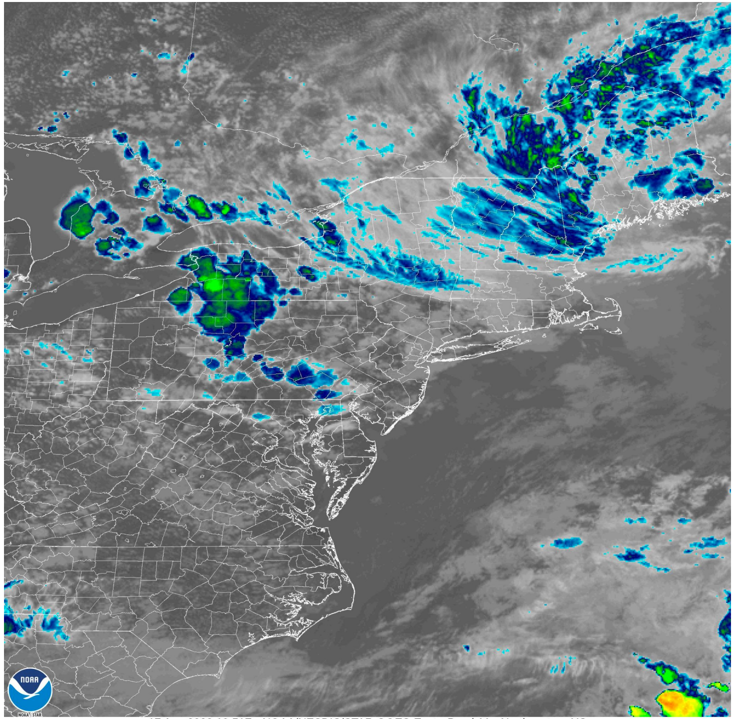


17 Aug 2022 19:51Z - NOAA/NESDIS/STAR GOES-East - Band 11 - Northeastern US

# Estimating Wind Speeds via Cloud Cover Imagery from GOES-16

Whiting School of Engineering, Johns Hopkins
ROMITA BISWAS

# Background

- Motivation for this project is to be able to analyze satellite imagery to determine if an area is a good place to install a wind turbine
  - Scope of this project
    - Analyze satellite imagery to measure wind speeds
      - Research various GOES satellites
      - Research various tracking and integration methods
      - Understand wind tracking algorithm used by NOAA today
      - Replicate or use own motion tracking algorithm
    - Propose a method to further research to analyze and predict wind speeds globally
      - Machine Learning solution for time series images
      - Include spatial angular radians when modelling wind - 2D to 3D

# Research Papers

- Methods to model wind speeds:
  - Use GOES-15 brightness temperature data to model wind motions as a spatial process drifting in time [5]
  - Uses GOES-R with ABI sensor to track important features like wind and estimate target height [2]
  - Uses hindcast data produced by ECMWF to model motions as a hidden Markov chain capturing motions dependent on position of principal air masses [1]
  - ECMWF wind imagery is extracted into large and small components. The large scale component is modelled as an AR process and predictions are made using Kalman filter [4]
  - Use water vapor data alongside cloud displacement data to make short range predictions (ECMWF) [7]

# Properties of Wind

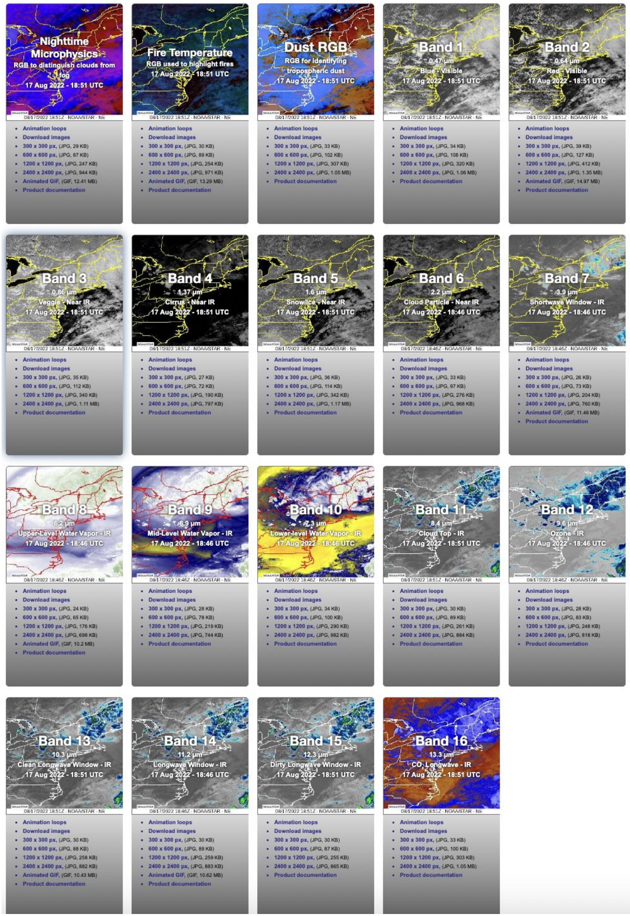
- Wind speed is derived using a sequence of visible or IR spectral bands to track the motion of cloud features and water vapor gradients
  - Cloud Movement
  - Cloud Height
  - Vapor
  - Air Mass
- For this project only cloud cover motion images were used
- Different heights correspond to different bands of the satellite
- Wind at different heights is affected by different properties
  - Mid- to upper tropospheric level: (6.7µm - 7.3µm) water vapor channels and longwave (10.7µm) infrared (LWIR) channel
  - Lower tropospheric level: a combination of the visible (VIS) and IR channel in daytime
  - Lower tropospheric level: shortware (3.9µm) infrared (SWIR) channel compliments the LWIR channel during nighttime. Shortwave is more sensitive to warmer temperature features

# GOES-16 Satellite

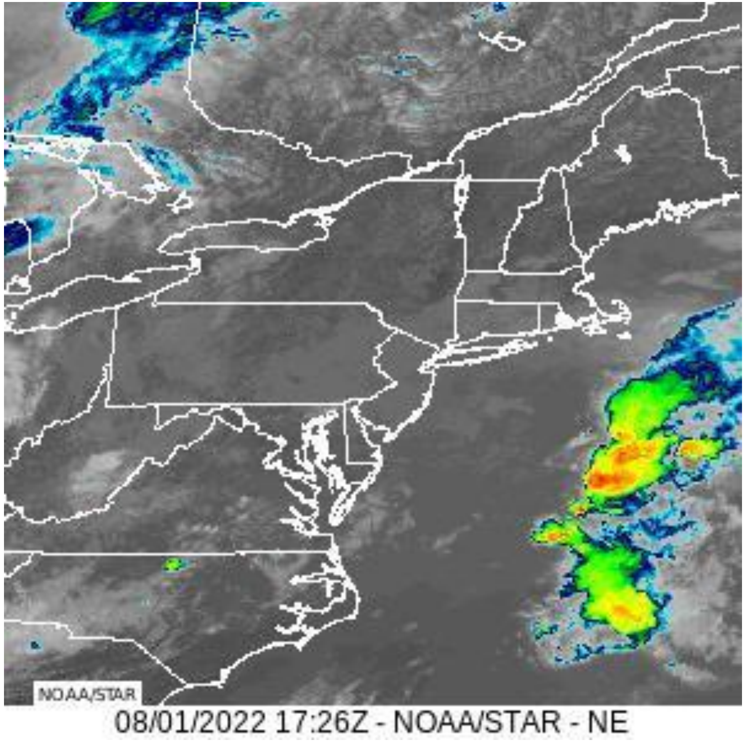
- Operational geostationary weather satellite in the GOES East position at 75.2°W
- Advanced Baseline Imager provides high resolution imagery through 16 spectral bands 1372 × 1300 pixel staring CCD sensitive to 777.4 nm light
  - ABI has specific bands to specialize in detecting different types of vapor and cloud cover of IR and spectral bands
- Band 11 for 300x300 pixels: pixel footprint ≃ 4km
- Orbit Height: 35,786 km
- Image taken every 5 minutes in continuous mode otherwise 15
- Centered over Americas

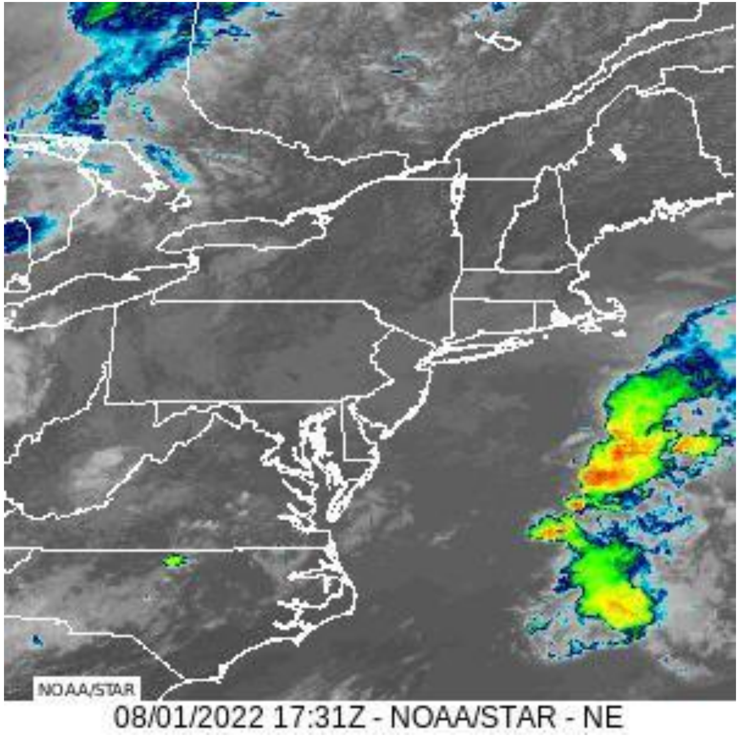
# Bands of the ABI

- ABI has 16 bands detecting various weather features through IR and spectral bands
- Each wavelength corresponds to a height in the atmosphere
  - Can detect: cloud tops, fog, dust, wildfire smoke, volcanic ash, water vapor
- Why Band 11?
  - Band of 8.5 µm there is little atmospheric absorption of energy in clear skies at this wavelength
  - Brightness temperatures will be modulated by water vapor
  - Movement of cloud top is best representation of clouds moving at upper tropospheric levels
  - Source [3]



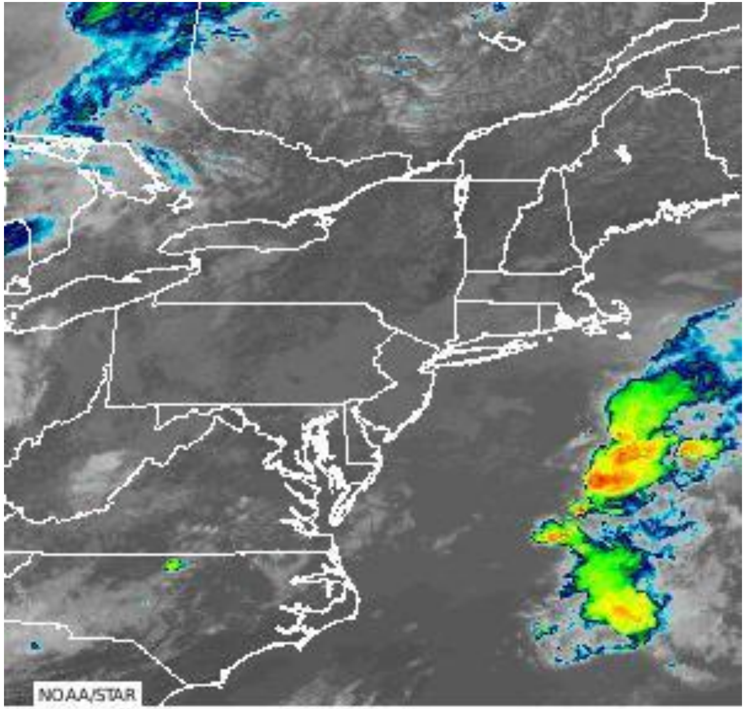
# Examples of Images taken over North America



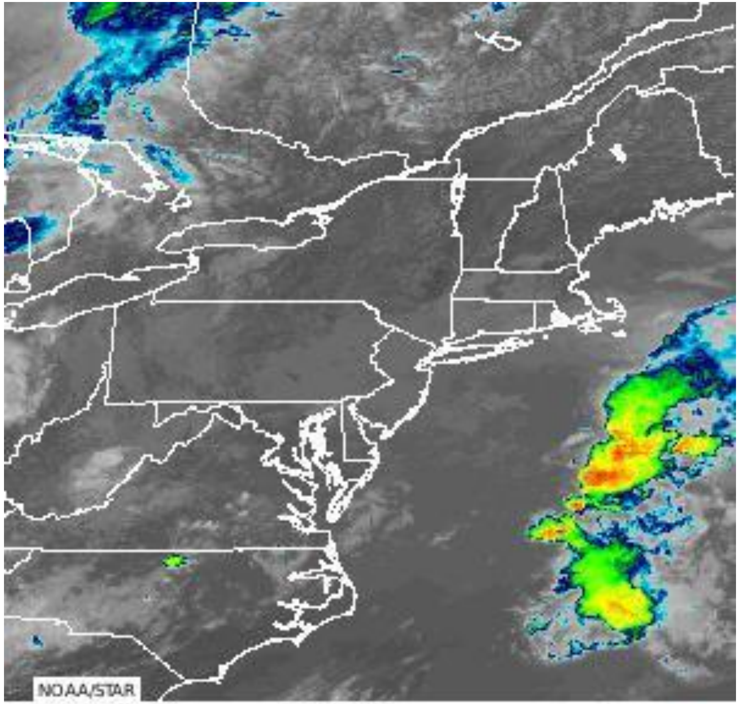




Source [3]

# Estimate Wind speed by tracking the movement of this cloud: taken 5 minutes apart




08/01/2022 17:26Z - NOAA/STAR - NE




08/01/2022 17:31Z - NOAA/STAR - NE

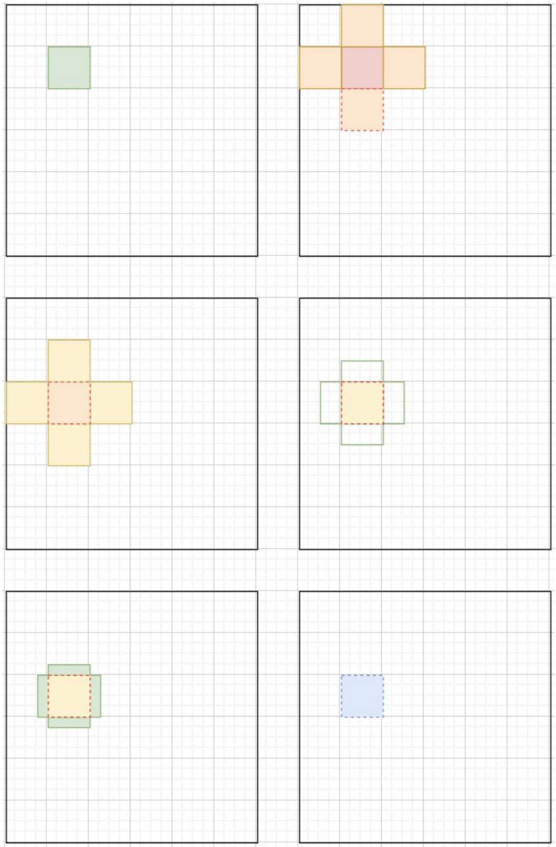
Source [3]

# Motion Estimation Algorithm: 2D Log Search

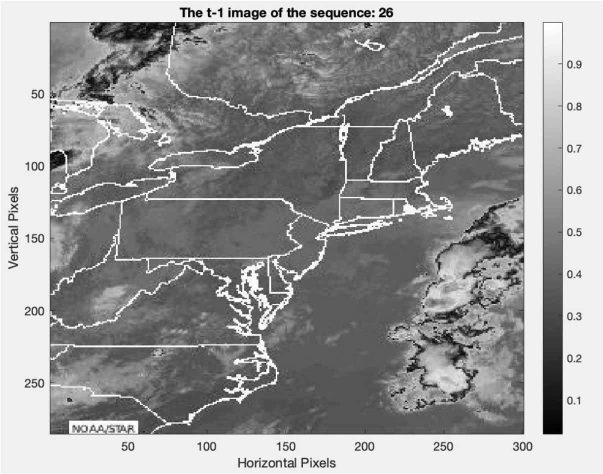
- 2D log search is fast
- Search in all 8 directions surrounding reference block
- Time Complexity: O(logMN), N = search step size, M = target block size
- Algorithm [6]:
    - Select an initial step size N
    - Search for 4 locations at a distance of N from center on the X and Y axes
    - Find the location of point with least cost function
    - If a point other than center is the best matching point,
        - Select this point as the new center
        - Search for 4 locations at a distance of N from new center on the X and Y axes
    - If the best matching point is at the center, set S = S/2
    - If S = 1, all 8 locations around the center at a distance S are searched
    - Set the motion vector as the point with least cost function

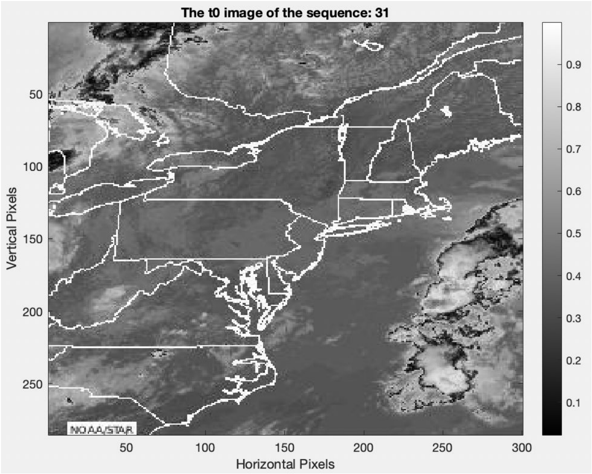
# Motion Estimation Algorithm: 2D Log Search

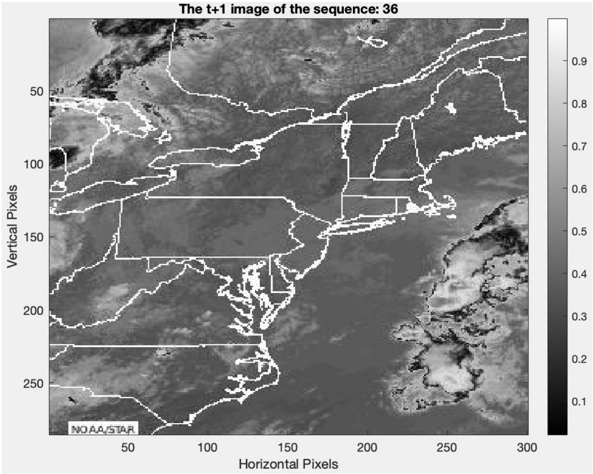
```
for anchor_x = start_x:block_size:end_x
       for anchor_y = start_y:block_size:end_y
           while (current_step > 1)
               initial_error = inf(1, 5);

               %CENTER
              …find center cost
               %WEST - can we go negative step size on x axis?
              …find west cost
               %EAST - can we go positive step size on x axis?
               …find east cost
               %SOUTH - e we go positive step size on x axis?
          …find south cost
               %NORTH
               … find north cost
               %Find minimum cost and update center
               [value, index] = min(initial_error);
               if (index == 1)
                   current_step = current_step/2;
               else
                   target_x = updated_x(index);
                   target_y = updated_y(index);
               end
           end
           matchblock_error(mesh_y, mesh_x) = value;
           matchblock_x(mesh_y, mesh_x) = target_x - anchor_x;
           matchblock_y(mesh_y, mesh_x) = target_y - anchor_y;
       end
   end
```



# Sequence of Wind Images

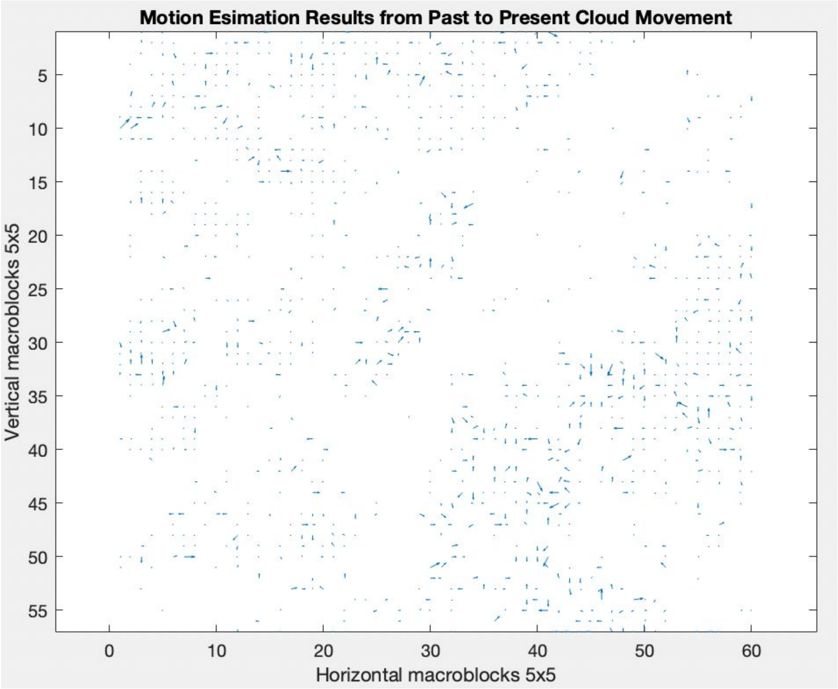


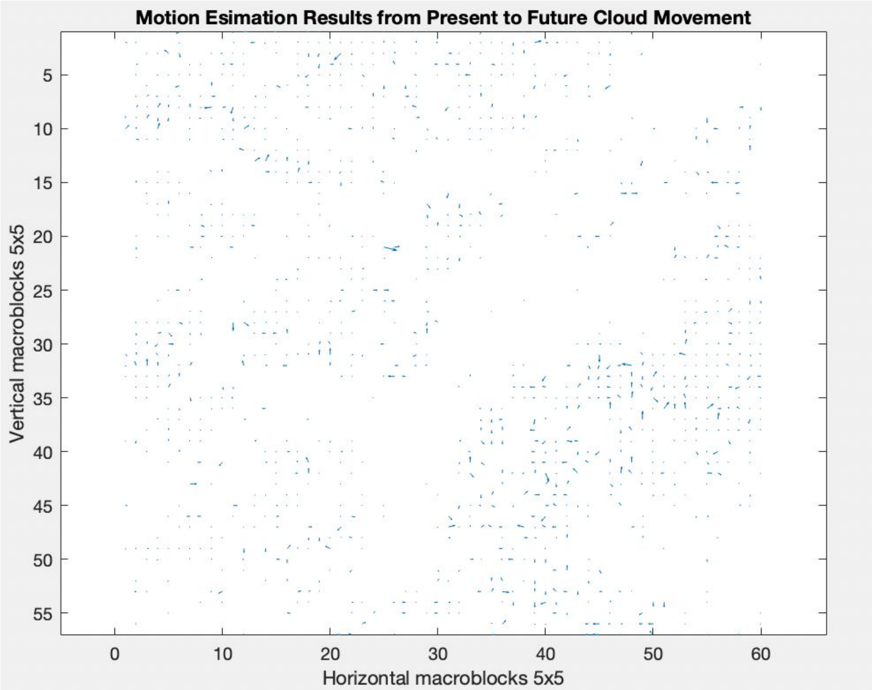



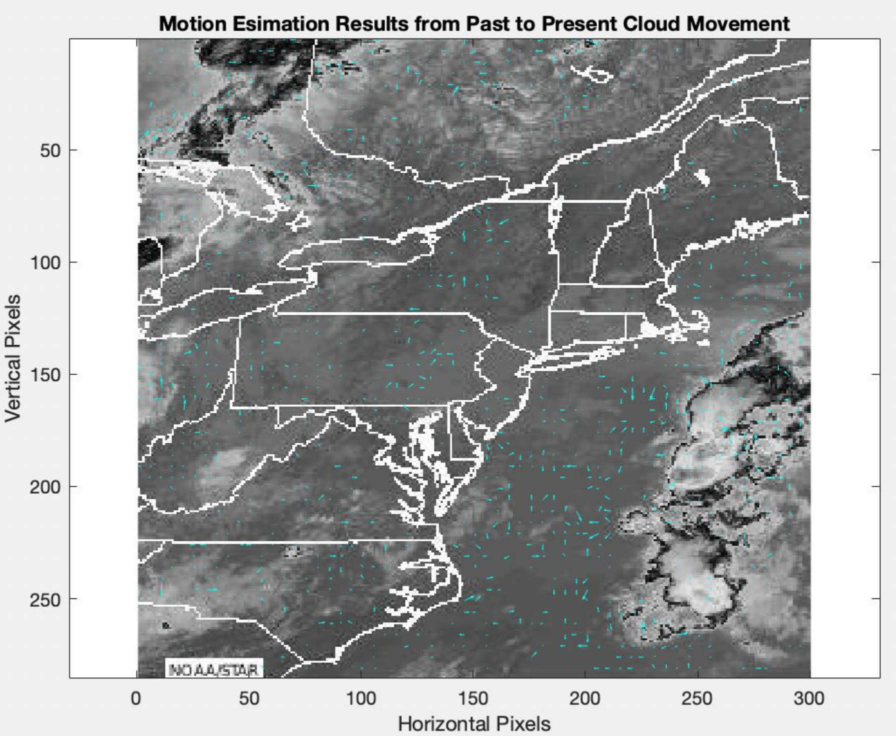

Source [3]

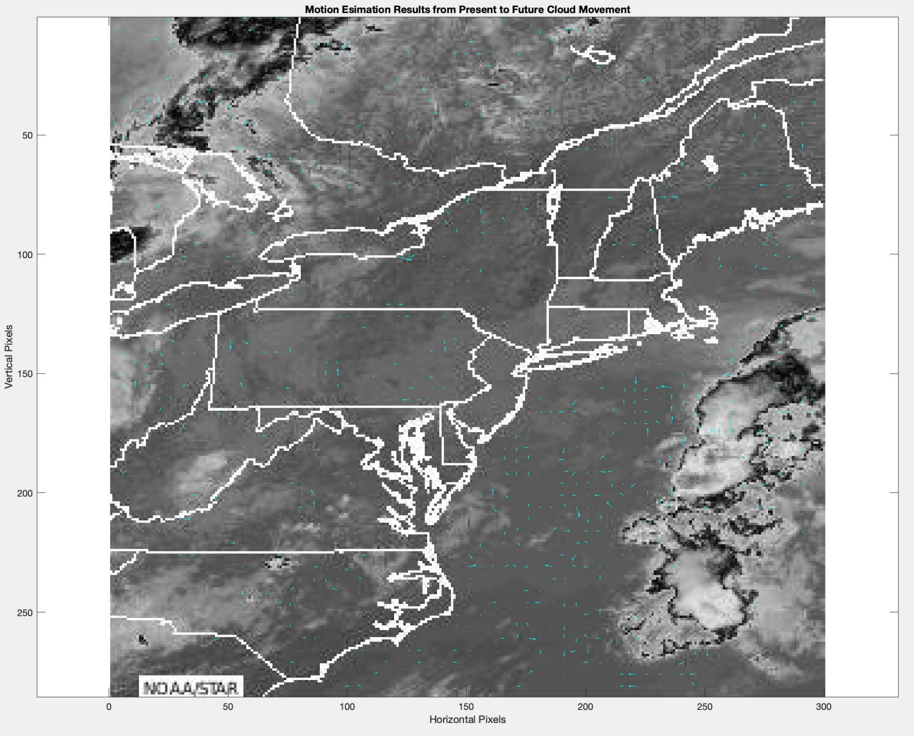
# Generated Wind Fields



# Generated Wind Fields on top of Images



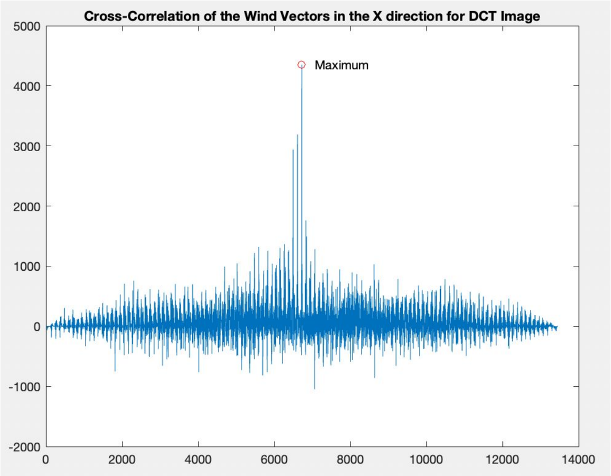



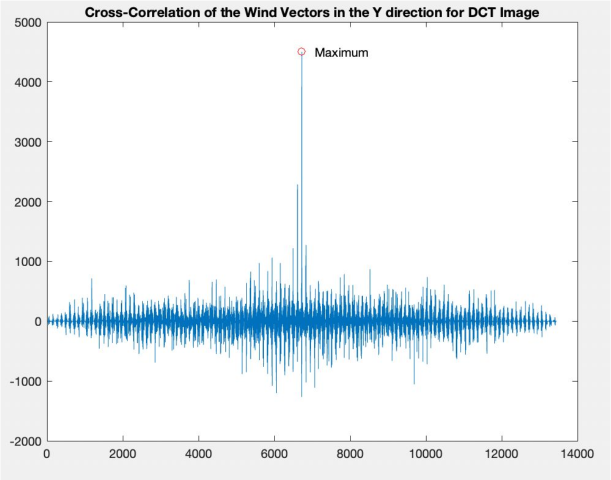
# Enhanced vs Unenhanced Images

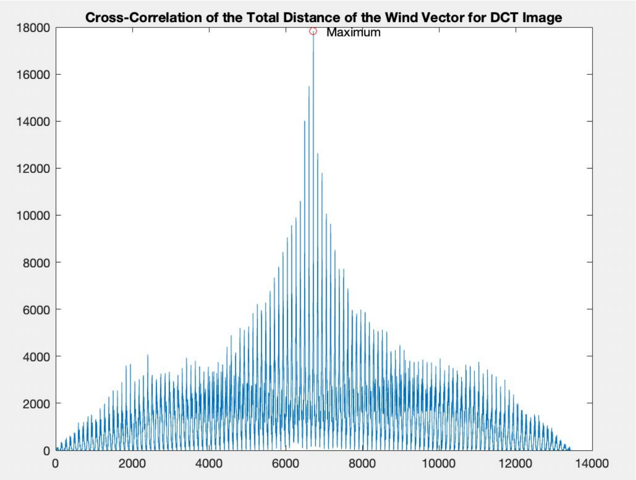
- Performed the same block search algorithm on enhanced images: DCT and imerode
- DCT cleaned images had the lowest total mean absolute error between the reference images and target images
- DCT cleaned images had high frequency zeroed out
- DCT cleaned images showed highest cross correlation between wind speeds or rms value of wind vectors
    - *Refer to Appendix Slides for visualization of results*

# Correlation as a performance criteria for 2D log search

- Calculated the correlation of wind speeds and wind directions between the 2 generated wind fields
- High cross correlation between time steps for total distance of wind vector - speed



# How can wind speed be modeled?

- Unable to model 2D wind fields as autoregressive functions - not within scope of this project
- It can be modeled as a time series
- Challenge is non linear temporal and spatial changes
- No correlation shown between x and y directions since directions change quickly
    - Wind speed appears predictable using wind field images
    - Wind directions do not appear predictable
    - Idea: spatial angle may need to be taken into consideration or rotation matrix of each vector could be quantified and modeled

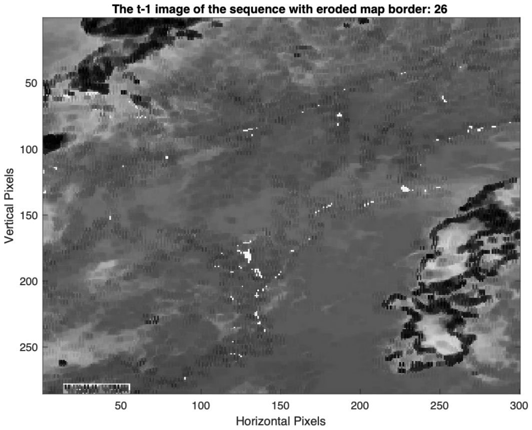
# Further Research Ideas

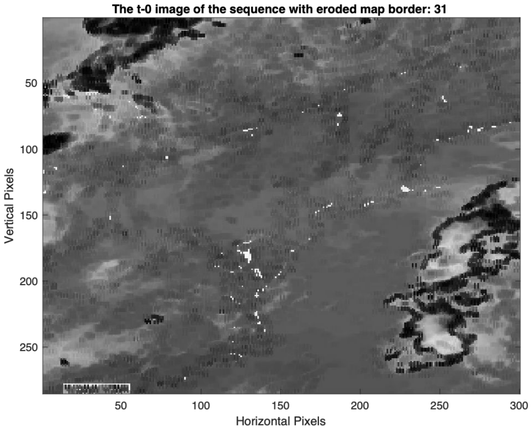
- Model this is an AutoRegressive process
  - Issue is spatial and temporal movement
  - Planning to use PCA to reduce dimensionality
- Incorporate wind vapor images and cloud height into analysis to predict wind speeds at various atmospheric heights
- Global prediction has not been done
- Spatial radian measure around globe should be included as parameter to model wind around globe

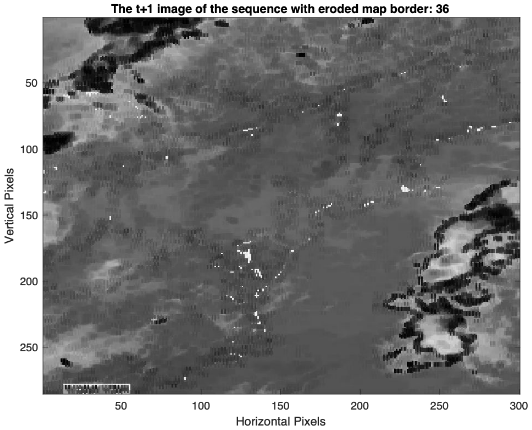
# References


1. Ailliot, P., Monbet, V., & Prevosto, M. (n.d.). *An autoregressive model with time-varying coefficients for wind fields ...* Retrieved August 17, 2022, from https://www.researchgate.net/publication/29490452_An_autoregressive_model_with_time-varying_coefficients_for_wind_fields
2. Daniels, J., Bresky, W., Wanzong, S., Velden, C., & Berger, H. (n.d.). *Center for Satellite Applications and research - NOAA / NESDIS / star*. NOAA / NESDIS / STAR website. Retrieved August 17, 2022, from https://www.star.nesdis.noaa.gov/goesr/documentation_ATBDs.php
3. *Goes image viewer - center for satellite applications and research - NOAA/NESDIS/star*. GOES Image Viewer - NOAA/NESDIS/STAR website. (n.d.). Retrieved August 17, 2022, from https://www.star.nesdis.noaa.gov/GOES/sector.php?sat=G16§or=ne
4. Malmberg, A., Holst, U., & Holst, J. (2004, November 30). *Forecasting near-surface ocean winds with Kalman filter techniques*. Ocean Engineering. Retrieved August 17, 2022, from https://www.sciencedirect.com/science/article/pii/S0029801804001556
5. Sahoo, I., Guinness, J., & Reich, B. J. (2021, June 15). *Estimating atmospheric motion winds from satellite image data using space-time drift models*. arXiv.org. Retrieved August 17, 2022, from https://arxiv.org/abs/1902.09653
6. Saikia, M., & Choudhury, H. A. (n.d.). *Comparative study of block matching algorithms for motion estimation*. Research Gate. Retrieved August 17, 2022, from https://www.researchgate.net/publication/259183926_COMPARATIVE_STUDY_OF_BLOCK_MATCHING_ALGORITHMS_FOR_MOTION_ESTIMATION
7. Tomassini, M., Kelly, G., & Saunders, R. (1999, June 1). *Use and impact of satellite atmospheric motion winds on ECMWF analyses and forecasts*. AMETSOC. Retrieved August 17, 2022, from https://journals.ametsoc.org/view/journals/mwre/127/6/1520-0493_1999_127_0971_uaiosa_2.0.co_2.xml

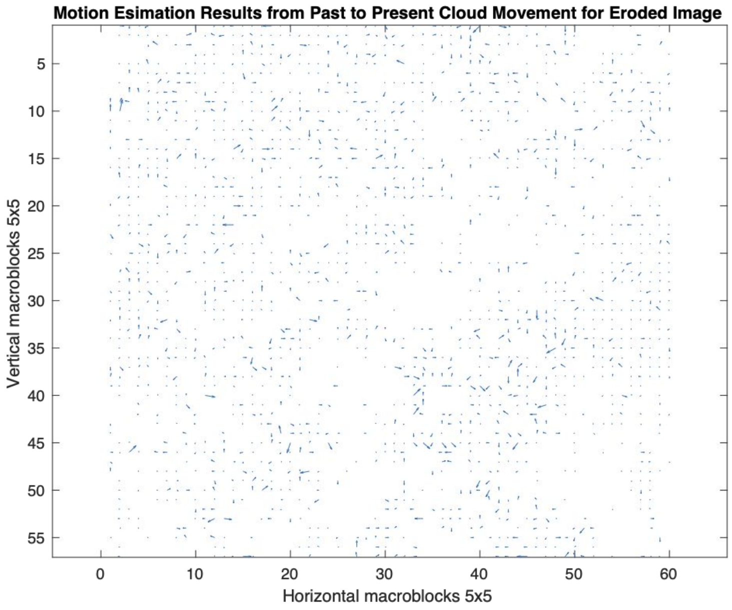
# Sequence of Eroded Wind Images

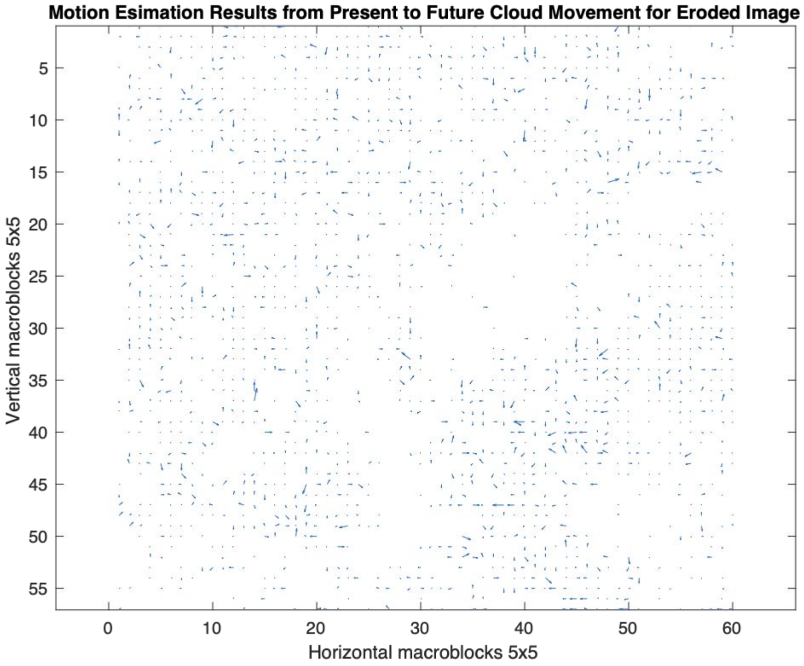


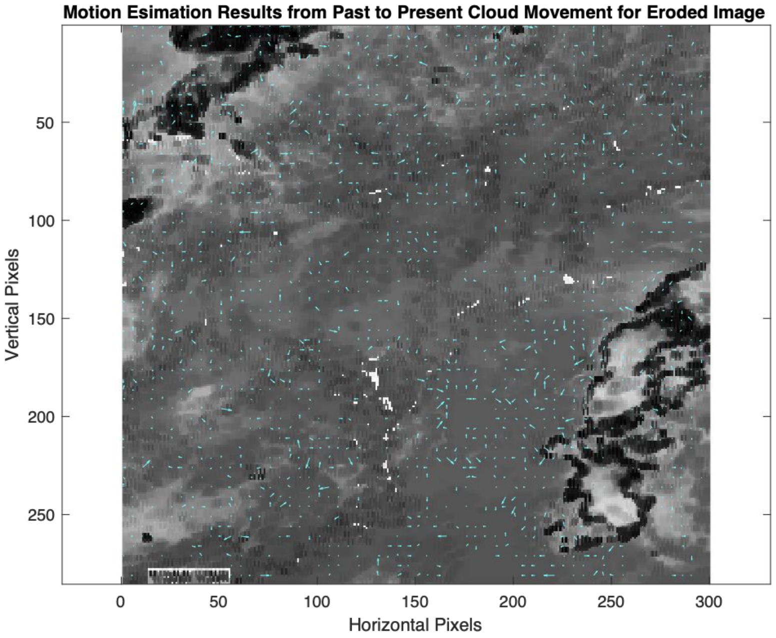



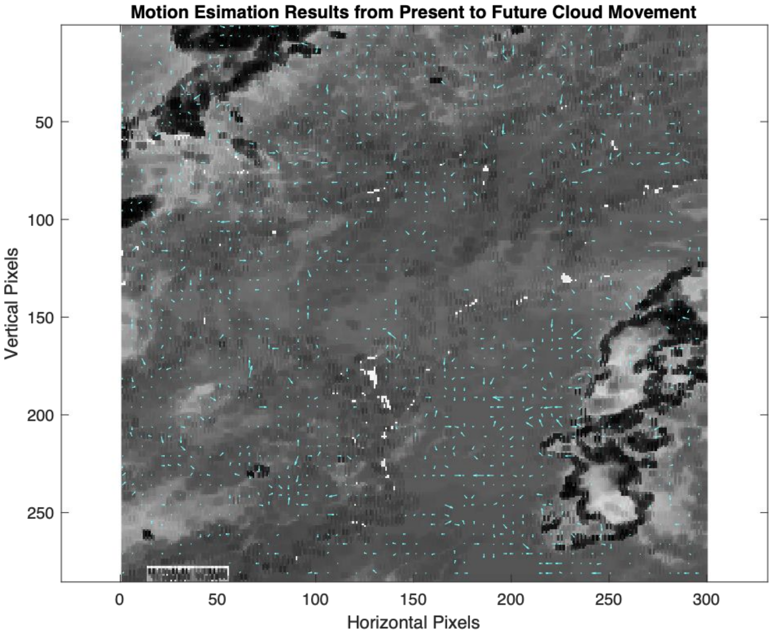
# Generated Wind Fields for Eroded Images



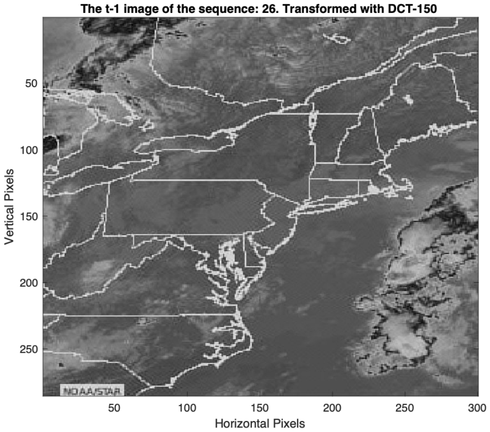



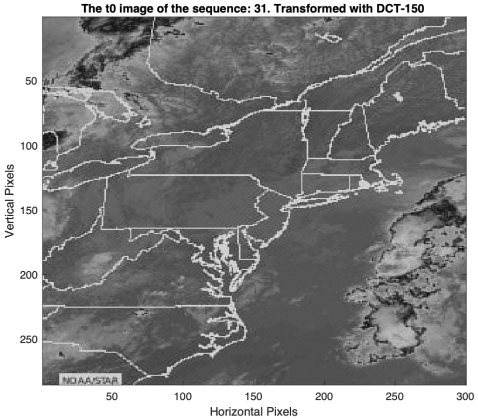
# Generated Wind Fields on top of Eroded Images

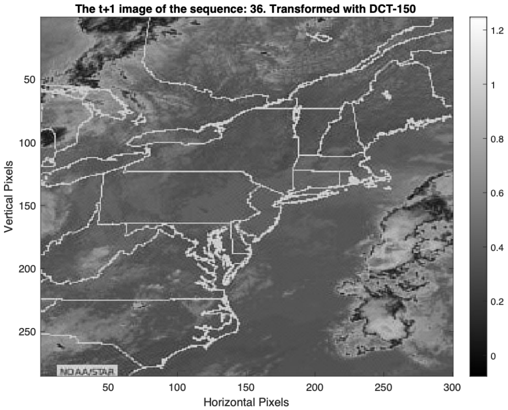


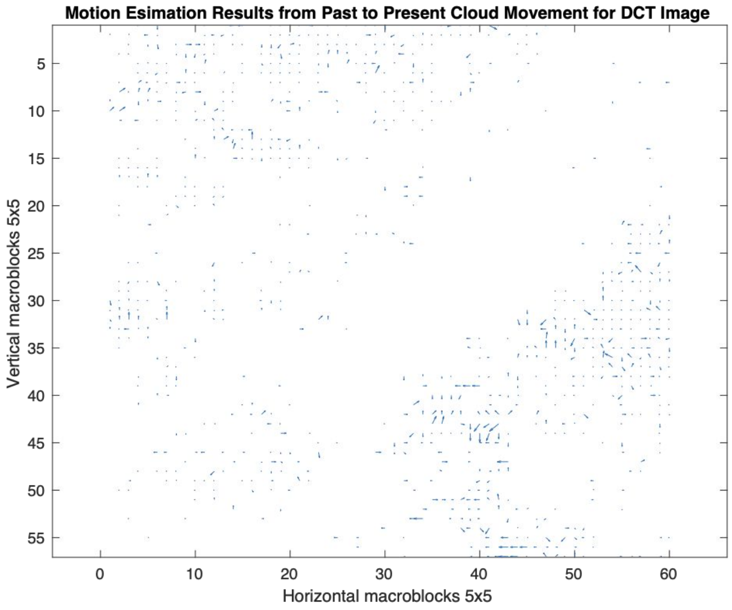
# Sequence of DCT Cleaned Wind Images

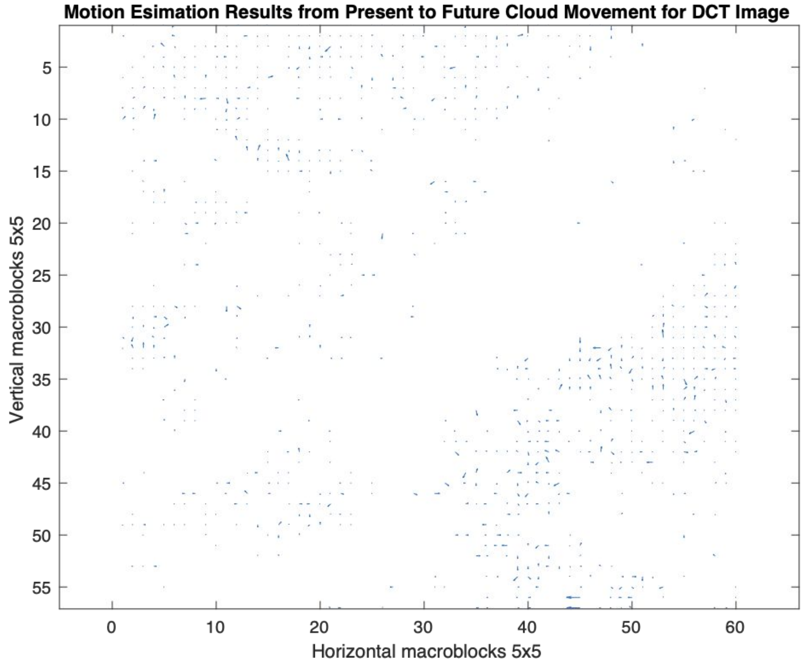


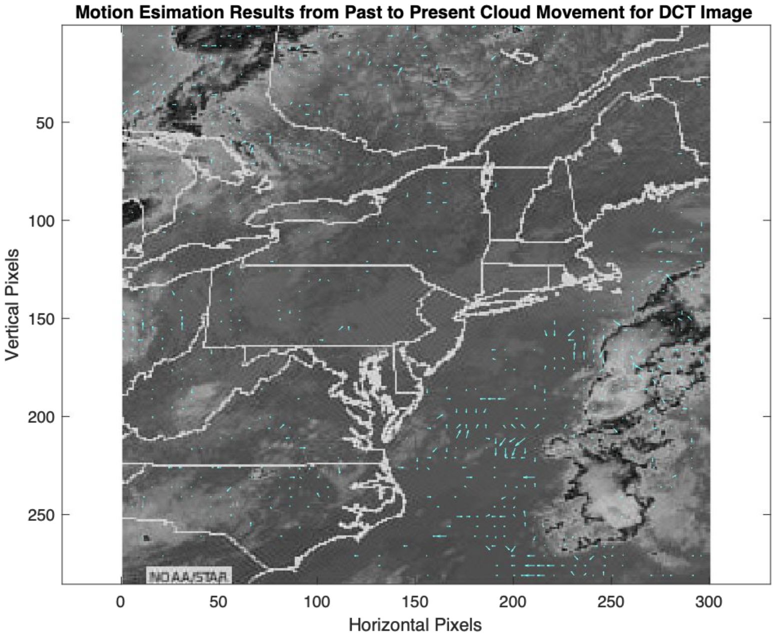



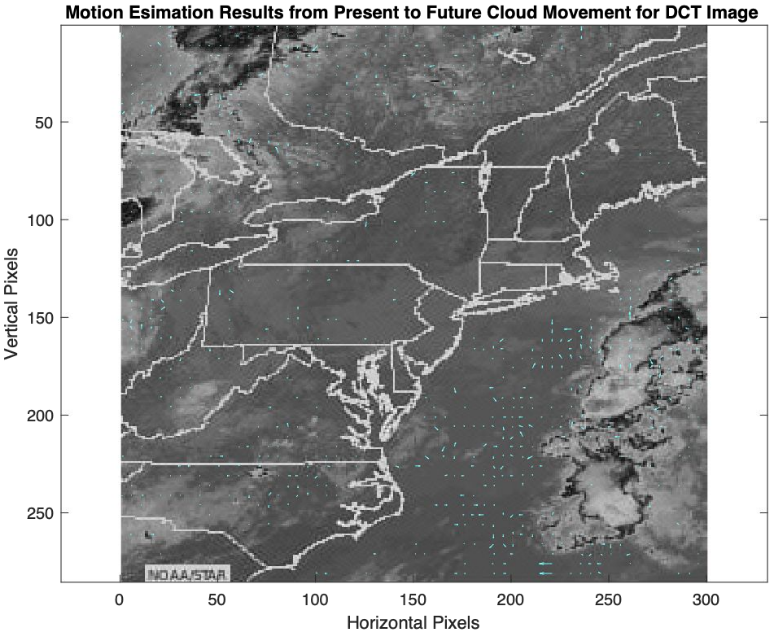
# Generated Wind Fields of DCT Cleaned Images



# Generated Wind Fields on top of DCT Cleaned Images



# Code to find each direction - Center

```
function [error, y, x] = analyze_center(anchor_block, target_image, ...
    block_size, previous_y, previous_x)

    dist = (block_size - 1)/2;
    target_block = target_image([previous_y - dist : previous_y + dist], ...
        [previous_x - dist : previous_x + dist]);
    error = mean(abs(anchor_block - target_block), 'all');
    y = previous_y;
    x = previous_x;

end
```

# Code to find each direction - East

```
function [error, y, x] = analyze_east(anchor_block, target_image, ...
    search_step, block_size, previous_y, previous_x)

    dist = (block_size - 1)/2;
    current_x = previous_x + search_step;
    target_block = target_image([previous_y - dist : previous_y + dist], ...
        [current_x - dist : current_x + dist]);
    error = mean(abs(anchor_block - target_block), 'all');
    y = previous_y;
    x = current_x;

end
```

# Code to find each direction - West

```
function [error, y, x] = analyze_west(anchor_block, target_image, ...
    search_step, block_size, previous_y, previous_x)

    dist = (block_size - 1)/2;
    current_x = previous_x - search_step;
    target_block = target_image([previous_y - dist : previous_y + dist], ...
        [current_x - dist : current_x + dist]);
    error = mean(abs(anchor_block - target_block), 'all');
    y = previous_y;
    x = current_x;

end
```

# Code to find each direction - North

```
function [error, y, x] = analyze_north(anchor_block, target_image, ...
    search_step, block_size, previous_y, previous_x)

    dist = (block_size - 1)/2;
    current_y = previous_y - search_step;
    target_block = target_image([current_y - dist : current_y + dist], ...
        [previous_x - dist : previous_x + dist]);
    error = mean(abs(anchor_block - target_block), 'all');
    y = current_y;
    x = previous_x;

end
```

# Code to find each direction - South

```
function [error, y, x] = analyze_south(anchor_block, target_image, ...
    search_step, block_size, previous_y, previous_x)

    dist = (block_size - 1)/2;
    current_y = previous_y + search_step;
    target_block = target_image([current_y - dist : current_y + dist], ...
        [previous_x - dist : previous_x + dist]);
    error = mean(abs(anchor_block - target_block), 'all');
    y = current_y;
    x = previous_x;

end
```

# Code to find matched block

```
function [matchblock_x, matchblock_y, matchblock_error] = find_matching_block( ...
    search_step, block_size, mesh_rows, mesh_cols, past_image, present_image)

%The find matching block function searches for the best matching block
%using a 2 dimensional logarithmic search. It searches in 4 directions -
%positive negative of x and y axes in steps of search_step. The error is
%calculated by taking the absolute mean difference between the reference
%block in the present image and the target block in the current image. The
%direction with the lowest error is chosen as the new target block. If the
%center is the lowest error it means the matched block is closer within
%that range that the other directions. We then half the step size and
%repeat the same process of searching in 4 directions until our step size
%is 1. Once our step size is 1 we have searched in all relevant directions
%of our reference block.

    matchblock_error = ones(mesh_rows, mesh_cols);
    matchblock_x = ones(mesh_rows, mesh_cols);
    matchblock_y = ones(mesh_rows, mesh_cols);
    dist_center = (block_size - 1)/2;

    start_x = 1 + dist_center;
    end_x = (block_size * mesh_cols) - dist_center;
    end_y = (block_size * mesh_rows) - dist_center;
    start_y = 1 + dist_center;
    mesh_x = 0;
    mesh_y = 0;
    for anchor_x = start_x:block_size:end_x
        mesh_x = mesh_x + 1;
        mesh_y = 0;
        for anchor_y = start_y:block_size:end_y
            mesh_y = mesh_y + 1;
            target_x = anchor_x;
            target_y = anchor_y;
            current_step = search_step;
            anchor_block = present_image([anchor_y - dist_center: ...
                anchor_y + dist_center], [anchor_x - dist_center:anchor_x + dist_center]);
```

```
while (current_step > 1)
    initial_error = inf(1, 5);
    updated_x = inf(1, 5);
    updated_y = inf(1, 5);
    %CENTER
    [initial_error(1), updated_y(1), updated_x(1)] = analyze_center( ...
        anchor_block, past_image, block_size, target_y, target_x);
    %WEST - can we go negative step size on x axis?
    if ((target_x - current_step) >= start_x)
        [initial_error(2), updated_y(2), updated_x(2)] = analyze_west( ...
            anchor_block, past_image, current_step, block_size, target_y, target_x);
    end
    %EAST - can we go positive step size on x axis?
    if ((target_x + current_step) <= end_x)
        [initial_error(3), updated_y(3), updated_x(3)] = analyze_east( ...
            anchor_block, past_image, current_step, block_size, target_y, target_x);
    end
    %SOUTH
    if ((target_y + current_step) <= end_y)
        [initial_error(4), updated_y(4), updated_x(4)] = analyze_south( ...
            anchor_block, past_image, current_step, block_size, target_y, target_x);
    end
    %NORTH
    if ((target_y - current_step) >= start_x)
        [initial_error(5), updated_y(5), updated_x(5)] = analyze_north( ...
            anchor_block, past_image, current_step, block_size, target_y, target_x);
    end
```

# Code to find matched block

```
                %Once all the major direction errors have been computed, we
                %choose the smallest one to be the new center that we
                %search from. If the center is the least we are close, half the
                %step_size. If not update the center.
                [value, index] = min(initial_error);
                if (index == 1)
                    current_step = current_step/2;
                else
                    target_x = updated_x(index);
                    target_y = updated_y(index);
                end
            end
            matchblock_error(mesh_y, mesh_x) = value;
            matchblock_x(mesh_y, mesh_x) = target_x - anchor_x;
            matchblock_y(mesh_y, mesh_x) = target_y - anchor_y;
        end
    end
end
```