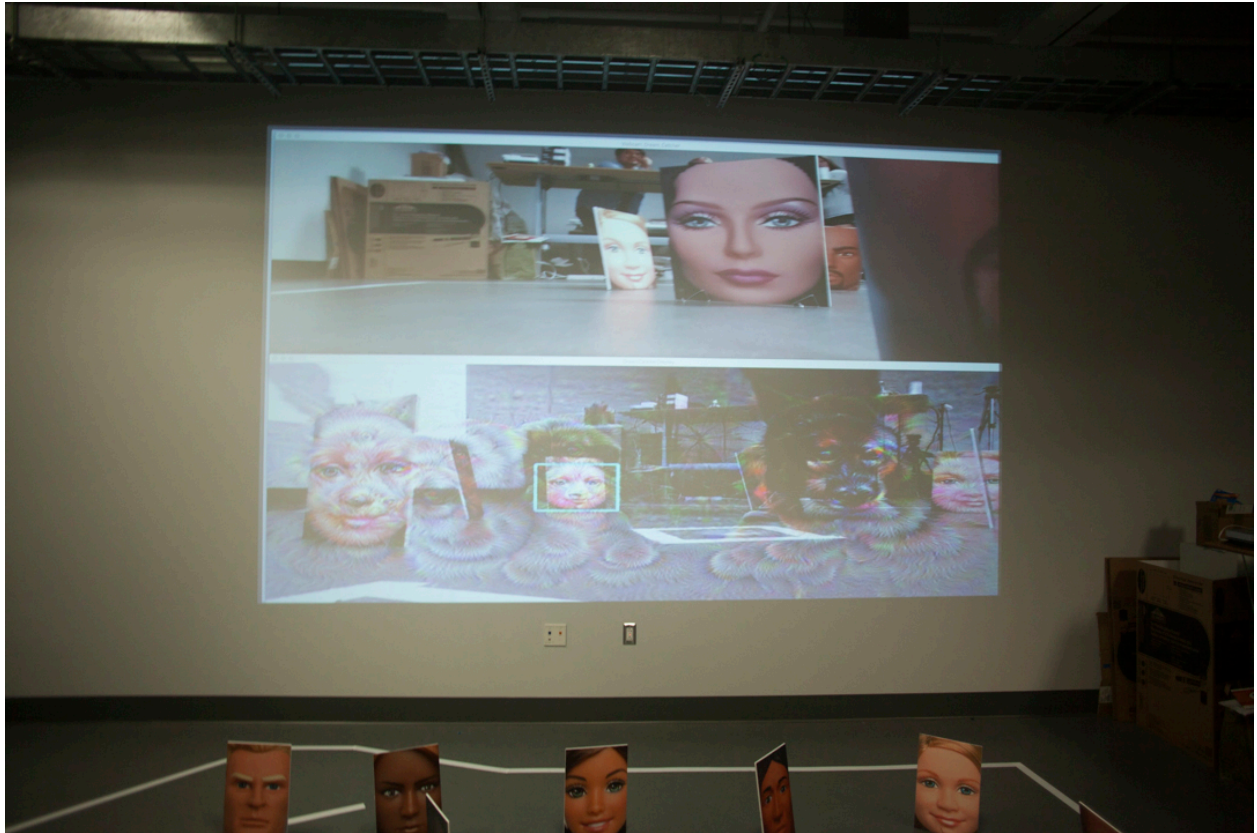
The Dream Catcher takes a picture every time it sees a face and sends it to a local laptop through the wireless network. Once the pictures are on the laptop, they are processed through the Deep Dream algorithm

```
deepdream2.py -- ~/Downloads

deepdream2.py
63 img = np.copy(img)
64 factors = (1, float(size[0]) / img.shape[1], float(size[1]) / img.shape[2], 1)
65 return scipy.ndimage.zoom(img, factors, order=1)
66
67 # Simple save function based on scipy
68 def save_image(img, fname):
69     pil_img = deprocess_image(np.copy(img))
70     scipy.misc.imsave(fname, pil_img)
71
72 def show_image(img, fmt='jpeg'):
73     img = deprocess_image(np.copy(img))
74     f = BytesIO()
75     PIL.Image.fromarray(img).save(f, fmt)
76     display(Image(data=f.getvalue()))
77
78 def deep_dream():
79
80     global index
81
82     # Dictionary of the names of the layers and the layers themselves
83     layer_dict = dict([(layer.name, layer) for layer in model.layers])
84
85     # Input of the first layer, in the example: model.input
86     dream = model.layers[0].input
87
88     # A TF variable (persistent)
89     loss = K.variable(0.)
90
91     # Iterate only over the layers picked above
92     for layer_name in settings['features']:
93         print(layer_name)
94         assert layer_name in layer_dict.keys(), 'Layer ' + layer_name + ' not found in model.' # Layer
95         coeff = settings['features'][layer_name] # Coefficient for the layer
```

---

A live stream of the Dream Catcher's point of view and the final images processed through Deep Dream is then projected onto the wall.<sup>3</sup>



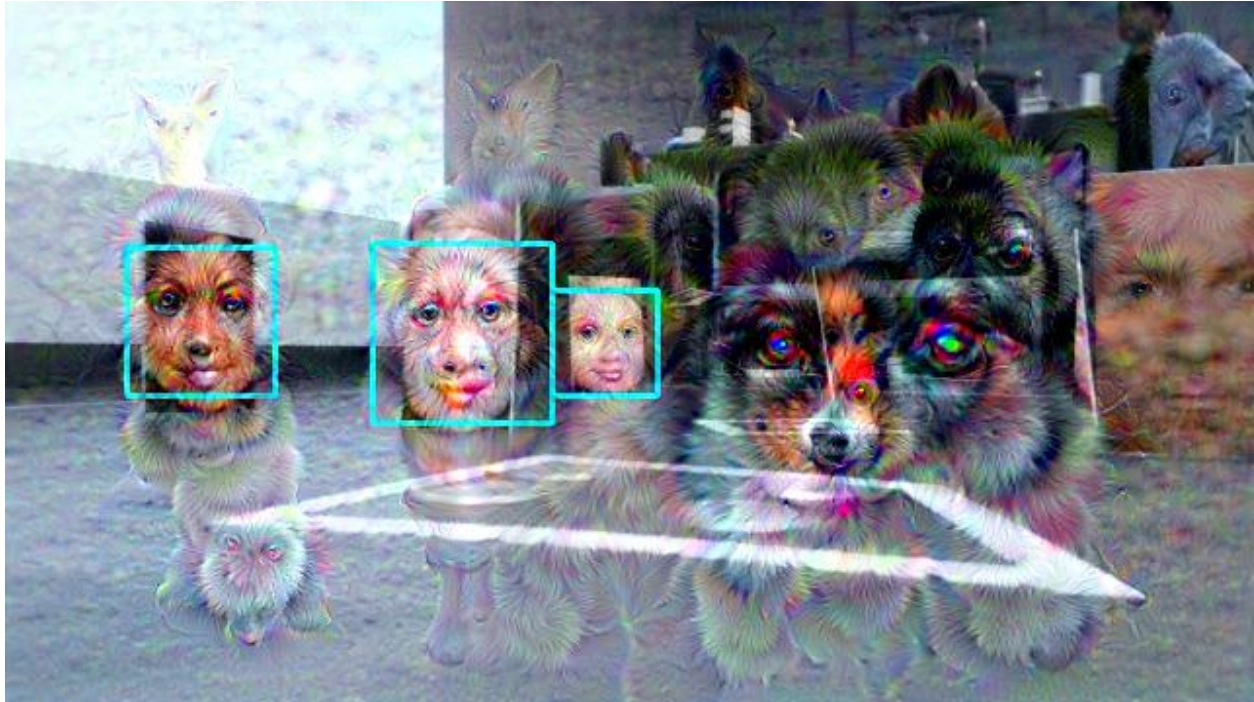


---

## Final Result



The Dream Catcher was able to do what we wanted in the end however we did have to use some alternative techniques than imagined to work. Originally we envisioned the Raspberry-Pi's camera to be able to live stream a video feed as well as take pictures once it recognizes faces but since we were running multiple python codes at the same time, there was a conflict. To have a working video feed of the Dream Catcher, we strapped a cell phone to the Zumo and used it's camera for video.



## Distribution of Labor

Adam – Explored Deep Dream code and optimized it for integration of project. Provided resources and time for prints. Explored alternative live stream resources. Worked on the presentation video.

Robert –Explored Processing and Display codes. Worked on the communication of the the projects' different components. Facilitated the development and direction of the project. Rendered proposal and also Report.

Chris – Coded for Zumo Arduion based behavior. Troubleshoot different codes throughout project. Supervised robot environment during presentation. Worked on Report.

## References

RodgerLuo (2018) UCSB Intelligent Machine Vision Course, (available at <https://github.com/RodgerLuo/robotic-vision#ucsb-intelligent-machine-vision-course>).

[https://github.com/RodgerLuo/robotic vision/tree/master/Face\\_Sound/5\\_face\\_to\\_sound](https://github.com/RodgerLuo/robotic-vision/tree/master/Face_Sound/5_face_to_sound)

DeepDream Code provided through

<https://github.com/google/deepdream>



---

[https://processing.org/reference/image\\_.html](https://processing.org/reference/image_.html)

Showing\_one\_image\_code // Processing code provided by Rodger Lou.

<https://github.com/onlylemi/processing-android-capture>