Direction of Arrival Analysis on a Mobile Platform

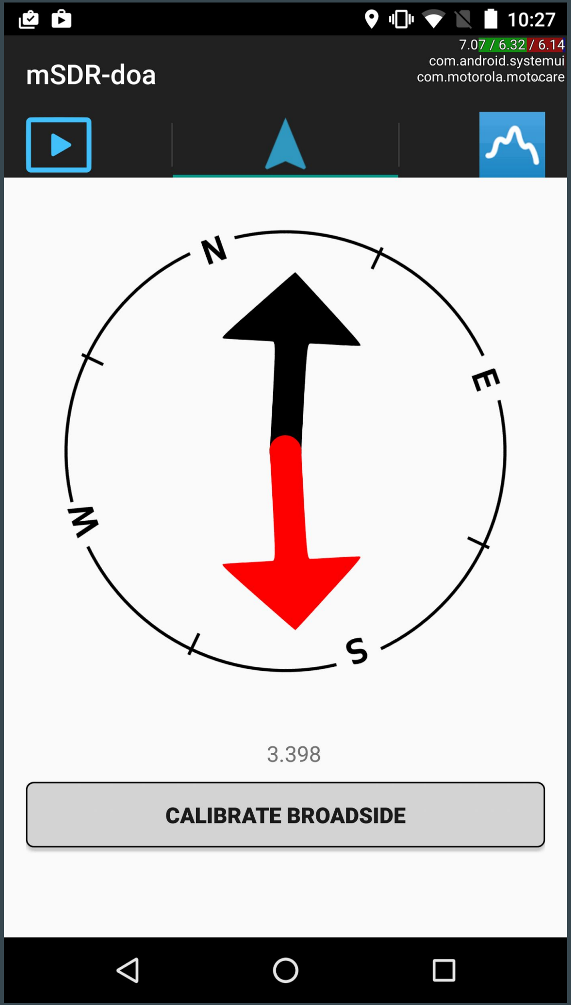
•••

Sam Whiting, Dana Sorensen, Todd Moon
Utah State University

Objectives

- Find a transmitter

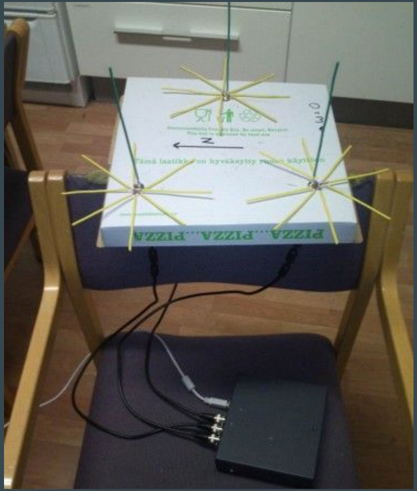
- Be mobile



Previous Work

Tatu Peltola - 3 RTL dongles

https://www.youtube.com/watch?v=8Wzb1mgZ0EE

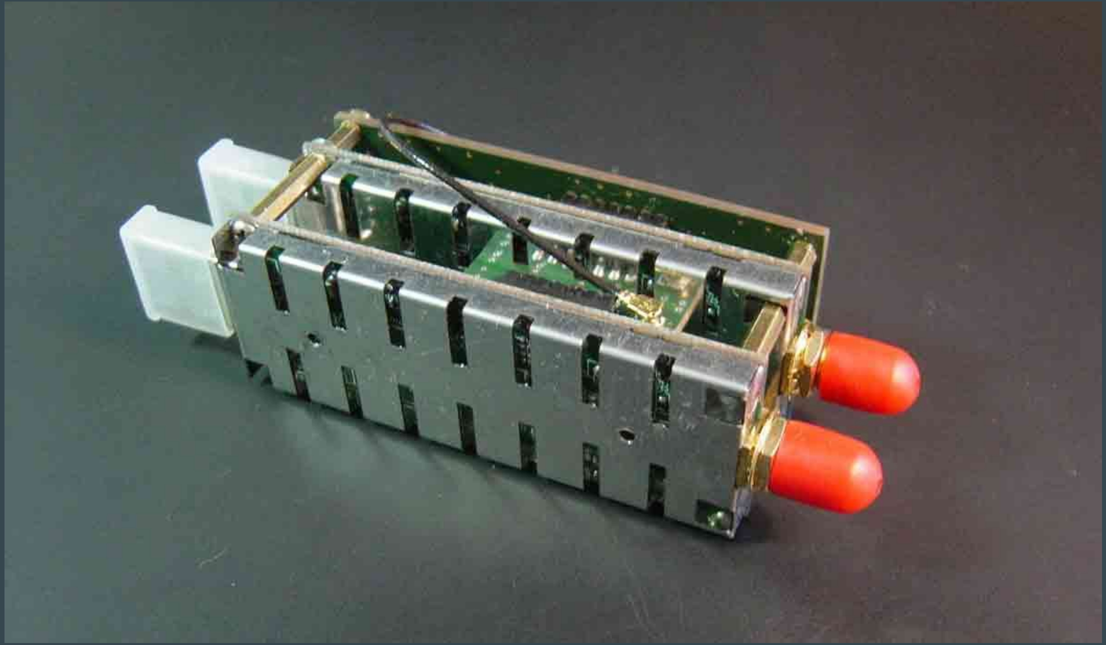


Our Hardware

- 2 RTL-SDR

http://coherent-receiver.com

- Moto X Pure



Previous Work - Methods

- MUSIC IEEE Trans. Antennas Propagation, Vol. AP-34 (March 1986), pp.276-280.
- MN (Min Norm) http://ieeexplore.ieee.org/document/4350746/
- MVDR (Capon) J. Capon “High-Resolution Frequency-Wavenumber Spectrum Analysis”
- Cross Correlation
- TDOA (time difference of arrival)

Cross Correlation

Didn’t give us the DOA results we wanted, but played an important role in synchronizing our receivers.

$$r_p(T) = \sum_{t=-\infty}^{\infty} x_1(t)x_2(t+T)$$

$$\tau = argmax(r_p(T))$$

x1, x2: signals from receivers

r : cross correlation

Tau: the sample difference

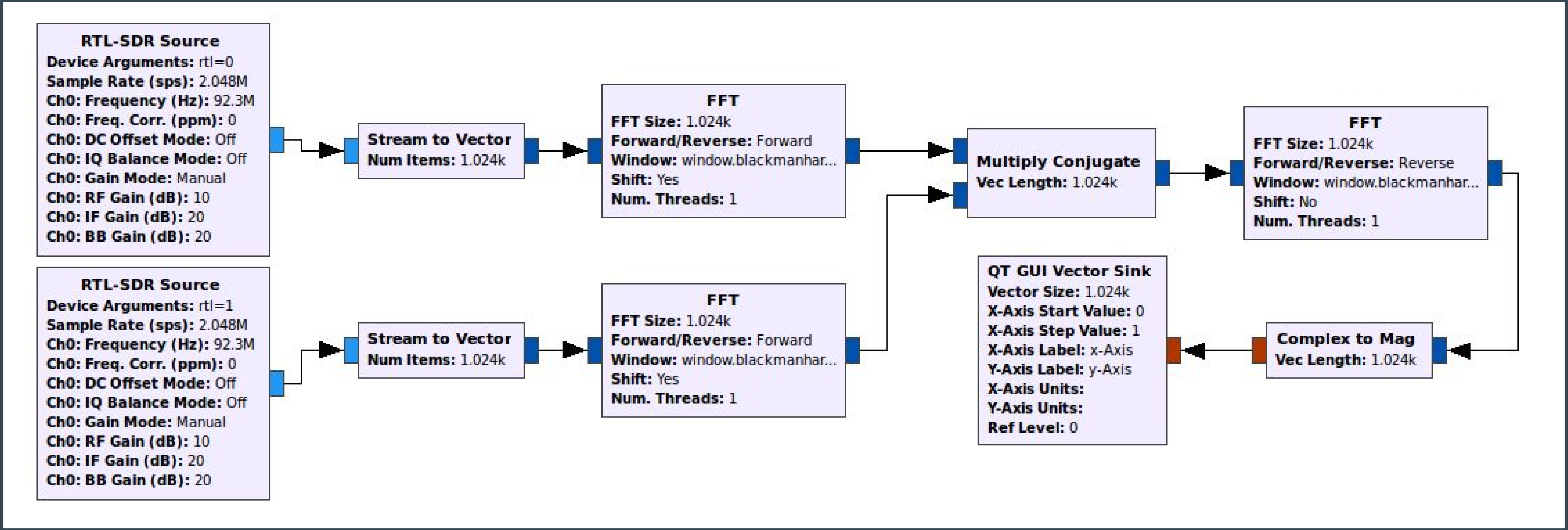
Cross Correlation

We can reduce the needed number of computations significantly by completing this problem in the frequency domain.

$$x_1(t) \Leftrightarrow X_1(f)$$
$$x_2(t) \Leftrightarrow X_2(f)$$
$$r_p(t) \Leftrightarrow R_p(f)$$

$$R_p(f) = X_1(f) \times X_2(f)^*$$

Cross Correlation



RTL-SDR Source
Device Arguments: rtl=0
Sample Rate (sps): 2.048M
Ch0: Frequency (Hz): 92.3M
Ch0: Freq. Corr. (ppm): 0
Ch0: DC Offset Mode: Off
Ch0: IQ Balance Mode: Off
Ch0: Gain Mode: Manual
Ch0: RF Gain (dB): 10
Ch0: IF Gain (dB): 20
Ch0: BB Gain (dB): 20

RTL-SDR Source
Device Arguments: rtl=1
Sample Rate (sps): 2.048M
Ch0: Frequency (Hz): 92.3M
Ch0: Freq. Corr. (ppm): 0
Ch0: DC Offset Mode: Off
Ch0: IQ Balance Mode: Off
Ch0: Gain Mode: Manual
Ch0: RF Gain (dB): 10
Ch0: IF Gain (dB): 20
Ch0: BB Gain (dB): 20

Stream to Vector
Num Items: 1.024k

Stream to Vector
Num Items: 1.024k

FFT
FFT Size: 1.024k
Forward/Reverse: Forward
Window: window.blackmanhar...
Shift: Yes
Num. Threads: 1

FFT
FFT Size: 1.024k
Forward/Reverse: Forward
Window: window.blackmanhar...
Shift: Yes
Num. Threads: 1

Multiply Conjugate
Vec Length: 1.024k

FFT
FFT Size: 1.024k
Forward/Reverse: Reverse
Window: window.blackmanhar...
Shift: No
Num. Threads: 1

Complex to Mag
Vec Length: 1.024k

QT GUI Vector Sink
Vector Size: 1.024k
X-Axis Start Value: 0
X-Axis Step Value: 1
X-Axis Label: x-Axis
Y-Axis Label: y-Axis
X-Axis Units:
Y-Axis Units:
Ref Level: 0

Cross Correlation

For electromagnetic waves traveling at the speed of light and a sample rate of 2 MHz, this solution is able to resolve differences of...

$$Resolution = c \times SamplePeriod$$

$$= \frac{c}{SampleRate}$$

$$= \frac{3e8}{2e6} = 150 \text{ meters}$$

Sub sample resolution can be achieved through interpolation

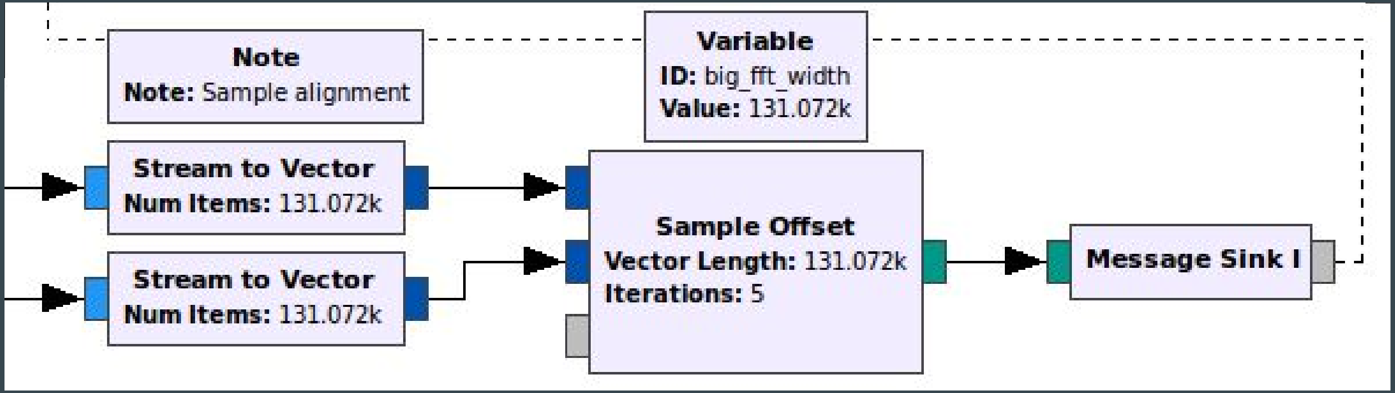
Why bring up Cross Correlation?

The RTL-SDR receivers don’t start at the same time (a sample offset).

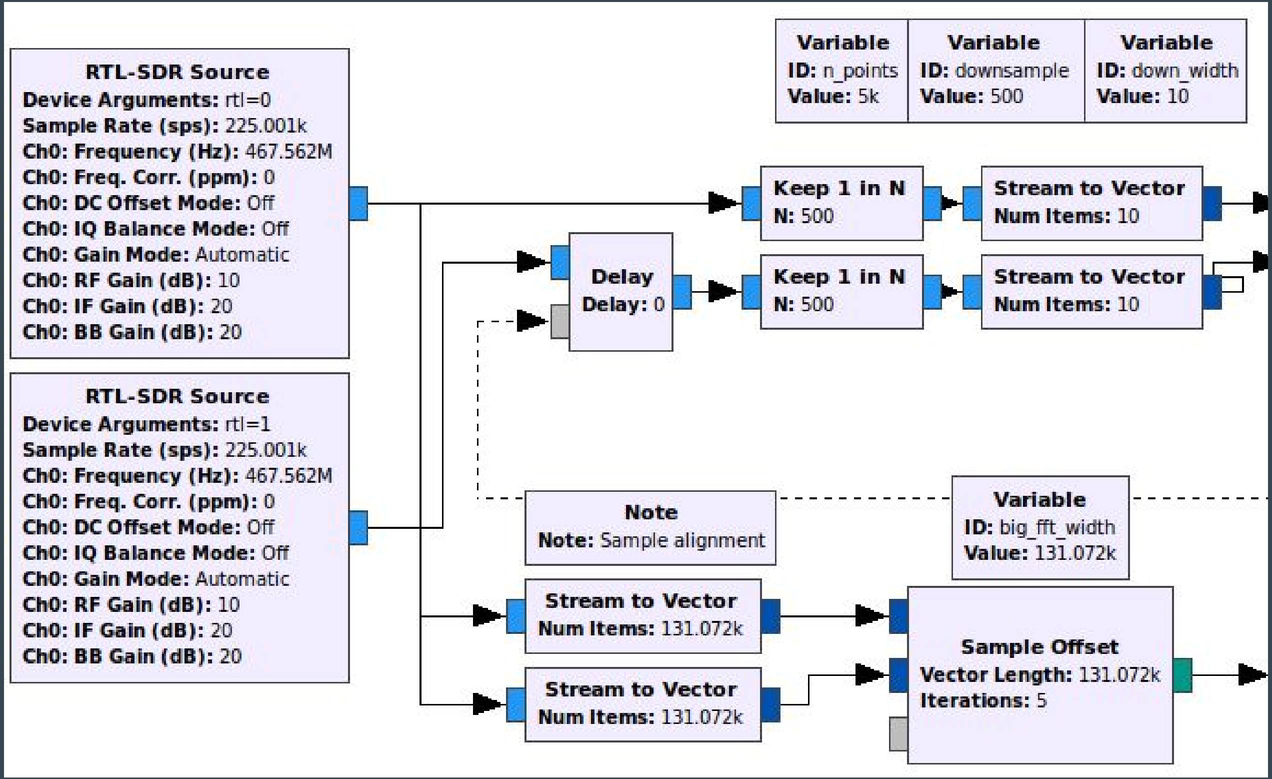
We can correct this by delaying one of the signals until they are roughly aligned in time. The delay we need to apply is the value we get from cross correlation.

Sample Offset Block

- Median sample offset from several cross-correlations
- Stops after *n* iterations
- Can be reset via message



Sample Offset Block



Variable
ID: n_points
Value: 5k

Variable
ID: downsample
Value: 500

Variable
ID: down_width
Value: 10

RTL-SDR Source
Device Arguments: rtl=0
Sample Rate (sps): 225.001k
Ch0: Frequency (Hz): 467.562M
Ch0: Freq. Corr. (ppm): 0
Ch0: DC Offset Mode: Off
Ch0: IQ Balance Mode: Off
Ch0: Gain Mode: Automatic
Ch0: RF Gain (dB): 10
Ch0: IF Gain (dB): 20
Ch0: BB Gain (dB): 20

RTL-SDR Source
Device Arguments: rtl=1
Sample Rate (sps): 225.001k
Ch0: Frequency (Hz): 467.562M
Ch0: Freq. Corr. (ppm): 0
Ch0: DC Offset Mode: Off
Ch0: IQ Balance Mode: Off
Ch0: Gain Mode: Automatic
Ch0: RF Gain (dB): 10
Ch0: IF Gain (dB): 20
Ch0: BB Gain (dB): 20

Keep 1 in N
N: 500

Stream to Vector
Num Items: 10

Delay
Delay: 0

Keep 1 in N
N: 500

Stream to Vector
Num Items: 10

Note
Note: Sample alignment

Variable
ID: big_fft_width
Value: 131.072k

Stream to Vector
Num Items: 131.072k

Stream to Vector
Num Items: 131.072k

Sample Offset
Vector Length: 131.072k
Iterations: 5

Eigenvector method

- Phase difference method
- Still works after downsampling to less than 1 kS/sec
- Steer our phased array with a vector until we find the most power

$$\max\ w^H R w$$

$$\text{s.t. } w^H w = 1$$

Eigenvector method

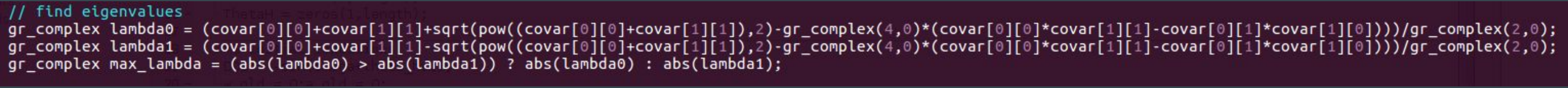
1. Estimate covariance matrix

2. Find maximum eigenvalue

3. Return the argument of the eigenvector

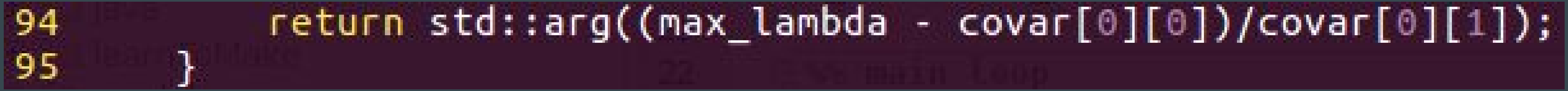
Eigenvector method (Not Complicated)

Analytical solution for the eigenvalue problem.



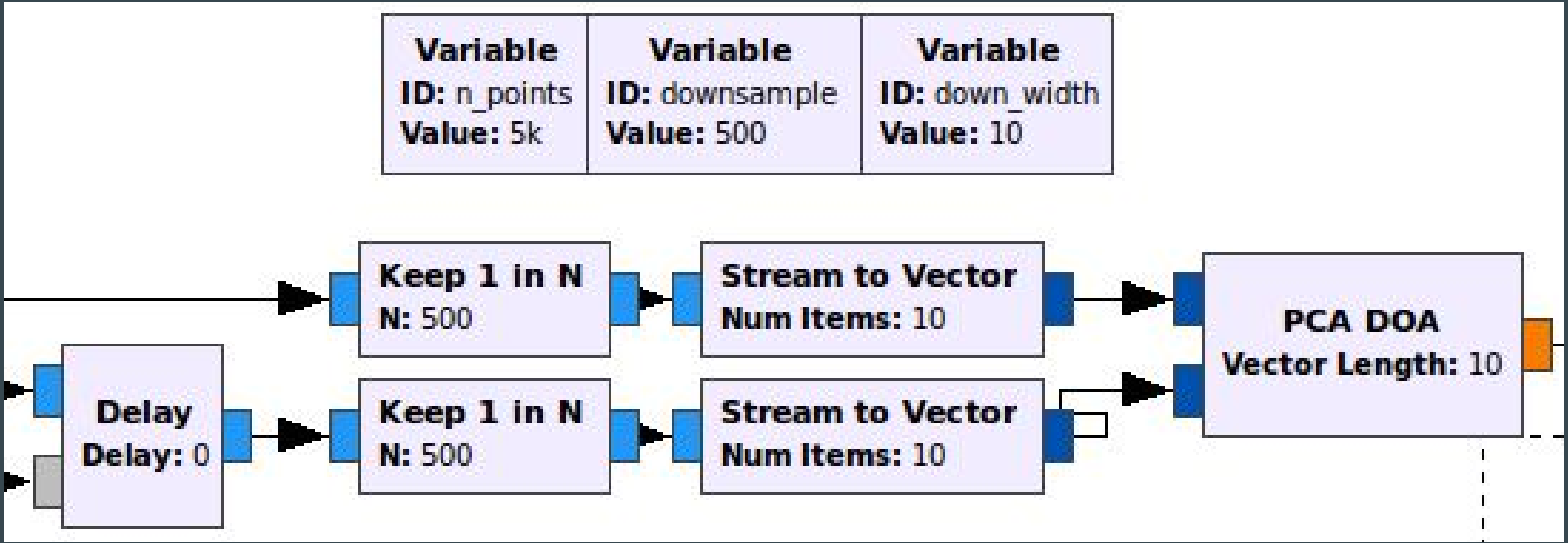
```
// find eigenvalues
gr_complex lambda0 = (covar[0][0]+covar[1][1]+sqrt(pow((covar[0][0]+covar[1][1]),2)-gr_complex(4,0)*(covar[0][0]*covar[1][1]-covar[0][1]*covar[1][0])))/gr_complex(2,0);
gr_complex lambda1 = (covar[0][0]+covar[1][1]-sqrt(pow((covar[0][0]+covar[1][1]),2)-gr_complex(4,0)*(covar[0][0]*covar[1][1]-covar[0][1]*covar[1][0])))/gr_complex(2,0);
gr_complex max_lambda = (abs(lambda0) > abs(lambda1)) ? abs(lambda0) : abs(lambda1);
```

Return the argument of the eigenvector

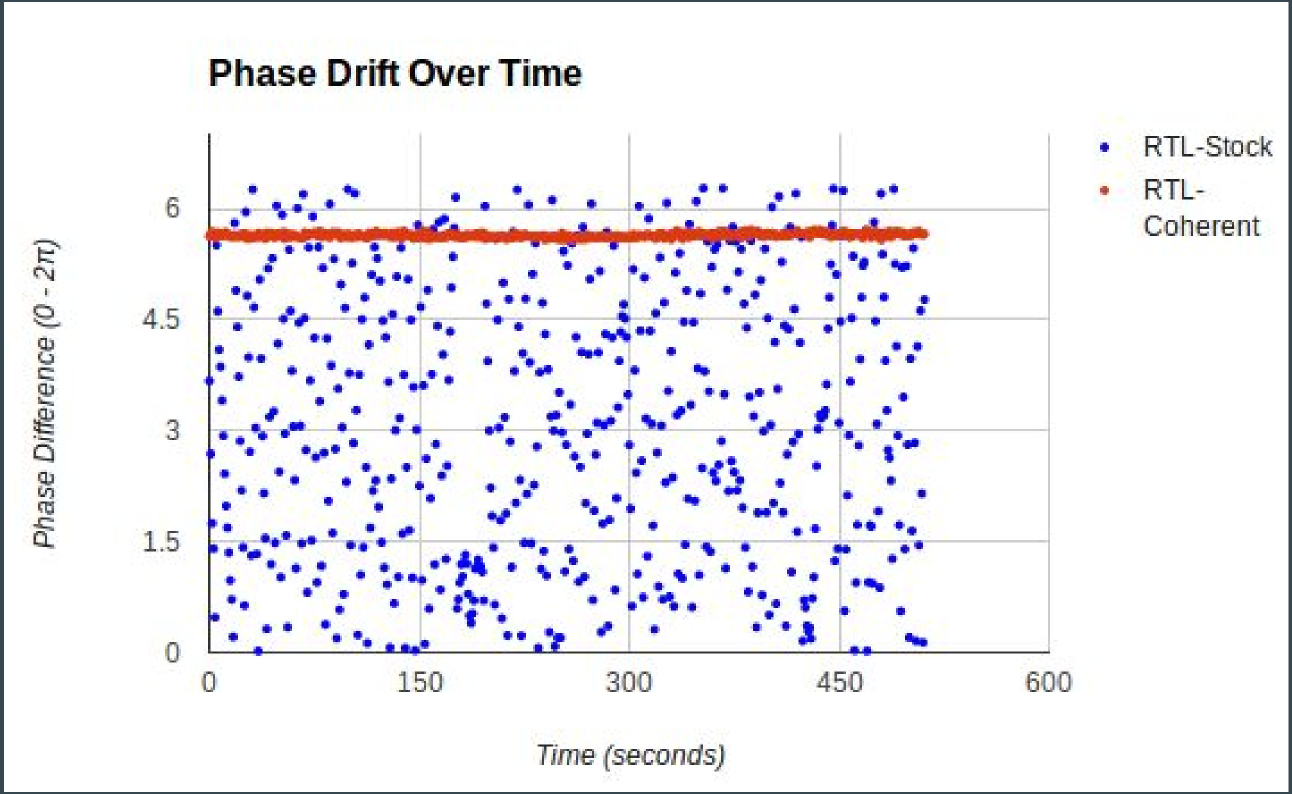


```
        return std::arg((max_lambda - covar[0][0])/covar[0][1]);
    }
```

Eigenvector method

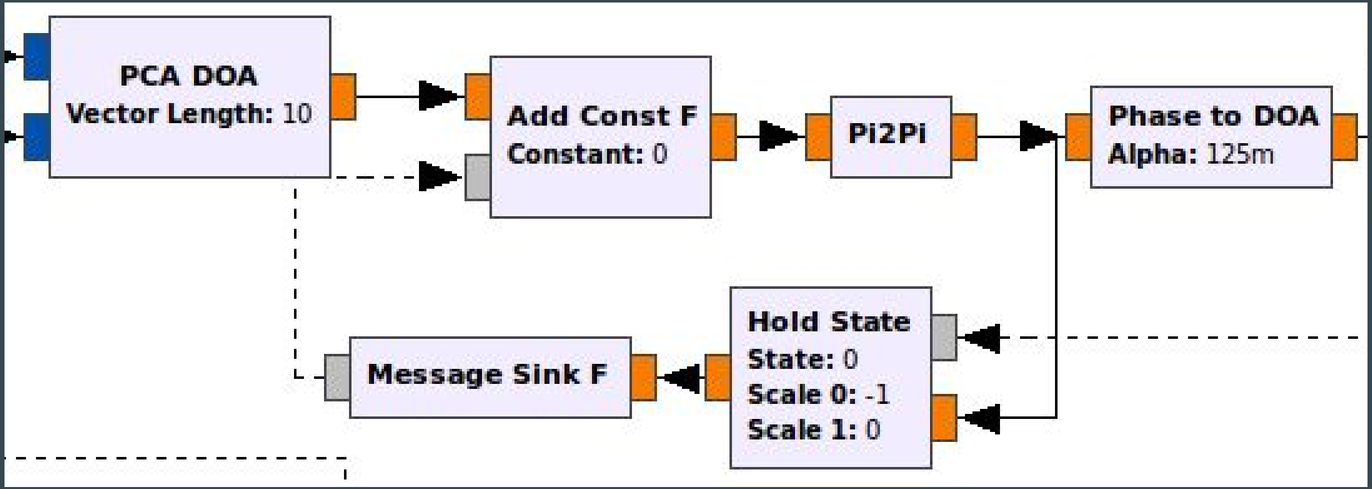


Phase Coherency



Phase Coherency

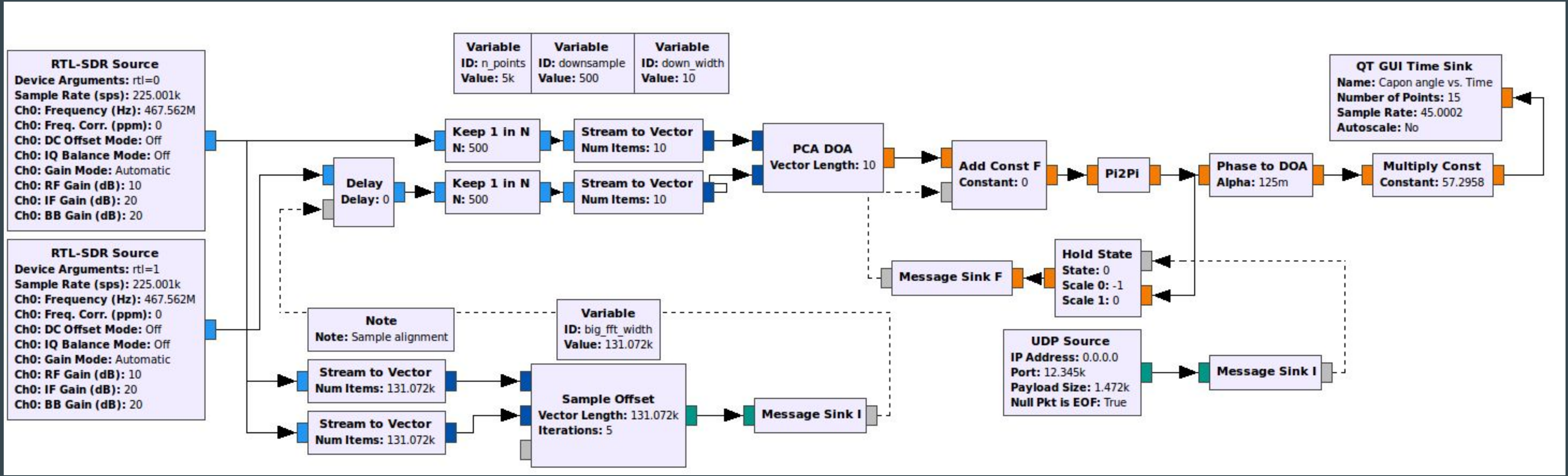
- A constant phase offset can be corrected with addition



Which blocks are custom?

- *Sample Offset* - cross correlation once (or n times)
- *PCA DOA* - returns phase difference
- *Pi2Pi* - wraps a float into the range from -pi to pi
- *Phase to DOA* - map from phase to DOA with arc cosine
- *Hold State* - holds/updates a calibration value
- *Message sinks* - turns a value into a message
- *Delay* - added message control
- *Add Const F* - message control for adding a float constant

Complete Flowgraph



Variable
ID: n_points
Value: 5k

Variable
ID: downsample
Value: 500

Variable
ID: down_width
Value: 10

RTL-SDR Source
Device Arguments: rtl=0
Sample Rate (sps): 225.001k
Ch0: Frequency (Hz): 467.562M
Ch0: Freq. Corr. (ppm): 0
Ch0: DC Offset Mode: Off
Ch0: IQ Balance Mode: Off
Ch0: Gain Mode: Automatic
Ch0: RF Gain (dB): 10
Ch0: IF Gain (dB): 20
Ch0: BB Gain (dB): 20

RTL-SDR Source
Device Arguments: rtl=1
Sample Rate (sps): 225.001k
Ch0: Frequency (Hz): 467.562M
Ch0: Freq. Corr. (ppm): 0
Ch0: DC Offset Mode: Off
Ch0: IQ Balance Mode: Off
Ch0: Gain Mode: Automatic
Ch0: RF Gain (dB): 10
Ch0: IF Gain (dB): 20
Ch0: BB Gain (dB): 20

Delay
Delay: 0

Keep 1 in N
N: 500

Keep 1 in N
N: 500

Stream to Vector
Num Items: 10

Stream to Vector
Num Items: 10

PCA DOA
Vector Length: 10

Add Const F
Constant: 0

Pi2Pi

Phase to DOA
Alpha: 125m

Multiply Const
Constant: 57.2958

QT GUI Time Sink
Name: Capon angle vs. Time
Number of Points: 15
Sample Rate: 45.0002
Autoscale: No

Hold State
State: 0
Scale 0: -1
Scale 1: 0

Message Sink F

UDP Source
IP Address: 0.0.0.0
Port: 12.345k
Payload Size: 1.472k
Null Pkt is EOF: True

Message Sink I

Note
Note: Sample alignment

Variable
ID: big_fft_width
Value: 131.072k

Stream to Vector
Num Items: 131.072k

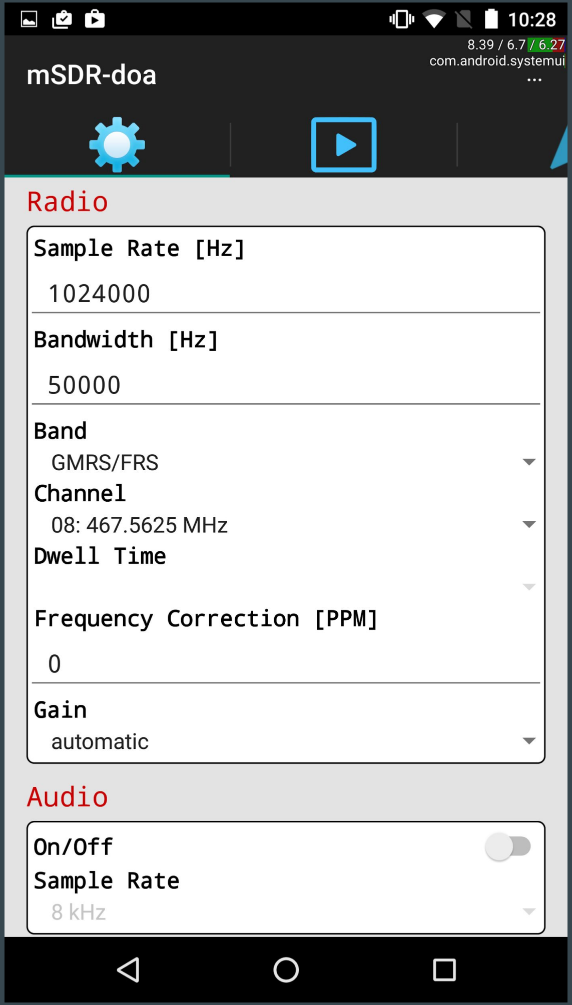
Stream to Vector
Num Items: 131.072k

Sample Offset
Vector Length: 131.072k
Iterations: 5

Message Sink I

Making it Mobile

- Gather parameters in Java UI


10:28
8.39 / 6.7 / 6.27
com.android.systemui

mSDR-doa

Radio

Sample Rate [Hz]
1024000

Bandwidth [Hz]
50000

Band
GMRS/FRS

Channel
08: 467.5625 MHz

Dwell Time

Frequency Correction [PPM]
0

Gain
automatic

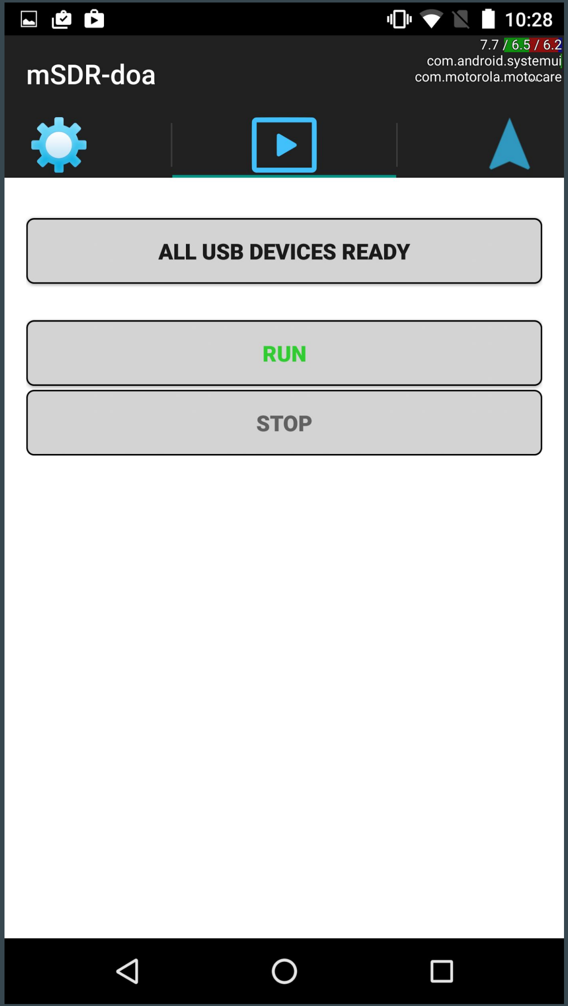
Audio

On/Off

Sample Rate
8 kHz

Making it Mobile

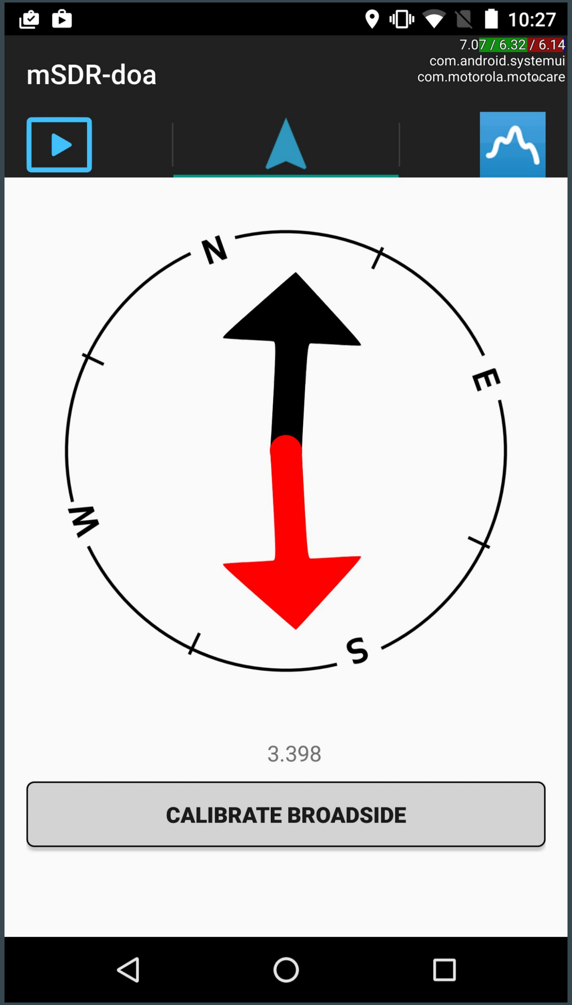
- Cross-compile C++ flowgraph to statically linked executable
- Call it using Android Java Native Interface wrapper



Making it Mobile

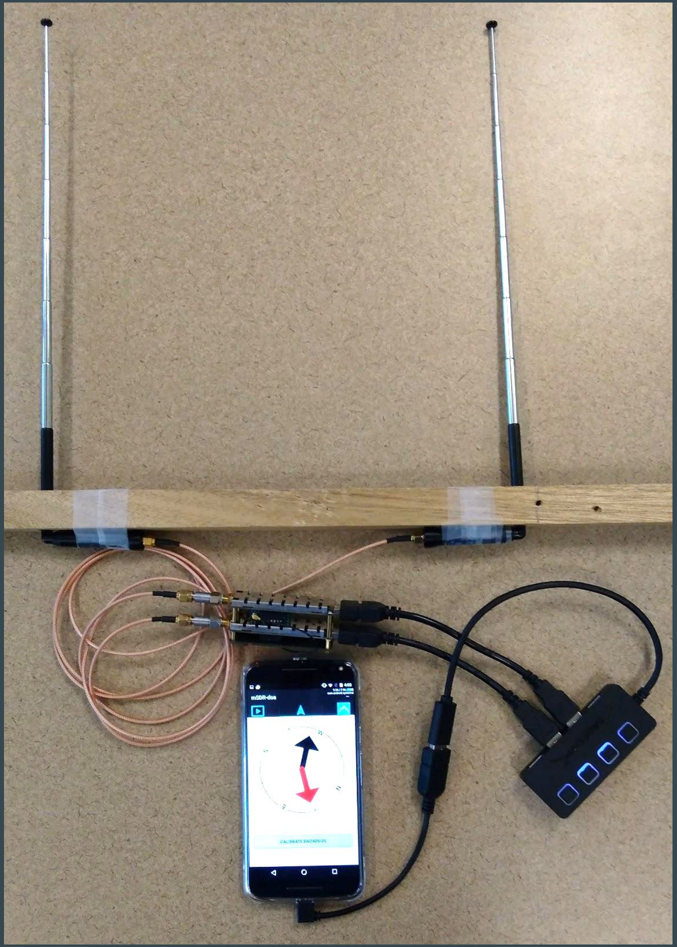
- UDP packets for data, control between Java and Gnuradio
- Now we can add maps, headings, arrows etc.

Red arrow represents the direction ambiguity due to using a phase difference method



The Prototype

- Finest-quality plywood and tape
- Half-wavelength spacing (FRS band)
- USB hub for RTL-SDR dongles



Testing

- Walkie Talkies (FRS Band)
- FM Radio Stations

http://a.co/goIIb5u

Videos

- https://www.youtube.com/watch?v=aUubSNUxX8g

Videos

- https://www.youtube.com/watch?v=RKDnkXq2mVM&t=8s

Code

- https://github.com/samwhiting/gnuradio-doa

Thanks

This research was funded by the Laboratory for Telecommunication Sciences in Maryland.

Template app by Dave Mahoney

Auxiliary Slides

Principal Component Analysis

Our incoming signal *s(t)* is received by two different antennas as *x1(t)* and *x2(t)*. We'll call *x1(t)* our reference, and let *x2(t)* have a delay represented as Tau.

$$a(t) = \text{ baseband signal}$$
$$s(t) = \text{ modulated signal}$$
$$s(t) = a(t)cos(\omega t)$$

$$x_1(t) = s(t) = a(t)cos(\omega t)$$
$$x_2(t) = s(t-\tau) = a(t-\tau)cos(\omega(t-\tau))$$

Principal Component Analysis

Use the narrow-band assumption to say that the primary difference between the two signals can be represented by a phase shift in the carrier frequency.

$$x_1(t) = s(t) = a(t)cos(\omega t)$$
$$x_2(t) = s(t-\tau) = a(t-\tau)cos(\omega(t-\tau))$$
$$x_2(t) = a(t-\tau)cos(\omega t - \phi)$$
$$\text{Where } \phi = \omega\tau$$

$$\tilde{x}_1(t) = a(t)$$
$$\tilde{x}_2(t) = a(t)e^{-j\phi}$$

Principal Component Analysis

Stack our signals into a vector

$$\bar{x}(t) = \begin{bmatrix} \tilde{x}_1(t) \\ \tilde{x}_2(t) \end{bmatrix}$$

$$\bar{x}(t) = a(t) \begin{bmatrix} 1 \\ e^{-j\phi} \end{bmatrix}$$

Principal Component Analysis

Set up a beam steering vector, and apply it to our signal.

$$y(t,\hat{\phi}) = \begin{bmatrix} 1 \\ e^{-j\hat{\phi}} \end{bmatrix}^H \bar{x}(t)$$

$$y(t,\hat{\phi}) = \begin{bmatrix} 1 \\ e^{-j\hat{\phi}} \end{bmatrix}^H a(t) \begin{bmatrix} 1 \\ e^{-j\phi} \end{bmatrix}$$

$$y(t,\hat{\phi}) = a(t)(1 + e^{j(\hat{\phi}-\phi)})$$

$$y(t,\hat{\phi}) \text{ has a maximum when } \hat{\phi} = \phi$$

Principal Component Analysis

Take the variance of y in order to look at power of the filter output.

$$\text{let } w = \begin{bmatrix} 1 \\ e^{-j\hat{\phi}} \end{bmatrix}$$

with a zero-mean signal, $\sigma_y^2 = E|y|^2$

$$E|y|^2 = E|w^H \bar{x}(t)|^2 = E[(w^H \bar{x})(w^H \bar{x})^*]$$

$$= E[w^H \bar{x}\bar{x}^H w] = w^H E[\bar{x}\bar{x}^H]w = w^H R_{xx} w$$

Principal Component Analysis

Maximize the power by changing the steering vector with a constraint to limit the gain.

$$\max\ w^H R w$$

$$\text{s.t. } w^H w = 1$$

Principal Component Analysis

Setup the constrained optimization problem with Lagrange multipliers.

$$\nabla f = \lambda \nabla g$$

$$\frac{\mathrm{d}}{\mathrm{d}w} w^H R w = \lambda \frac{\mathrm{d}}{\mathrm{d}w} (w^H w - 1)$$

$$Rw = \lambda w$$

Principal Component Analysis

We are left with an eigenvalue problem.

$$Rw = \lambda w$$
$$w(R - \lambda I) = 0$$

Eigenvalue/vector pairs

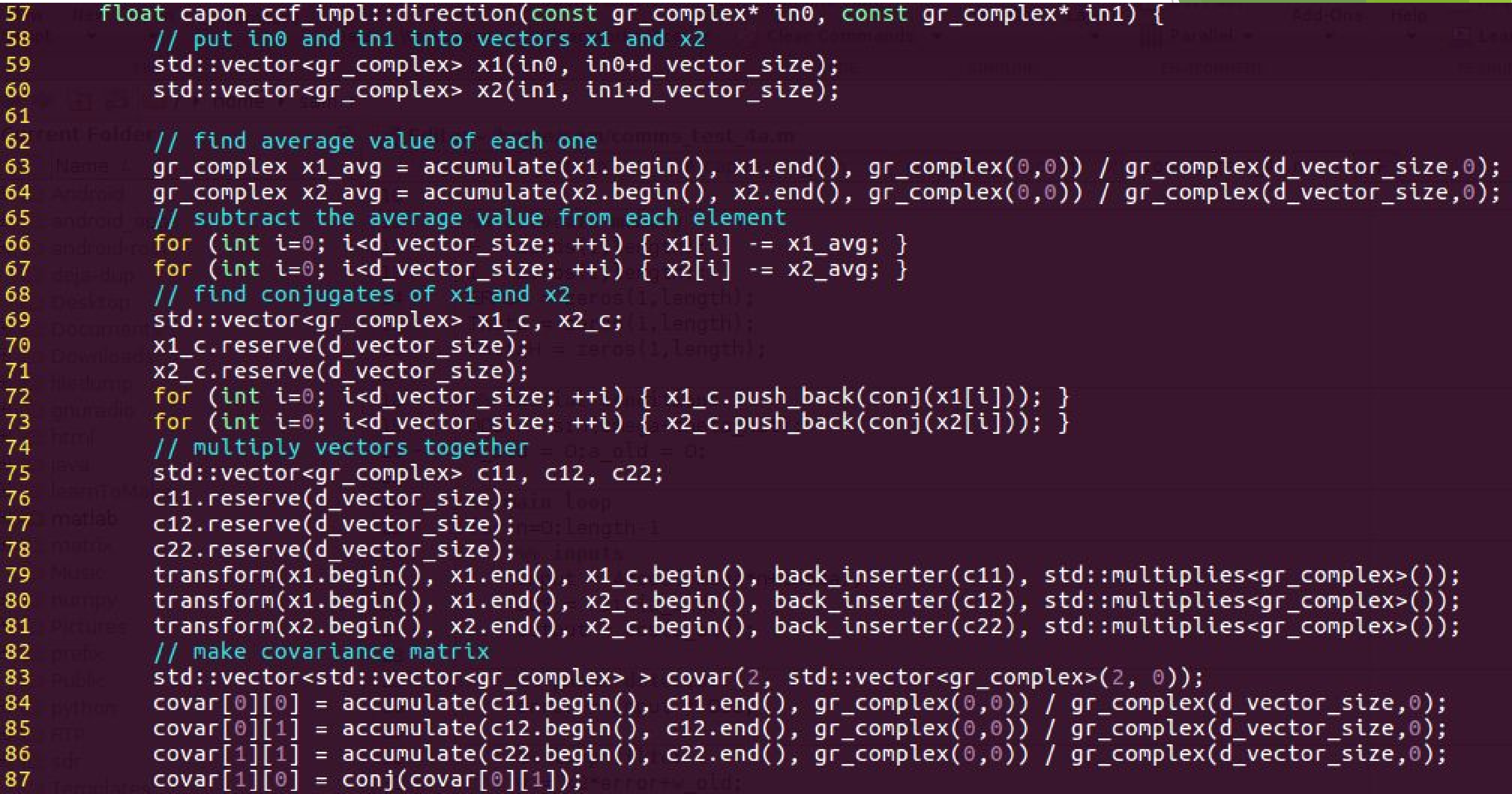
$$\lambda_1 \leftrightarrow E_1$$
$$\lambda_2 \leftrightarrow E_2$$

Principal Component Analysis

The maximum eigenvalue has an eigenvector that provides us with our optimal steering vector. This can be normalized, and the phase angle extracted.

$$\lambda_1 \leftrightarrow E_1$$
$$\lambda_2 \leftrightarrow E_2$$
$$\max\left[\lambda_1, \lambda_2\right] = \lambda \leftrightarrow E$$

$$E = \begin{bmatrix} u \\ v \end{bmatrix} = c \begin{bmatrix} 1 \\ e^{j\hat{\phi}} \end{bmatrix}$$

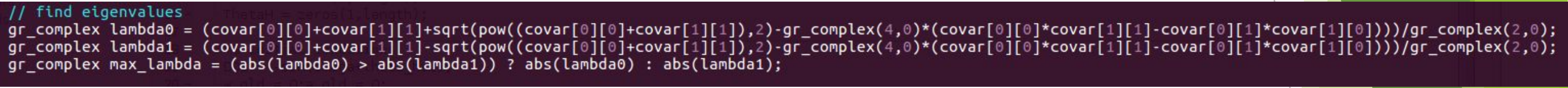


```
float capon_ccf_impl::direction(const gr_complex* in0, const gr_complex* in1) {
    // put in0 and in1 into vectors x1 and x2
    std::vector<gr_complex> x1(in0, in0+d_vector_size);
    std::vector<gr_complex> x2(in1, in1+d_vector_size);

    // find average value of each one
    gr_complex x1_avg = accumulate(x1.begin(), x1.end(), gr_complex(0,0)) / gr_complex(d_vector_size,0);
    gr_complex x2_avg = accumulate(x2.begin(), x2.end(), gr_complex(0,0)) / gr_complex(d_vector_size,0);
    // subtract the average value from each element
    for (int i=0; i<d_vector_size; ++i) { x1[i] -= x1_avg; }
    for (int i=0; i<d_vector_size; ++i) { x2[i] -= x2_avg; }
    // find conjugates of x1 and x2
    std::vector<gr_complex> x1_c, x2_c;
    x1_c.reserve(d_vector_size);
    x2_c.reserve(d_vector_size);
    for (int i=0; i<d_vector_size; ++i) { x1_c.push_back(conj(x1[i])); }
    for (int i=0; i<d_vector_size; ++i) { x2_c.push_back(conj(x2[i])); }
    // multiply vectors together
    std::vector<gr_complex> c11, c12, c22;
    c11.reserve(d_vector_size);
    c12.reserve(d_vector_size);
    c22.reserve(d_vector_size);
    transform(x1.begin(), x1.end(), x1_c.begin(), back_inserter(c11), std::multiplies<gr_complex>());
    transform(x1.begin(), x1.end(), x2_c.begin(), back_inserter(c12), std::multiplies<gr_complex>());
    transform(x2.begin(), x2.end(), x2_c.begin(), back_inserter(c22), std::multiplies<gr_complex>());
    // make covariance matrix
    std::vector<std::vector<gr_complex> > covar(2, std::vector<gr_complex>(2, 0));
    covar[0][0] = accumulate(c11.begin(), c11.end(), gr_complex(0,0)) / gr_complex(d_vector_size,0);
    covar[0][1] = accumulate(c12.begin(), c12.end(), gr_complex(0,0)) / gr_complex(d_vector_size,0);
    covar[1][1] = accumulate(c22.begin(), c22.end(), gr_complex(0,0)) / gr_complex(d_vector_size,0);
    covar[1][0] = conj(covar[0][1]);
```

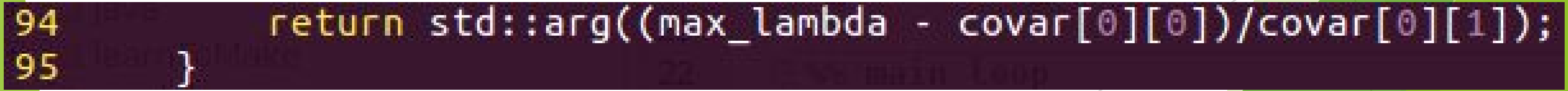
Capon Method

Analytical solution for the eigenvalue problem.



```
// find eigenvalues
gr_complex lambda0 = (covar[0][0]+covar[1][1]+sqrt(pow((covar[0][0]+covar[1][1]),2)-gr_complex(4,0)*(covar[0][0]*covar[1][1]-covar[0][1]*covar[1][0])))/gr_complex(2,0);
gr_complex lambda1 = (covar[0][0]+covar[1][1]-sqrt(pow((covar[0][0]+covar[1][1]),2)-gr_complex(4,0)*(covar[0][0]*covar[1][1]-covar[0][1]*covar[1][0])))/gr_complex(2,0);
gr_complex max_lambda = (abs(lambda0) > abs(lambda1)) ? abs(lambda0) : abs(lambda1);
```

Return the argument of the eigenvector



```
        return std::arg((max_lambda - covar[0][0])/covar[0][1]);
    }
```