Code Listings

Matlab Visualization Tools

```
% filename: netscope.m
% author: Paul Adams

function netscope(varargin)
    % parse input args
    args = containers.Map(varargin(1:2:end), varargin(2:2:end));
    %socket_ip =char(py.socket.gethostbyname('thebes')));
    socket_ip = get_arg(args, 'socket_ip', 'thebes');
    sub_topic = get_arg(args, 'sub_topic', 'raw');
    v2db = @(x) 10*log10(x);
    dev = init_data_device(socket_ip, sub_topic);
    not_init = true;
    figure(1); clf; figure(2); clf;
    i_frame = 0;

    % main loop
    while true
        % update data
        data = dev.recv();
        idx = strfind(char(data), ';;;');
        header = data(1:idx);
        data = data(idx+3:end); % remove header
        shape = regexp(char(header), '.*n_row: (\d+).*', 'tokens');
        n_row = str2double(shape{1});
        if n_row > 10
            n_row = 1;
        end
        v = cell2mat(cell(py.list(py.numpy.fromstring(data))));
        v = reshape(v, length(v)/n_row, n_row).';

        if strcmp(sub_topic, 'filt') || strcmp(sub_topic, 'avg')
            v = v2db(v);
        end

        if not_init
            not_init = false;
            [x, wf, pl, img] = init_figures(v);
            n_frame = size(wf, 1);
        end

        % update waterfall
        i_frame = i_frame + n_row;
        if i_frame <=  n_frame
            idx = i_frame;
        else
            idx = n_frame;
            wf = circshift(wf, [-n_row, 0]);
        end
        wf((0:n_row - 1) + idx, 1:length(v)) = v;
        x(1:n_row, 1:length(v)) = v;
```

```
        %update plots
        %caxis([6, 30] + median(wf(:)));
        img.CData = wf;
        if length(pl(1).YData) < length(x)
            [x, wf, pl, img] = init_figures(v);
        end
        for i = 1:n_row
            try
                pl(i).YData = x(i, :);
            catch
            end
        end
        drawnow;
    end
end

function val = get_arg(args, key, default)
    if isKey(args, key)
        val = args(key);
    else
        val = default;
    end
end

function dev = init_data_device(socket_ip, sub_topic)
    tcp = sprintf('tcp://%s:5556', socket_ip);
    ctx = py.zmq.Context();
    dev = ctx.socket(py.zmq.SUB);
    dev.connect(tcp);
    dev.setsockopt(py.zmq.SUBSCRIBE, py.str(sub_topic));
end

function [x, wf, pl, img] = init_figures(v)
    n_frame = 500;

    % init figures
    f1 = figure(1);
    f1.WindowStyle = 'docked';
    t = (0:length(v) - 1)/48000;
    x = zeros(length(t), size(v, 1))';
    pl = plot(t*1000, v, '.-');
    grid on;
    xlabel('time [ms]')

    f2 = figure(2);
    f2.WindowStyle = 'docked';
    wf = zeros(n_frame, length(v));

    wf((0:size(v, 1) - 1) + 1, :) = v;
    %img = imagr(wf, 'fignum', 5);
    img = imagesc(wf);
    colorbar; colormap hot
    xlabel('sample')
```

```
    ylabel('time')
    %caxis([-20, 20])
    ui_init();
end

function cb_clim(src, evnt)
    obj = src.Parent.Children(end - 1);
    cur_value = obj.Limits;
    dval = 1;
    if strcmp(src.Tag, 'nf_up')
        new_val = cur_value + dval;
    elseif strcmp(src.Tag, 'nf_dn')
        new_val = cur_value - dval;
    elseif strcmp(src.Tag, 'cal')
        new_val = [3, 30] + median(src.Parent.Children(end).Children(1).CData(:));
    elseif strcmp(src.Tag, 'dr_up')
        new_val(1) = cur_value(1) - dval;
        new_val(2) = cur_value(2) + dval;
    elseif strcmp(src.Tag, 'dr_dn')
        new_val(1) = cur_value(1) + dval;
        new_val(2) = cur_value(2) - dval;
    elseif strcmp(src.Tag, 'dr')
        new_val = cur_value*src.Value;
    elseif strcmp(src.Tag, 'nf')
        new_val = cur_value + src.Value;
    end
    obj.Limits = new_val;
    caxis(new_val);
end

function ui_init()
    cb = colorbar();

    x0 = cb.Position(1);
    dx = cb.Position(3);
    y0 = cb.Position(2);
    dy = cb.Position(4);

    uicontrol('Style', 'text', 'string', 'NF', ...
        'units', 'normalized',...
        'position', [x0 + 1.4*dx, y0, dx, dx], ...
        'Fontweight', 'bold', 'fontsize', 14);

    uicontrol('Style', 'pushbutton', 'string', 'cal',...
        'units', 'normalized',...
        'position', [x0 + 1.2*dx, y0 + 5*dx, dx, dx], ...
        'callback', @cb_clim, 'tag', 'cal');

    uicontrol('Style', 'pushbutton', 'string', '+', ...
        'units', 'normalized',...
        'position', [x0 + 1.2*dx, y0 + dx, dx, dx], ...
        'callback', @cb_clim, 'tag', 'nf_up');

    uicontrol('Style', 'pushbutton', 'string', '-', ...
```

```
        'units', 'normalized',...
        'position', [x0 + 2.4*dx, y0 + dx, dx, dx], ...
        'callback', @cb_clim, 'tag', 'nf_dn');

    uicontrol('Style', 'text', 'string', 'DR', ...
        'units', 'normalized',...
        'position', [x0 + 1.4*dx, y0 + dy - dx, dx, dx], ...
        'Fontweight', 'bold', 'fontsize', 14);

    uicontrol('Style', 'pushbutton', 'string', '+', ...
        'units', 'normalized',...
        'position', [x0 + 1.2*dx, y0 + dy - 2*dx, dx, dx], ...
        'callback', @cb_clim, 'tag', 'dr_up');

    uicontrol('Style', 'pushbutton', 'string', '-',...
        'units', 'normalized',...
        'position', [x0 + 2.4*dx, y0 + dy - 2*dx, dx, dx], ...
        'callback', @cb_clim, 'tag', 'dr_dn');
end
```

Matlab Algorithm Protyping

```
% filename: batch_fmcw_detection.m
% author: Paul Adams
%
function x = batch_fmcw_detection(varargin)
    % init and params
    dbv = @(x) 20*log10(abs(x));
    fname = dir('*radar_data*');
    [~, idx] = max([fname.datenum]);
    fname = fname(idx).name;

    % parse input args
    args = containers.Map(varargin(1:2:end), varargin(2:2:end));
    oversample = get_arg(args, 'oversample', 5);
    window_func = get_arg(args, 'window_func', @rectwin);
    fname = get_arg(args, 'fname', fname);
    detrend_flag = get_arg(args, 'detrend_flag', false);

    sync_chan = 1;
    signal_chan = 2;

    % fetch data
    [x, fs] = audioread(fname);

    % sync pulses
    [pulse_idx, n_samp_pulse] = pulse_sync(x, sync_chan);

    % derived params
    n_pulse = length(pulse_idx) - 1;
    N = size(x, 1);
    t = (0:N - 1)/fs;
    n_fft = 2^nextpow2(n_samp_pulse)*8;
    f = (0:n_fft/2-1)*fs/n_fft;
    taper = repmat(window(window_func, n_samp_pulse)', n_pulse, 1);
    pulse_proto = 1 - abs(linspace(-1, 1, n_samp_pulse));

    %radar parameters
    c = 3E8; %(m/s) speed of light
    fstart = 2260E6; %(Hz) LFM start frequency for example
    fstop = 2590E6; %(Hz) LFM stop frequency for example
    BW = fstop-fstart; %(Hz) transmti bandwidth

    %range resolution
    rr = c/(2*BW);
    max_range = rr*n_samp_pulse/2;
    r = linspace(0, max_range, n_fft);

    % form pulse matrix (optionally detrend)
    y = zeros(n_pulse, n_samp_pulse);
    for i = 1:n_pulse
        pulse = x((1:n_samp_pulse) + pulse_idx(i), signal_chan)';
        y(i, :) = pulse - detrend_flag*mean(pulse);
    end
```

```
    % clutter suppression
    dy = [zeros(1, n_samp_pulse); y(1:n_pulse - 1, :)];
    dyy = [zeros(2, n_samp_pulse); y(1:n_pulse - 2, :)];

    z2 = y - dy; % 2-pulse canceller
    z3 = y - 2*dy + dyy; % 3-pulse canceller

    if 1
        % Taper, FFT, Power
        Z1 = fft(y.*taper, n_fft, 2);
        Z1= dbv(Z1(:, 1:n_fft/2));

        im = imagr(Z1, 'x', r, 'y', t, 'fignum', 101, 'colormap', 'hot');
        xlabel('range bin');
        ylabel('time');
        %%xlim([0, 50]);
        title('Range Time Indicator - No clutter suppresion');

        % Taper, FFT, Power
        Z2 = fft(z2.*taper, n_fft, 2);
        Z2= dbv(Z2(:, 1:n_fft/2));

        im = imagr(Z2, 'x', r, 'y', t, 'fignum', 102, 'colormap', 'hot');
        xlabel('range bin');
        ylabel('time');
        %xlim([0, 50]);
        title('Range Time Indicator - 2-Pulse Canceller');
    end

    % Taper, FFT, Power
    Z3 = fft(z3.*taper, n_fft, 2);
    Z3= dbv(Z3(:, 1:n_fft/2));
    %Z3 = dbv(Z3);

    im = imagr(Z3, 'x', r, 'y', t, 'fignum', 103, 'colormap', 'hot');
    xlabel('range bin');
    ylabel('time');
    %xlim([0, 50]);
    title('Range Time Indicator - 3-Pulse Canceller');

    [~, range_gates] = max(Z3, [], 2);
    figure(3)
    plot(r(range_gates), t(pulse_idx(1:end-1)), 'o')
    xlabel('range [m]'); ylabel('time')
    %xlim([0, 50]);
    grid on;
    ax = gca;
    ax.YAxis.Direction = 'reverse';
    title(fname);
end

function [pulse_idx, n_samp_pulse] = pulse_sync(x, sync_chan)
    % perform pulse sync
```

```
    a = x(:, sync_chan) > 0; % square wave
    b = diff([0; a]) > 0.5; % true on rising edges
    c = diff([0; a]) < - 0.5; % true on falling edges

    rise_idx = find(b);
    fall_idx = find(c);
    n = min(length(rise_idx), length(fall_idx));
    ramp_idx = fall_idx(1:n) - rise_idx(1:n);

    % guaranteed shortest ramp time so we don't accidentally integrate over boundaries
    n_samp_pulse = min(ramp_idx);
    pulse_idx = rise_idx;
end

function val = get_arg(args, key, default)
    if isKey(args, key)
        val = args(key);
    else
        val = default;
    end
end
```

```
% filename: DopplerConfig.m
% author: Paul Adams

classdef DopplerConfig < handle
    properties
        ax
        state
        wav
        is_file
        n_total
        n_samp
        Tp
        oversample
        n_dwell_detect
        taper
        i_samp
        i_chan
        i_dwell
        n_filt
        v_mph
        funcs
        max_filt
        debug_level
        noise_est
        alpha_n
        beta_n
    end

    properties(Constant)
        MPH_CF = 2.23694;
        FC = 2590E6; %(Hz) Center frequency (connected VCO Vtune to +5 for example)
        C_LIGHT = 3e8; %(m/s) speed of light
```

```
        AUDIO_DEVICE_ID = 2;
        FS = 44100;
        % Thresholds
        pass_thresh = 0.5; % instantaneous noise level indicating vehicle crossing
        actv_thresh = 28; % peak must be X dB above noise est
        v_max_mph = 4000;% suppress speeds above 50 mph
    end

    methods
        function me = DopplerConfig(args)
            taper_func = eval(sprintf('@%swin', args('windowing_function')));
            me.debug_level = args('debug_level');

            me.Tp = args('dwell_period_ms')/1000;
            me.oversample = args('oversample_factor');
            me.n_dwell_detect = args('n_detection_dwell');
            me.i_samp = 0;
            me.i_dwell = 0;
            me.wav = args('wav_file');
            me.i_chan = args('channel_vector');
            if ~isempty(me.wav)
                I = audioinfo(me.wav);
                me.n_total = I.TotalSamples;
                me.FS = I.SampleRate;
                me.is_file = true;
            else
                me.n_total = inf;
                me.is_file = false;
                me.wav = audiorecorder(me.FS, 16, (me.i_chan), me.AUDIO_DEVICE_ID);
                me.wav.record(2*me.Tp);
            end
            me.noise_est = [];
            me.n_samp = round(me.Tp*me.FS);
            me.Tp = me.n_samp/me.FS;
            me.taper = window(taper_func, me.n_samp).';
            me.n_filt = me.oversample*2^nextpow2(me.n_samp);

            % State
            me.state.real_time = args('real_time');
            me.state.Active = false; % indicates there is moving object in frame,
            % assuming peak value is x dB above noise
            me.state.Passing = false; % indicates an object is passing the radar,
            % assumes instantaneous noise level is x dB above historical value
            me.state.VehicleCount = 0;

            % calculate velocity
            lambda = me.C_LIGHT/me.FC;
            df = me.FS/me.n_filt;
            doppler = 0:df:me.FS/2 - df; % doppler spectrum
            me.funcs.dop2mph = @(dop) me.MPH_CF*dop*lambda/2;
            me.funcs.mph2dop = @(mph) 2*mph/(me.MPH_CF*lambda);
            me.v_mph = me.MPH_CF*doppler*lambda/2;
            max_doppler = 2*me.v_max_mph/(me.MPH_CF*lambda);
            [~, me.max_filt] = min(abs(doppler - max_doppler));
```

```
            % iir filter coeffs
            % noise filter
            me.alpha_n = 0.90; % historical value weighting
            me.beta_n = 1 - me.alpha_n; % current measurement weighting

            plot(me.v_mph, me.v_mph, '.-'); hold on;
            plot(me.v_mph(1), 1, 'r*'); hold off;
            xlim([0, 200])
            ylim([-50, 50])
            xlabel('velocity [mph]'); ylabel('dB'); title('Audio Spectrum');
            me.ax = gca;
            grid on;
        end
    end
end
```

```
% filename: event_doppler.m
% author: Paul Adams
%
function dc = event_doppler(varargin)
    % config
    args = containers.Map(varargin(1:2:end), varargin(2:2:end));
    dc = DopplerConfig(args);
    dc.wav.record(dc.n_dwell_detect*dc.Tp);

    % Loop until EOF
    eof = false;
    while ~eof
        tic;
        x = fetch_and_format(dc);
        [v, i, n0] = reduce_doppler(x, dc);
        update_state(v, i, n0, dc);

        % Check for EOF and enforce real-time
        eof = dc.i_samp >= dc.n_total - dc.n_dwell_detect*dc.n_samp; % within one dwell of eof

        show_state(dc, v, n0);

        margin = dc.Tp*dc.n_dwell_detect - toc;
        if args('real_time')
            pause(margin);
        else
            drawnow
        end
    end
end

function x = fetch_and_format(dc)
    if dc.is_file
        start_samp = dc.i_samp + 1;
        dc.i_samp = start_samp + dc.n_samp*dc.n_dwell_detect - 1;
        if dc.debug_level >= 2; fprintf('Fetch and Format... samples: %d to %d...', start_samp, dc.i_sa
        [v, ~] = audioread(dc.wav, [start_samp, dc.i_samp]);
```

```
    else
        while dc.wav.get('TotalSamples') < dc.n_samp*dc.n_dwell_detect; end
        v = dc.wav.getaudiodata();
        dc.wav.record(dc.n_dwell_detect*dc.Tp);
    end
    x = reshape(v(:, dc.i_chan)', dc.n_samp, dc.n_dwell_detect).'; % let each row have n_samp samples
end

function [v, i, n0] = reduce_doppler(x, dc)
    if dc.debug_level >= 2; fprintf('Doppler Process... \n'); end
    x_w = bsxfun(@times, x, dc.taper); % apply taper to each row
    X = fft(x_w, dc.n_filt, 2); % fft along rows
    %X = 20*log10(abs(X(:, 1:dc.max_filt)));
    X = 20*log10(abs(X(:, 1:dc.n_filt/2)));
    n0 = mean(X(:));
    [v, i] = max(mean(X, 1)); % take mean over rows, then select the max filter bin
    dc.ax.Children(2).YData = mean(X, 1);
    dc.ax.Children(1).XData = dc.v_mph(i);
    dc.ax.Children(1).YData = v;
end

function update_state(v, i, n0, dc)
    if isempty(dc.noise_est)
        dc.noise_est = n0; % initial value
    else
        % current noise est. is weight last est. + weighted current measurement
        dc.noise_est = dc.alpha_n*dc.noise_est + dc.beta_n*n0;
    end

    dc.state.SpeedEst = nan;
    if n0 > dc.noise_est + dc.pass_thresh
        if ~dc.state.Passing % if changing state, inc. counter
            dc.state.VehicleCount = dc.state.VehicleCount + 1;
        end
        dc.state.Passing = true;
    elseif v > dc.noise_est + dc.actv_thresh
        dc.state.Passing = false; % turn off Passing
        dc.state.Active = true;
        dc.state.SpeedEst = dc.v_mph(i);
    end
    if v <= dc.noise_est + dc.actv_thresh; dc.state.Active = false; end

    % update debug variables
    if dc.debug_level >= 1
        dc.i_dwell = dc.i_dwell + 1;
        dc.state.n_i(dc.i_dwell) = n0;
        dc.state.n_a(dc.i_dwell) = dc.noise_est;
        dc.state.v_mph(dc.i_dwell) = dc.state.SpeedEst;
    end
end

function show_state(dc, v, n0)
    fprintf('Time: %.2f\tnoise: %.2f\tpower: %.2f\tPeak: %.2f\tPassing: %d\tActive: %d\tSpeed: %0.1f\tVe
        dc.i_samp/dc.FS, dc.noise_est, n0, v, dc.state.Passing, dc.state.Active, round(dc.state.SpeedEst
```

```
end
```

Python Main Programs

```
#!/usr/bin/python2
# -*- coding: utf-8 -*-
# __file__ sdr_main.py
# __author__ Paul Adams

import sys

MODE = sys.argv[1]

def main():
    if MODE == 'fmcw':
        import fmcw_serv
        fmcw_serv.main()
    elif MODE == 'dopp':
        import dopp_serv
        dopp_serv.main()

if __name__ == '__main__':
    main()
```

```
#!/usr/bin/python2
# -*- coding: utf-8 -*-
# __file__ sdr.py
# __author__ Paul Adams

# third-party imports
import numpy as np
import socket
from scipy.signal import lfilter
import zmq
import time

SYNC_CHAN = 0
SGNL_CHAN = 1
N_CHAN = 2
SUB_PORT = 5555
PUB_PORT = 5556
lpf_b = np.fromfile('/home/paul/ee542/lpf.npy')

class Zmq():
    def __init__(self):
        # setup zmq
        self.ctx = zmq.Context()
        self.sub = self.ctx.socket(zmq.SUB)
        self.sub.setsockopt(zmq.SUBSCRIBE, 'pcm_raw')
        self.ip = socket.gethostbyname('thebes')
        tcp = "tcp://%s:%s" % (self.ip, SUB_PORT)
        self.sub.connect(tcp)
        print 'SDR: Listening on %s' % tcp

        self.pub = self.ctx.socket(zmq.PUB)
        tcp = "tcp://%s:%s" % (self.ip, PUB_PORT)
```

```
        self.pub.bind(tcp)
        print 'SDR: Publishing on %s' % tcp

class Queue():
    def __init__(self):
        self.buff_idx = 0
        self.ref = []
        self.sig = []
        self.raw = []
        self.n_buff =  1
        self.time = []

    def re_init(self):
        self.buff_idx = 0
        self.ref = []
        self.sig = []

    def fetch_format(self):
        # fetch format data
        data = z.sub.recv()
        idx = data[0:100].find(';;;')
        header = data[0:idx]
        self.time = float(header[header.find(':')+1:header.find(';;;')])
        #import pdb; pdb.set_trace()
        x = (np.fromstring(data[idx+3::], np.int32)).astype(np.float)/2**31
        self.sig = x[SGNL_CHAN::2]
        self.ref = x[SYNC_CHAN::2]
        debug_hook(self.ref, 'clock')
        debug_hook(self.sig, 'signal')

    def update_buff(self):
        self.buff_idx += 1
        self.fetch_format()

    def is_full(self):
        return self.buff_idx == self.n_buff

def debug_hook(data, topic):
    n_row = data.shape[0]
    z.pub.send('%s n_row: %s;;;%s' % (topic, str(n_row), data.tostring()))

def lowpass(x):
    if len(x.shape) == 1:
        axis = 0
    else:
        axis = 1

    return lfilter(lpf_b, 1, x, axis=axis)

def fft_filter(x, n_fft):
    if len(x.shape) == 1:
        x = np.abs(np.fft.fft(x, n=n_fft)[0:n_fft/2])**2
    else:
        x = np.abs(np.fft.fft(x, n=n_fft)[:, 0:n_fft/2])**2
```

```
    debug_hook(x, 'filt')
    return x

def averager(x):
    x = np.mean(x, axis=0)
    debug_hook(x, 'avg')
    return x

z = Zmq()
```

```
#!/usr/bin/python2
# -*- coding: utf-8 -*-
# __file__ serv-alsa.py
# __author__ Paul Adams

# third-party imports
import socket
import pyaudio
import zmq
import time
import sys

# local imports
N_SAMP = int(sys.argv[1])
N_SAMP_BUFF = 10*N_SAMP
N_CHAN = 2
FS = 48000
PUB_PORT = 5555
pa = pyaudio.PyAudio()

# setup zmq
ctx = zmq.Context()
pub = ctx.socket(zmq.PUB)
tcp = "tcp://%s:%s" % (socket.gethostbyname('thebes'), PUB_PORT)
pub.bind(tcp)
print 'ALSA: Publishing on %s' % tcp

def alsa_callback(data, frames, time, status):
    pub.send('%s %s;;;%s' % ('pcm_raw', 'time:%f' % (time['current_time']), data))
    return (data, pyaudio.paContinue)

class Alsa():
    def __init__(self):
        self.stream = pa.open(format=pyaudio.paInt32,
                rate=FS, input=True, channels=N_CHAN,
                frames_per_buffer=N_SAMP_BUFF,
                stream_callback=alsa_callback)

    def loop(self):
        while self.stream.is_active():
            time.sleep(0.01)

        # stop stream
```

```
        self.stream.stop_stream()
        self.stream.close()
        pa.terminate()

def main():
    a = Alsa()
    a.loop()

if __name__ == '__main__':
    main()
```

```
#!/usr/bin/python2
# -*- coding: utf-8 -*-
# __file__ serv-fmcw.py
# __author__ Paul Adams

# third-party imports
import numpy as np
import time
import json

# local imports
import sdr

# constants
M2FT = 3.28084
C_LIGHT = 3e8
BW = 300e6
N_FFT = 4096
FS = 48000
MAX_PERIOD_DELTA = 50


class Sync():
    def __init__(self):
        self.have_period = False
        self.period = []
        self.edges = {}
        self.T = []
        self.pulses ={}
        self.tail = {}
        self.tail['ref'] = np.zeros(2)
        self.head = {}
        self.stable_period_count = 0

    def get_edges(self, q):
        dref = np.diff((q.ref > 0).astype(np.float))
        # find indices of rising edges
        self.edges['rise'] = np.where(dref == -1)[0]

        # find indices of falling edges
        self.edges['fall'] = np.where(dref == +1)[0]

    def align_edges(self, q):
```

```
        # make sure fall follows rise, save head
        head_idx = np.argmax(self.edges['fall'] > self.edges['rise'][0])
        self.edges['fall'] = self.edges['fall'][head_idx::]
        head_idx = self.edges['rise'][0]

        # make sure each vector is equi-length
        if len(self.edges['rise']) > len(self.edges['fall']):
            self.edges['rise'] = self.edges['rise'][0:len(self.edges['fall'])]
        else:
            self.edges['fall'] = self.edges['fall'][0:len(self.edges['rise'])]

        # try stitch previous tail to current head
        self.head['ref'] = q.ref[0:head_idx]
        self.head['sig'] = q.sig[0:head_idx]
        self.stitch()
        self.tail['ref'] = q.ref[self.edges['fall'][-1]::]
        self.tail['sig'] = q.sig[self.edges['fall'][-1]::]

    def check_period(self):
        if self.period:
            prev_period = self.period
        else:
            prev_period = 0

        self.period = np.floor(np.mean(self.edges['fall'] - self.edges['rise'])).astype(np.int16)
        rez = np.abs(self.period - prev_period)

        if rez < MAX_PERIOD_DELTA:
            self.stable_period_count += 1
            if not self.have_period:
                print 'pulse period acquired --> %d samples' % (self.period)
                self.have_period = True
                self.T = self.period*FS

        elif self.have_period:
            self.stable_period_count = 0
            self.have_period = False
            self.period = []
            print 'pulse period lost. residual --> %d samples' % (rez)

    def stitch(self):
        if self.tail['ref'].any():
            x = np.hstack((self.tail['ref'], self.head['ref']))
            y = np.hstack((self.tail['sig'], self.head['sig']))

            # sync clock signal
            dx = np.diff((x > 0).astype(np.float))

            # find indices of rising edges
            rise = np.where(dx == -1)[0].tolist()

            while rise:
                r = rise.pop(0)
                if self.period.__class__ is list:
```

```
                    pass
                else:
                    # import pdb; pdb.set_trace()
                    ts = float(r)/self.period*(q.time - 1)
                    self.pulses[ts] = y[r:r+self.period]

    # given a buffer of audio frames, find the pulses within the clock signal and extract received chir
    def extract_pulses(self, sig):
        rises = self.edges['rise'].tolist()
        while rises:
            idx = rises.pop(0)
            ts = float(idx)/self.period*q.time
            # import pdb; pdb.set_trace()
            self.pulses[ts] = sig[idx:idx + self.period]

class Processor():
    def __init__(self):
        self.x = 0
        self.detects = []
        self.range_rez = C_LIGHT/(2*BW)
        self.n_samp = 0
        self.report_dict = {}
        self.range_lu = np.zeros((1, 1))
        self.ranges = []
        self.prior = []
        self.string = ''
        self.t0 = 0

    def format(self, pulses):
        self.n_samp = min([len(p) for p in pulses.values()])
        self.x = np.zeros((len(pulses) , self.n_samp))
        k = pulses.keys()
        k.sort()
        for i, j in enumerate(k):
            self.x[i, :] = pulses[j][0:self.n_samp]

        sdr.debug_hook(self.x, 'raw')

    def canceller(self):
        (nrow, nsamp) = self.x.shape
        try:
            if len(self.prior) > 0:
                if len(self.prior) != nsamp:
                    nsamp = min([len(self.prior), self.x.shape[1]])
                    self.x = self.x[:, 0:nsamp]
                    self.prior = self.prior[0:nsamp]
            else:
                self.prior = np.zeros(nsamp)
        except:
            import pdb; pdb.set_trace()

        y = self.x.copy()
        dy = np.vstack((self.prior, y[0:nrow-1, :]))
        self.x = y - dy
```

```
        self.prior = y[nrow-1, :]

    def detect(self):
        #cfar = signal.lfilter(self.cfar_filt, 1, self.x)
        self.detects = [np.argmax(self.x[50:-1]) + 50]

    def transform(self):
        self.ranges = []
        if not self.range_lu.any():
            max_range = self.range_rez*self.n_samp/2
            self.range_lu = np.linspace(0, max_range, N_FFT/2)

        if len(self.detects) > 0:
            for d in self.detects:
                self.ranges.append(self.range_lu[d])

    def report(self):
        self.report_dict = {}
        for i in range(len(self.detects)):
            ts = time.time()
            self.report_dict[ts] = {}
            self.report_dict[ts]['gate'] = self.detects[i]
            self.report_dict[ts]['range'] = self.ranges[i]

        self.report_str = json.dumps(self.report_dict)

    def print_debug(self, doit):
        dt = q.time - self.t0
        self.t0 = q.time
        self.string += 'time: %.3f -> dt: %.3f ms -> pulses: %d -> samp: %d stable -> %d -> detect: %d r
        if doit:
            print self.string,
            self.string = ''

    def process_pulses(self, pulses):
        self.format(pulses)
        self.x = sdr.lowpass(self.x)
        self.canceller()
        self.x = sdr.fft_filter(self.x, N_FFT)
        self.x = sdr.averager(self.x)
        self.detect()
        self.transform()
        self.report()

# init objects
q = sdr.Queue()
s = Sync()
p = Processor()

def main():
    print 'RUN_FMCW: Queue and Sync initzd... Entering loop now... '

    while True:
        q.update_buff()
```

```
        if q.is_full():
            s.get_edges(q)
            s.align_edges(q)
            s.check_period()

            if s.have_period:
                s.extract_pulses(q.sig)
                p.process_pulses(s.pulses)
                p.print_debug(True)
                s.pulses = {} # reset pulses
            else:
                pass

            q.re_init()

if __name__ == '__main__':
    main()
```

```
#!/usr/bin/python2
# -*- coding: utf-8 -*-
# __file__ dopp-serv.py
# __author__ Paul Adams

# third-party imports
import numpy as np
from scipy.signal import hanning
import time
import json

# local imports
import sdr

# constants
M2FT = 3.28084
C_LIGHT = 3e8
BW = 300e6
N_FFT = 4096*4
FS = 48000

class Processor():
    def __init__(self):
        self.x = 0
        self.detects = []
        self.n_samp = 0
        self.report_dict = {}
        self.string = ''
        self.taper = np.zeros(1)
        self.t0 = 0

    def format(self):
        if not self.taper.any():
            self.taper = hanning(q.sig.shape[0])
        self.x = np.multiply(q.sig, self.taper)
```

```
        sdr.debug_hook(self.x, 'raw')

    def detect(self):
        self.detects = [np.argmax(self.x[50:-1]) + 50]

    def print_debug(self, doit):
        dt = q.time - self.t0
        self.t0 = q.time
        self.string += 'time: %.3f -> dt: %.3f ms -> detect: %d m\n' % (self.t0, dt*1e3, self.detects[0]
        if doit:
            print self.string,
            self.string = ''

    def process_pulses(self):
        self.format()
        self.x = sdr.lowpass(self.x)
        self.x = sdr.fft_filter(self.x, N_FFT)
        self.x -= np.mean(self.x)
        self.detect()

# init objects
q = sdr.Queue()
p = Processor()

def main():
    print 'RUN_DOPP: Queue initzd... Entering loop now... '

    while True:
        q.update_buff()

        if q.is_full():
            p.process_pulses()
            p.print_debug(True)
            q.re_init()

if __name__ == '__main__':
    main()
```

Shell Scripts and systemd Modules

```
#!/bin/bash
# file: eth0-startup.sh
# author: Paul Adams

carrier_state=$(</sys/class/net/eth0/carrier)
not_up=1

while [ "$not_up" -eq 1 ]; do
    if [ "$carrier_state" -eq 1 ]; then
        ip addr add 192.168.2.108/255.255.255.0 dev eth0
        ip link set eth0 up
        not_up=0
        echo "success"
    else
        echo "ethernet not connected foo"
        sleep 10
    fi
done

echo "exit program"
```

```
#!/bin/bash
# file: kill-all.sh
# author: Paul Adams

pgrep python2 | xargs kill -9
pgrep jackd | xargs kill -9
```

```
#!/bin/bash
# file: start-all.sh
# author: Paul Adams

/usr/bin/jackd -P70 -p16 -t2000 -d alsa -d hw:0 -p 128 -n 3 -r 48000 -s 2>/dev/null&
echo 'started jackd...'
sleep 1
/usr/bin/python2 /home/paul/ee542/alsa_serv.py $1 2>/dev/null&
# /usr/bin/python2 /home/paul/ee542/alsa_serv.py $1 &
echo "started serv-alsa.py N_SAMP=$1"
sleep 1
/usr/bin/python2 /home/paul/ee542/sdr_main.py $2
```

```
[Unit]
Description=my jackd management wrapper
Before=serv-alsa.service serv-fmcw.service

[Service]
ExecStart=/usr/bin/jackd -P70 -p16 -t2000 -d alsa -d hw:0 -p 128 -n 3 -r 48000 -s
Type=simple
Restart=always
RestartSec=15s

[Install]
WantedBy=multi-user.target
```

```
[Unit]
Description=my fmcw server wrapper
After=serv-alsa.service jackd.service
Requires=jackd.service serv-alsa.service

[Service]
Environment=PYTHONBUFFERED=true
ExecStart=/usr/bin/python2 /home/paul/ee542/serv-fmcw.py
Type=simple
Restart=always
RestartSec=15s

[Install]
WantedBy=multi-user.target
```

```
[Unit]
Description=my alsa server wrapper
After=jackd.service
Before=serv-fmcw.service
Requires=jackd.service

[Service]
Environment=PYTHONBUFFERED=true
ExecStart=/usr/bin/python2 /home/paul/ee542/serv-alsa.py
Type=simple
Restart=always
RestartSec=15s

[Install]
WantedBy=multi-user.target
```