Anurag Pallaprolu (9847112)

ECE 278C : Imaging Systems
Winter 2020

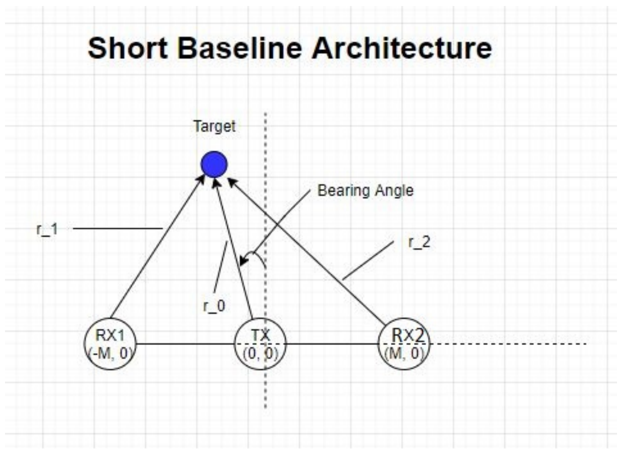
Term Project

Short Baseline Systems

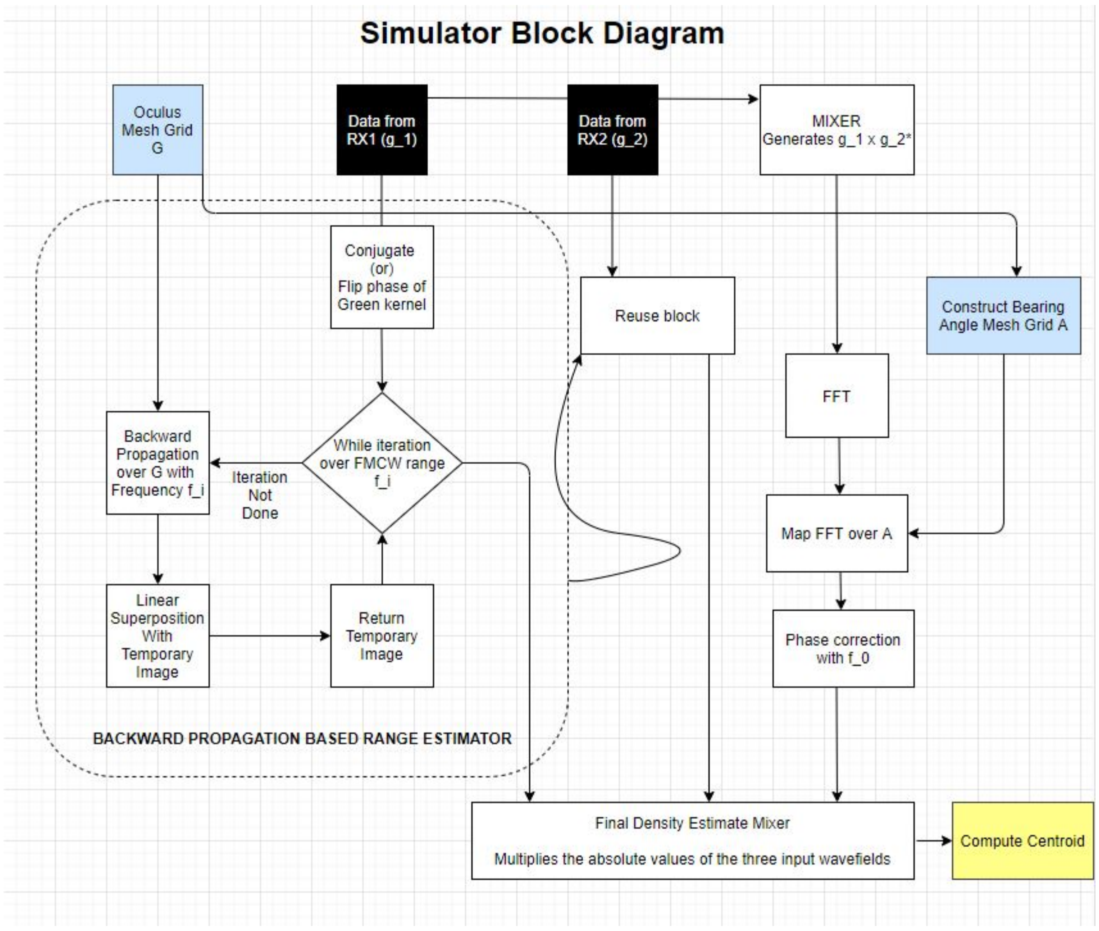
Submitted 21 March 2020

Algorithm

We show below the physical architecture of the system we are using for the simulator.



With this in mind, the following block diagram effectively describes the algorithm for image reconstruction:


Simulator Block Diagram
Oculus Mesh Grid G
Data from RX1 (g_1)
Data from RX2 (g_2)
MIXER Generates g_1 x g_2*
Conjugate (or) Flip phase of Green kernel
Reuse block
Construct Bearing Angle Mesh Grid A
FFT
Backward Propagation over G with Frequency f_i
Iteration Not Done
While iteration over FMCW range f_i
Map FFT over A
Linear Superposition With Temporary Image
Return Temporary Image
Phase correction with f_0
BACKWARD PROPAGATION BASED RANGE ESTIMATOR
Final Density Estimate Mixer
Multiplies the absolute values of the three input wavefields
Compute Centroid

We will now describe the implementation details with respect to the algorithm. The description of the results, discussion on efficiency of computation and image reconstruction accuracy have been moved to the summary. We are provided with two data-streams of electromagnetic field data for the receivers RX_1 and RX_2, named $g_1[n]$ and $g_2[n]$, and these receivers are located at $(-M\lambda_0, 0)$ and $(M\lambda_0, 0)$ respectively. The transmitter TX is placed at the midpoint, in this case the origin of the coordinate system. The mode of transmission is a step frequency modulated chirp signal whose wavelength variation is given by the formula:

$$\lambda_n = \frac{128\lambda_0}{n + 128}$$

The receivers capture the TX radiation that is scattered off the target, whose location r_0 we will have to estimate ($\hat{r}_0$). We can use the $RX_1 - TX$ system as an antenna array of aperture $M\lambda_0$ to estimate the range given by $r_0 + r_1$ as shown in the first figure on the previous page. Consequently, we can also use the $TX - RX_2$ system independently as another antenna array of the same aperture size to estimate the range that is denoted by $r_0 + r_2$. To wit, both of these ranges are nothing but the total distance travelled from TX to the target, and then from the target to the corresponding RX.

Thus, to simulate the presence of a *point target* at a location r_0, we must simply generate the following two wavefield sequences for each FMCW frequency:

$$g_1[n] = exp(j2\pi(r_0 + r_1)/\lambda_n)$$
$$g_2[n] = exp(j2\pi(r_0 + r_2)/\lambda_n)$$

Obviously, we have idealized the situation a lot, but the phase-only Green kernel works as well as the full kernel for the case of demonstration. Once we have these two sequences, we can straightaway use the ever reliable backward propagation algorithm, which is detailed below:

Initialize
The final image data structure $\mathcal{F}$ is initialized.
For each λ_n

$$\mathcal{F} = \mathcal{F} + g[n] \times exp(-j2\pi\frac{|r_0 + r_i|}{\lambda_n})$$

where $|r_0 + r_i|$ is constructed over the imaging mesh grid $\mathcal{G}$.

Thus the final image $\mathcal{F}_i$ is obtained from the $RX_i - TX$ pair. We will return to using these after we are done with the bearing angle estimation and we will use the FFT matrix technique instead to get the angle estimate. While the derivation is exactly the same as that of the FFT based range estimation construction, there is a difference in the way the projection is made onto the FFT matrix. To wit, we have the following formula for the FMCW range density estimation:

$$\mathcal{P}(r) = \sum_n g^*[n]exp(-\frac{j(\omega_0 + n\Delta\omega)r}{c})$$

The guiding principle behind bearing angle estimation is that, if the target is sufficiently far field (Fraunhofer zone) we have the incoming radiation scattering in parallel beams, with the only range offset given by $\Delta r = D\sin\theta$ where θ is the angle made by the beam with the azimuth w.r.t the lagging receiver. Therefore, if we can get the series that represents the ranging of $r_1 - r_2$ instead, we can then plug $r = \Delta r = r_1 - r_2$ into the density summation above and we would be now able to estimate the bearing angle for far field targets. This series can be constructed using $g_1[n]g_2^*[n]$ and we will call this series $q[n]$. We now have:

$$\mathcal{P}(\Delta r) = \sum_n q^*[n]exp(-\frac{j(\omega_0 + n\Delta\omega)\Delta r}{c})$$

$$\implies \mathcal{P}(\Delta r) = exp(-\frac{j\omega_0\Delta r}{c})\sum_n q^*[n]exp(-\frac{jn\Delta\omega\Delta r}{c})$$

$$\implies \mathcal{P}(D\sin\theta) = exp(-\frac{j\omega_0 D\sin\theta}{c})\sum_n q^*[n]exp(-\frac{jn\Delta\omega D\sin\theta}{c})$$

Before we proceed with any further derivation, we need to identify some of the constants in the expression. We are given that the wavelength evolution of the FMCW beamforming is given by:

$$\lambda_n = \frac{128\lambda_0}{n + 128}$$

$$\implies \frac{1}{\lambda_n} = \frac{n + 128}{128\lambda_0}$$

To simplify calculations, we can assume $\lambda_0 = 1$, we then have:

$$\implies f_n = \frac{c}{\lambda_n} = \frac{c}{128}n + c$$

$$\implies \omega_n = 2\pi f_n = \frac{2\pi c}{128}n + 2\pi c = \Delta\omega n + \omega_0$$

Therefore, we now have $\omega_0 = 2\pi c$ and $\Delta\omega = \frac{2\pi c}{128}$. Plugging these into the density equation, we have:

$$\mathcal{P}(D\sin\theta) = exp(-j2\pi D\sin\theta)\sum_n q^*[n]exp(-\frac{jn2\pi D\sin\theta}{128})$$

We note that this is precisely a 128 bit FFT of q^* and therefore identify the kernels:

$$\frac{j2\pi nk}{N} = \frac{jn2\pi D\sin\theta}{128}$$

$$\implies k = D\sin\theta$$

If we pick the value of the FFT of $q^*[n]$ at the index $[D\sin\theta]$ we will then only have to append the constant phase term outside the summation to complete the density computation. However, the practical aspect of implementing this technique would require us to transform the current Euclidean plane $\mathcal{G}$ with radial measures $\sqrt{x^2+y^2}$ into that of $D\sin\theta$, which we call $\mathcal{H}$. Mathematically, we have

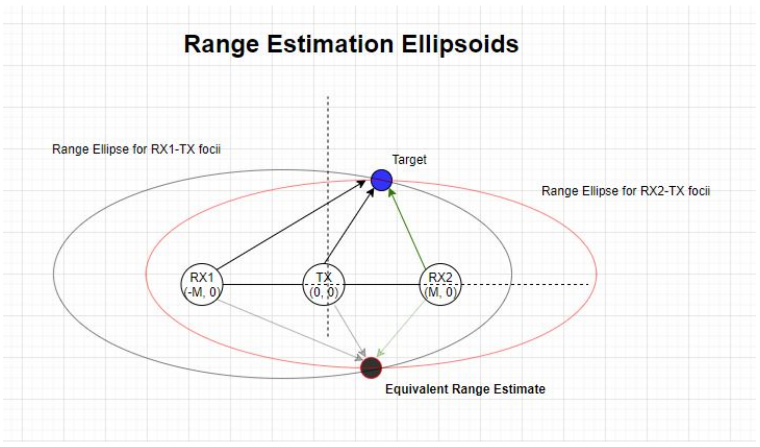
$$\theta = \tan^{-1}(\frac{x}{y})$$

$$\implies \mathcal{G}(x,y) \to \mathcal{G}(D\sin(\tan^{-1}(\frac{x}{y}))) = \mathcal{G}(2M\lambda_0\sin(\tan^{-1}(\frac{x}{y}))) = \mathcal{H}$$

Once we are done with this transformation, we construct the FFT matrix by mapping the buckets over the new grid $\mathcal{H}$, thereby giving us the final image density in this case $\mathcal{F}_3$ as:

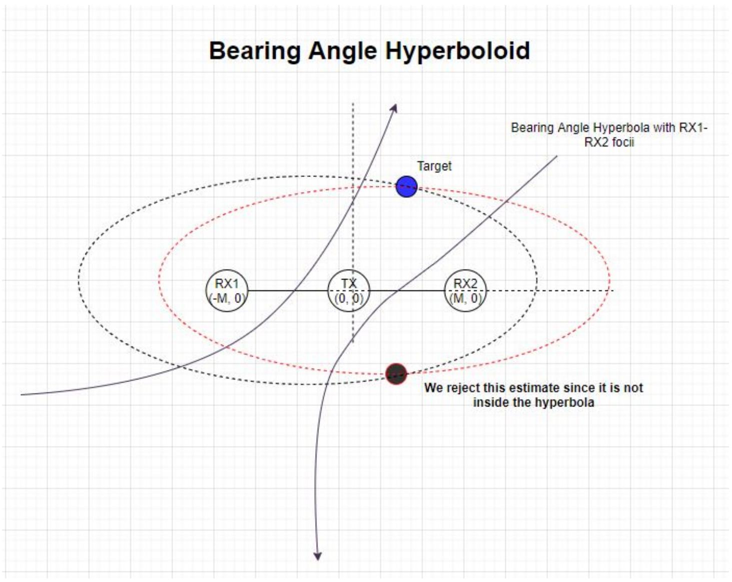
$$\mathcal{F}_3 = exp(-j2\pi\mathcal{H}) \times FFT(\mathcal{H})$$

Once we have computed $\mathcal{F}_1, \mathcal{F}_2, \mathcal{F}_3$, we need to think about combining these three pieces of information to give a consolidated image in the patch chosen as the Euclidean mesh grid $\mathcal{G}$. Fundamentally, these three objects are complex values of the wavefield over $\mathcal{G}$, computed as linear superposition of phase-only Green kernels. Since the phase only kernels obviously have unit magnitude, the (magnitudes of) objects $\mathcal{F}_i$ which represent the convolution sums can be seen as computing the probabilistic expectation of the range estimation at each point.[1] Therefore, for $\mathcal{F}_1$ and $\mathcal{F}_2$ we would expect the parts of $\mathcal{G}$ with high expectation value of the presence of the target as those points where $r_0 + r_1 = r_{target}$ i.e., they would be ellipses(2D)/ellipsoids(3D).


Range Estimation Ellipsoids
Range Ellipse for RX1-TX focii
Target
Range Ellipse for RX2-TX focii
RX1 (-M, 0)
TX (0, 0)
RX2 (M, 0)
Equivalent Range Estimate

[1]To be mathematically complete, we will have to introduce a range-only term which goes as $\frac{1}{\sqrt{r}}$ and the series $\sum\frac{1}{\sqrt{n}}$ will converge due to fixed number of frequencies transmitted, so we are still fine. Also, we take the magnitudes to get a real number for the probability measure.

However in the case of $\mathcal{F}_3$ we will have those parts of $\mathcal{G}$ representing high expectation value as those points where $\theta = \theta_{target}$. Just as we argued earlier, the Fraunhofer limit allows us to connect $\sin\theta$ to $\Delta r = r_1 - r_2$. Therefore, we are looking for points such that $r_1 - r_2 = D \sin\theta_{target}$ which would be a hyperbola(2D)/hyperboloid(3D) with focii at the two receivers.


Bearing Angle Hyperboloid
Bearing Angle Hyperbola with RX1-RX2 focii
Target
RX1 (-M, 0)
TX (0, 0)
RX2 (M, 0)
We reject this estimate since it is not inside the hyperbola

As can be seen above (and which will be further elaborated in the last section), out of the two geometrically possible locations[2] of the target from the range estimation, one of them lies outside the hyperbola and is therefore rejected to get our final location. Thus, if we have the same point with high density values on both the ellipses and the hyperbola we are left with the answer: so the solution must be to multiply these three wavefield values at each point on $\mathcal{G}$, since if a given point had high density values for the two ellipses but not for the hyperboloid the overall product would be much lesser than if it did have all three high densities. Mathematically:

$$\mathcal{F}_{image} = |\mathcal{F}_1| * |\mathcal{F}_2| * |\mathcal{F}_3|$$

where $*$ represents elementwise multiplcation over the mesh grid $\mathcal{G}$.

[2]As will be clarified in the Summary, in general two conics can meet at four points, but since $M \neq 0$, we will have only half of the symmetry i.e., two solutions.

Code

We will start with importing the base packages and some useful routines:

```python
#NumPy = Numerical Python for handling matrices
import numpy as np
#Basic Plotting
import matplotlib.pyplot as plt
#Contour Plotting Support
from mpl_toolkits.mplot3d import Axes3D
#Safe memory re-allocation for large matrices
import copy
# For video display
from IPython.display import clear_output
# For adding time delay between video frames
import time
# For finding hotspots in each reconstructed image.
from photutils.detection import find_peaks

# The Green propagator for the EM Field
def green2D(dx, dy, lam, mag=False):
    if mag:
        return (1/(np.complex(0, 1) * lam * np.sqrt(dx**2 + dy**2))) * np.
    exp(np.complex(0, 1)*2*np.pi*np.sqrt(dx**2 + dy**2)/lam)
    else:
        return np.exp(np.complex(0, 1)*2*np.pi*np.sqrt(dx**2 + dy**2)/lam)

# Routine to display the wavefield
def displayWF(wavefield, colormap, allComp=False, showPhase=False):
    if allComp:
        print("Real Part of Wavefield")
        plt.contourf(wavefield.real, cmap=colormap)
        plt.colorbar()
        plt.axis('scaled')
        plt.show()

        print("Imaginary Part of Wavefield")
        plt.contourf(wavefield.imag, cmap=colormap)
        plt.colorbar()
        plt.axis('scaled')
        plt.show()

    print("Absolute Value of Wavefield")
    plt.contourf(np.abs(wavefield), cmap=colormap)
    plt.colorbar()
    plt.axis('scaled')
    plt.show()

    if  showPhase:
        print("Phase Value of Wavefield")
        plt.contourf(np.angle(wavefield), cmap=colormap)
        plt.colorbar()
        plt.axis('scaled')
        plt.show()
```

We will then write our short baseline reconstruction engine as described in the block diagram of the earlier section as a routine `baselineSystem`. We follow it up with another routine to find the maximum likely centroid from the final image density computed by the same.

```python
def baselineSystem( \
                    g1, g2,\ # Input streams
                    M, \     # Spacing Parameter
                    xx, yy, \ # The mesh grid G
                    lam0, lamArray, \ #Base wavelength, FMCW Array
                    debug=False): # Show/Hide Plots

    rTX = (0, 0)
    rRX1 = (-M*lam0, 0)
    rRX2 = (M*lam0, 0)

    z_image_rx1 = 0.0
    idx = 0

    for lamTmp in lamArray:
        z_image_rx1 = z_image_rx1 + \
                      g1[idx] * \
                      np.exp(-np.complex(0, 1)*2*np.pi* \
                     (np.sqrt((xx - rRX1[0])**2 + (yy - rRX1[1])**2) + \
                      np.sqrt((xx - rTX[0])**2 + (yy - rTX[1])**2))/lamTmp)

        idx = idx + 1

    if debug: displayWF(z_image_rx1, 'gray_r', showPhase=False)

    z_image_rx2 = 0.0
    idx = 0

    for lamTmp in lamArray:
        z_image_rx2 = z_image_rx2 + \
                      g2[idx] *  \
                      np.exp(-np.complex(0, 1)*2*np.pi* \
                     (np.sqrt((xx - rRX2[0])**2 + (yy - rRX2[1])**2) + \
                      np.sqrt((xx - rTX[0])**2 + (yy - rTX[1])**2))/lamTmp)

        idx = idx + 1

    if debug: displayWF(z_image_rx2, 'gray_r', showPhase=False)

    angularBuckets = (2*M*lam0 * np.sin(np.arctan(xx/yy))).astype(int)
    condition = angularBuckets < -128
    # To correct for 32 bit integer underflow!
    np.place(angularBuckets, condition, 0)

    q = g1 * np.conjugate(g2)
    q_fft = np.fft.fft(np.conjugate(q), n=128)

    c = 1
    # Worst case, we can check for dimensions and substitute 3e+8.
    omega_0 = 2*np.pi*c/lam0

    z_image_bearing = 0.0



    FFTMatrix = np.array( \
                list(map(lambda u:
                np.array(u), \
                list(map(lambda x: \
                list(map(lambda t: \
                q_fft[-t], x)), \
                angularBuckets)))))

    z_image_bearing = np.exp(-np.complex(0, 1)*omega_0*angularBuckets/c) \
                      * FFTMatrix


    if debug: displayWF(z_image_bearing, 'gray_r', showPhase=False)

    # Final density (complex).
    displayWF(z_image_rx1 * z_image_rx2 * z_image_bearing, 'coolwarm',
   showPhase=False)

    return z_image_rx1 * z_image_rx2 * z_image_bearing


def find_centroid(finalDensity, f=0.7):
    locations = list(map(lambda t: \
                (t[0], t[1]), \
                find_peaks(np.abs(finalDensity), \ threshold=f*np.max(np.abs
   (finalDensity))).to_pandas().values.tolist()))

    xloc = list(map(lambda q: q[0], locations))
    yloc = list(map(lambda q: q[1], locations))

    x_cen = sum(xloc)/len(xloc)
    y_cen = sum(yloc)/len(yloc)

    se = list(map(lambda t: (t[0] - x_cen)**2 + (t[1] - y_cen)**2, locations
   ))

    return x_cen, y_cen, np.sqrt(np.mean(se))
```

For the sake of completeness, we also show the simulation of the wavefield arrival at RX_1, RX_2 by generating random targets in the given imaging patch:

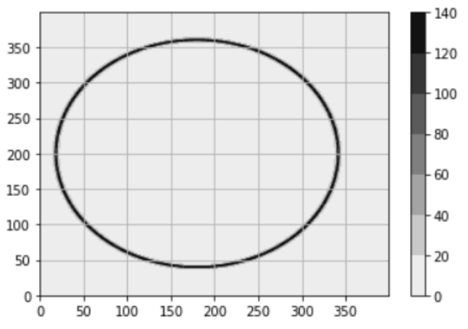
```python
lam0 = 1
x = np.arange(-20*lam0, 20*lam0, lam0/4)
y = np.arange(0, 40*lam0, lam0/4)
xx, yy = np.meshgrid(x, y)

x_loc = list(np.random.randint(-20*lam0, 20*lam0, 30))
y_loc = list(np.random.randint(0, 40*lam0, 30))

target = list(zip(x_loc, y_loc))

M = 15
rTX = (0, 0)
rRX1 = (-M*lam0, 0)
rRX2 = (M*lam0, 0)

```

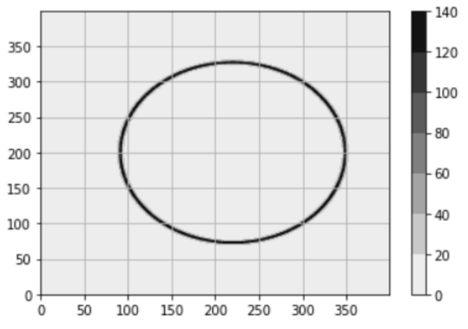
```
lamArray = list(map(lambda q: (128*lam0)/(q + 128), list(range(0, 128))))
for r0 in target:
    plt.xlim(-20*lam0, 20*lam0)
    plt.ylim(0, 40*lam0)
    plt.scatter(r0[0], r0[1])
    plt.show()

    g1 = []
    g2 = []

    for lam in lamArray:
        g1.append(green2D(r0[0] - rTX[0], r0[1] - rTX[1], lam) * \
        green2D(r0[0] - rRX1[0], r0[1] - rRX1[1], lam))
        g2.append(green2D(r0[0] - rTX[0], r0[1] - rTX[1], lam) * \
        green2D(r0[0] - rRX2[0], r0[1] - rRX2[1], lam))

    baselineSystem(g1, g2, M, xx, yy, lam0, lamArray, debug=False)
    time.sleep(1)
    clear_output(wait=True)
```

The output would be a 30 frame video of the target followed by the image reconstruction. We present some specimen samples of the evolution along with discussion on the significance of the parameter M in the next section.

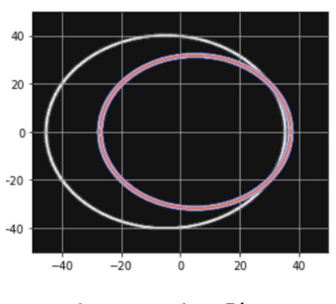
Summary

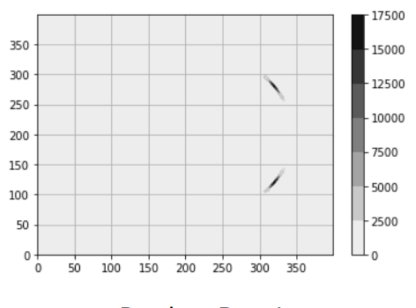
The short baseline system architecture converts the problem of target detection into a geometric problem of the intersection of multiple conics. Given a bunch of transmitters and receivers, we can always pair them up in two useful ways, namely a $TX - RX$ pair or an $RX - RX$ pair (since would need at least one receiver to perform any analysis). Every $TX - RX$ pair can be used to collect the "response" (at an RX) from the objects in the imaging patch (illuminated by a TX). Every $RX - RX$ pair can be used to collect the bearing angle, more specifically, the angle made by the target with the normal drawn to the line connecting the two RXs. In our example, we have only one TX and two RXs and therefore have two $TX - RX$ pairs and one $RX - RX$ pair. Each of the $TX - RX$ pairs, as explained in the first section, allow us to reconstruct the range profile of the target with respect to each receiver's position. The result of such a reconstruction would be a probability density of the location of the target, and in this case, every point that supplies such a "response" to the receiver must satisfy the condition $r_0 + r_1 = k$ where k is the constant dependent on the location of the point. If we set $M = 10$ and assume that the *point object* is at $(30\lambda_0, 20\lambda_0)$, we can expand our imaging patch to $(100\lambda_0 \times 100\lambda_0)$ and note the two ellipses for $TX - RX_1$ and $TX - RX_2$: When these two ellipses intersect,


TX-RX1

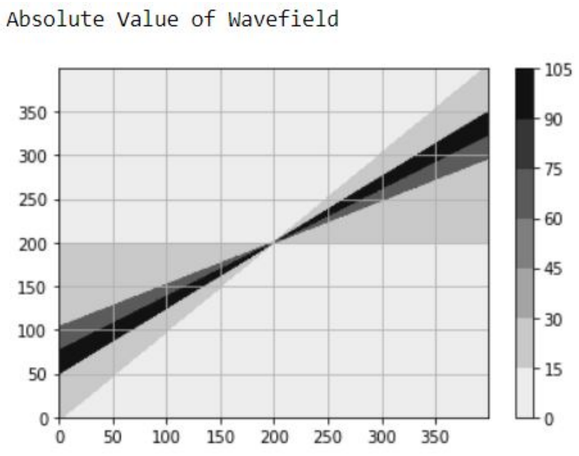

TX-RX2

we claim that have at the most two points of intersection. This is because, for the general case of the two ellipses intersecting in four locations the major axes must have a non-zero angle between them. Since our ellipses share the same major axis (but parallel minor axes) we will have at the most two crossings:

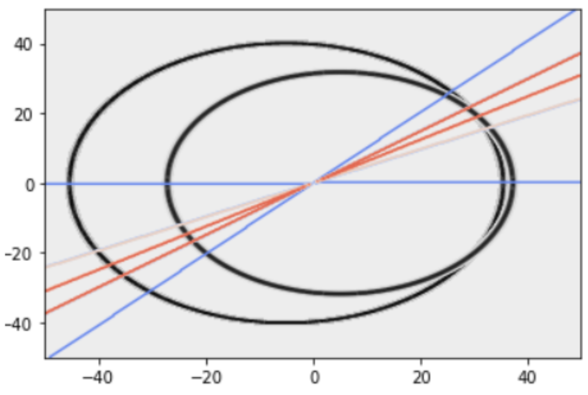

Intersection Plot

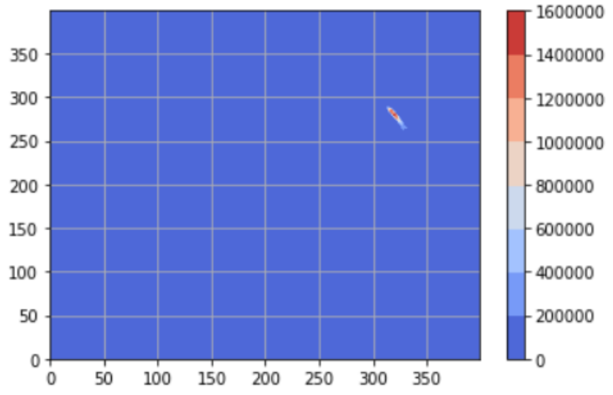

Product Density

Now, we are left with the one $RX - RX$ pair, and we use the FFT based reconstruction (as detailed in the first section) to get the final density plot for the bearing angle estimation. We would mathematically expect a hyperbola but we get a somewhat degenerate form: a pair of intersecting straight lines, which is a conic as well![3] Combining the three plots,



by multiplying the absolute value of each wavefield at each point in the grid we obtain our image. This completes the entire pipeline our image reconstruction strategy. We will

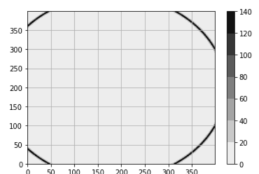

Overlaying the hyperbola


Final Product Density/Image

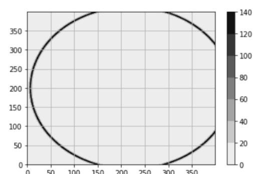
present a few more specimen plots for four different locations of the target followed by a discussion on the significance of varying M.

[3]To wit, a pair of intersecting lines will have the form $L_1L_2 = 0 \implies (y - m_1x - c_1)(y - m_2x - c_2) = 0$ which can be seen as a curve of genus 2, i.e., a conic.

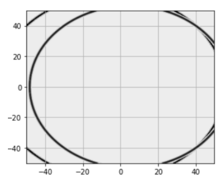
$$M = 10, x_{target} = 40\lambda_0, y_{target} = 40\lambda_0$$

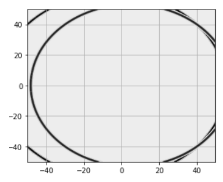

RX1-TX

X

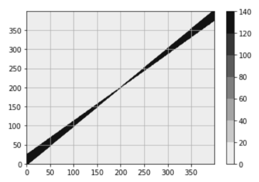

RX2-TX

=

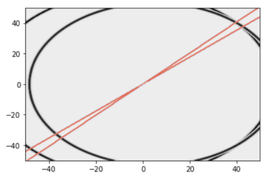

F1 X F2

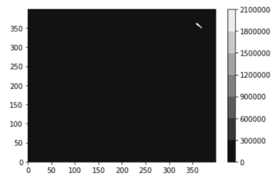

F1 X F2

X

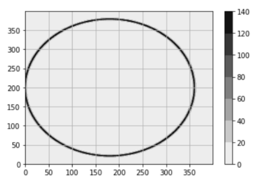

RX1-RX2

=

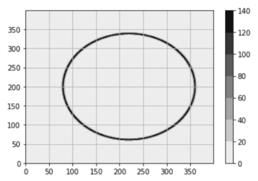

F1 X F2 X F3


FINAL IMAGE

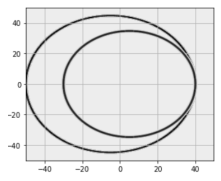
$$M = 10, x_{target} = 40\lambda_0, y_{target} = 2\lambda_0$$

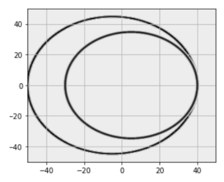

RX1-TX

X

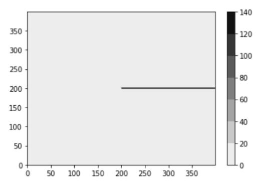

RX2-TX

=

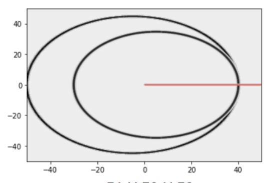

F1 X F2

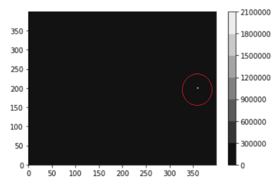

F1 X F2

X

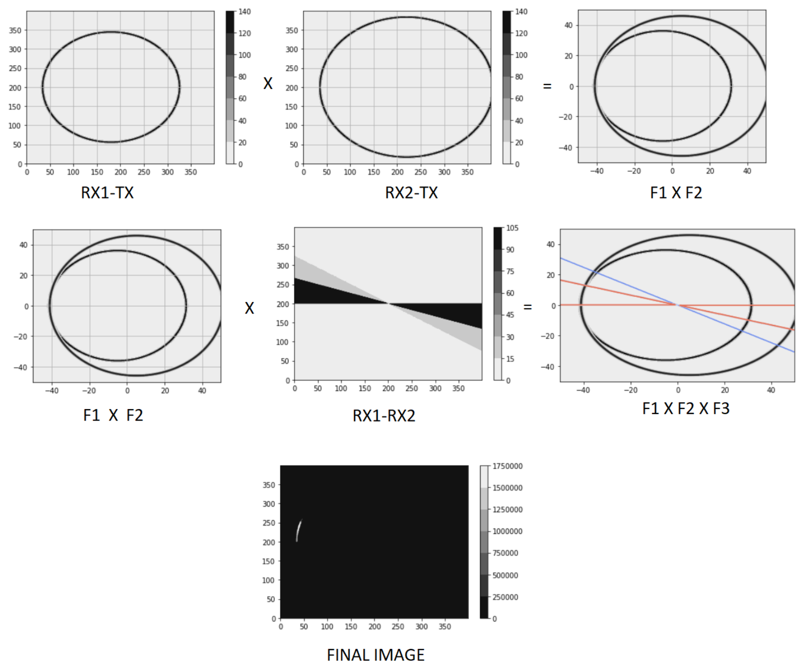

RX1-RX2

=

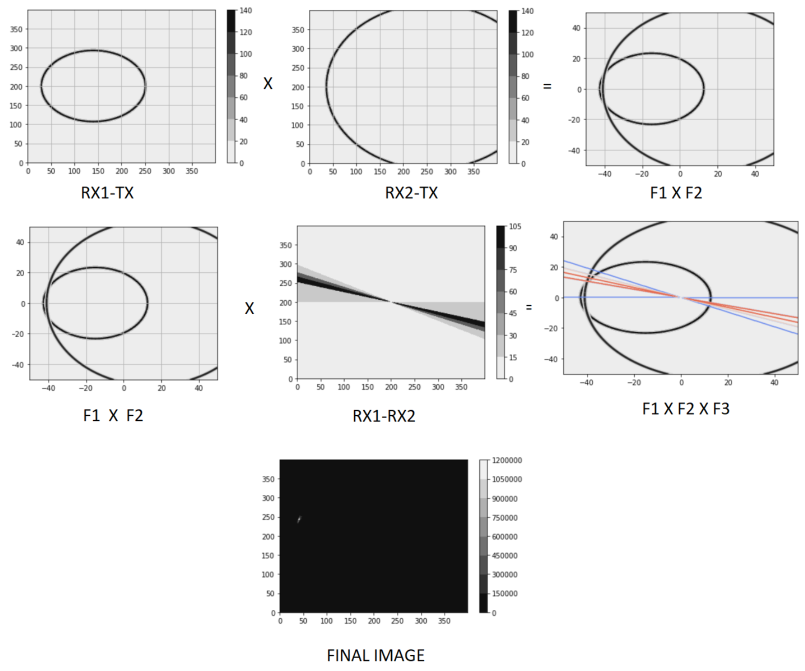

F1 X F2 X F3


FINAL IMAGE

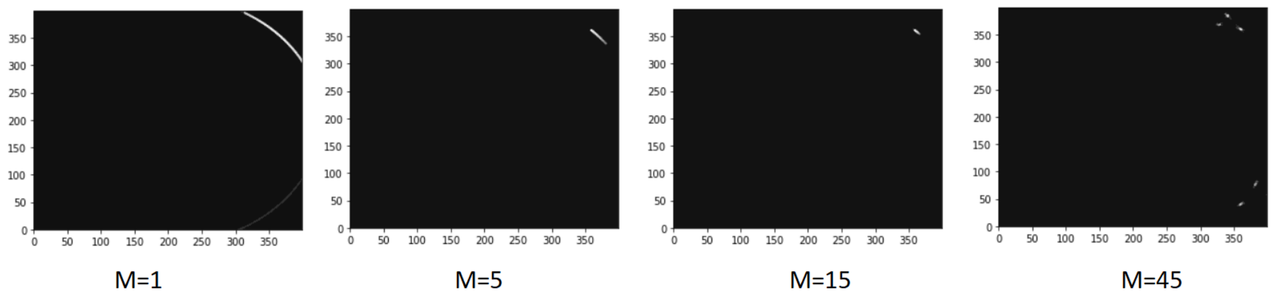
$$M = 10, x_{target} = -40\lambda_0, y_{target} = 10\lambda_0$$



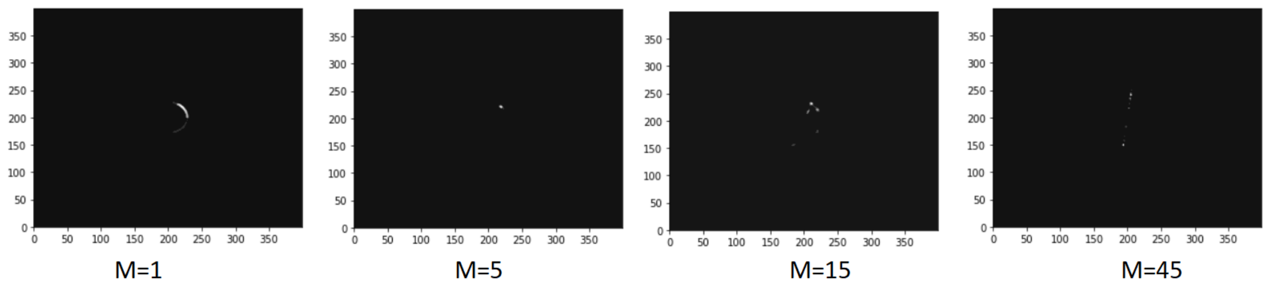
$$M = 30, x_{target} = -40\lambda_0, y_{target} = 10\lambda_0$$



The effect of varying M can be witnessed in the last two examples. The variation of reconstructed image for the target at $(40\lambda_0, 40\lambda_0)$ is shown below:



Clearly, while the range estimation looks decent for low M, the angular dispersion is quite high and this gets better as M is increased, until a point when range ambiguity takes over. At this point we propose an improvement where we take the centroid of the hotspots to improve the estimation since at high M the far-field approximation starts to degrade significantly. For an intrinsically near field target, we see similar inversion in performance (target is at $(5\lambda_0, 5\lambda_0)$):



If our hardware included a mechanism to vary M while collecting the snapshots, we could design a feedback loop to correct the error in M such that the total area occupied by the hotspots is localized in a circle of a given radius r_0 (tolerance limit). From the experiments shown above it seems that for this system the far field radius for accurate target estimation seems to be approximately $3M\lambda_0$. Thus the future scope of this project would involve analyzing the performance of the feedback (considering the delay et cetera) and trying to theoretically estimate the Fraunhofer limit for the $RX - TX - RX$ aperture as a whole.