Projet Imagerie Médicale

Windowed Radon Transform and Tensor Rank-1 Decomposition for Adaptive Beamforming in Ultrafast Ultrasound

S. Beuret and JP. Thiran

Oscar Gledel & Emilien Sanchez, UE Traitement du Signal et Applications en Imagerie, M2 IAFA

Objectifs du Projet :

- Lire, comprendre et analyser l'article
- Comprendre et analyser le code fourni en complément de l'article
- Reproduire les résultats obtenus dans à l'aide du code et des données fournies en complément de l'article

Objectifs du Projet :

- Lire, comprendre et analyser l'article
- Comprendre et analyser le code fourni en complément de l'article
- Reproduire les résultats obtenus dans à l'aide du code et des données fournies en complément de l'article

Windowed Radon Transform and Tensor Rank-1 Decomposition for Adaptive Beamforming in Ultrafast Ultrasound
S. Beuret and JP. Thiran

- Corriger le phénomène d'**aberration de phase** en imagerie US-UR
- Phénomène dû à la supposition sur la **vitesse constante** du son

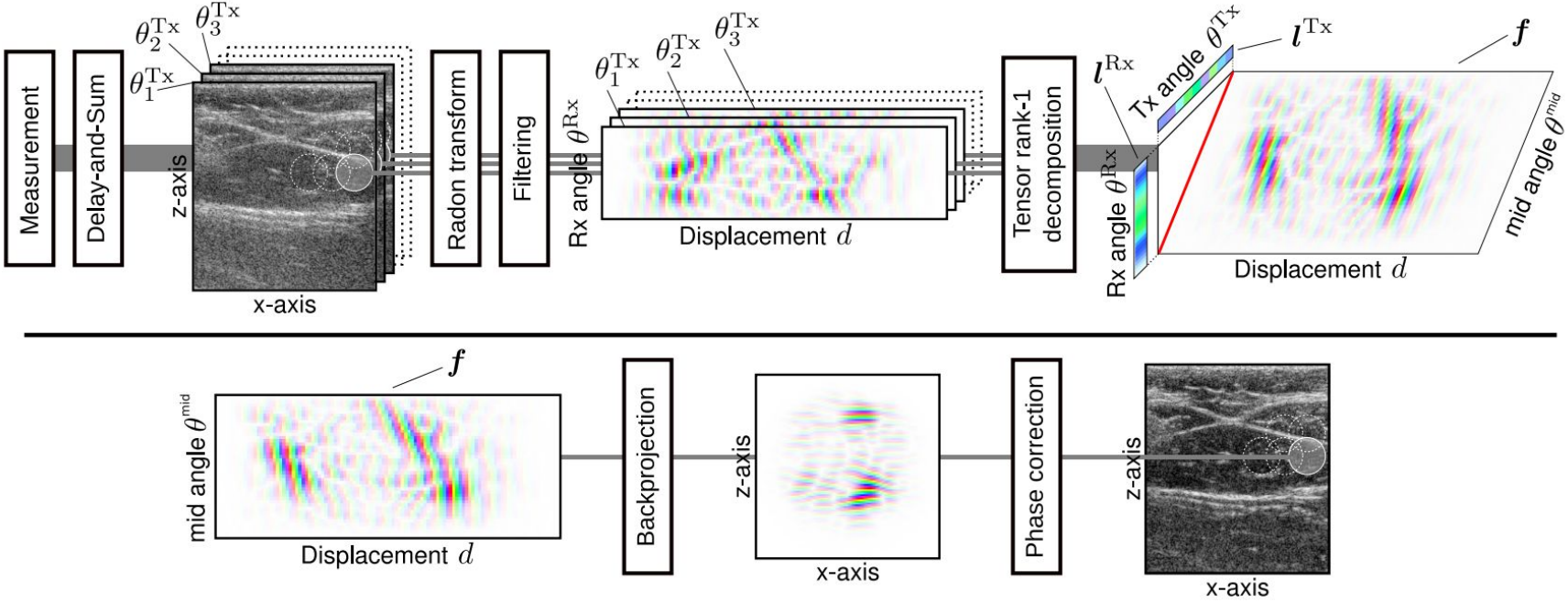


Figure. Pipeline de la méthode appliquée au Beamforming Adaptatif

Acquisition des images par Ultrasons Ultra-rapides

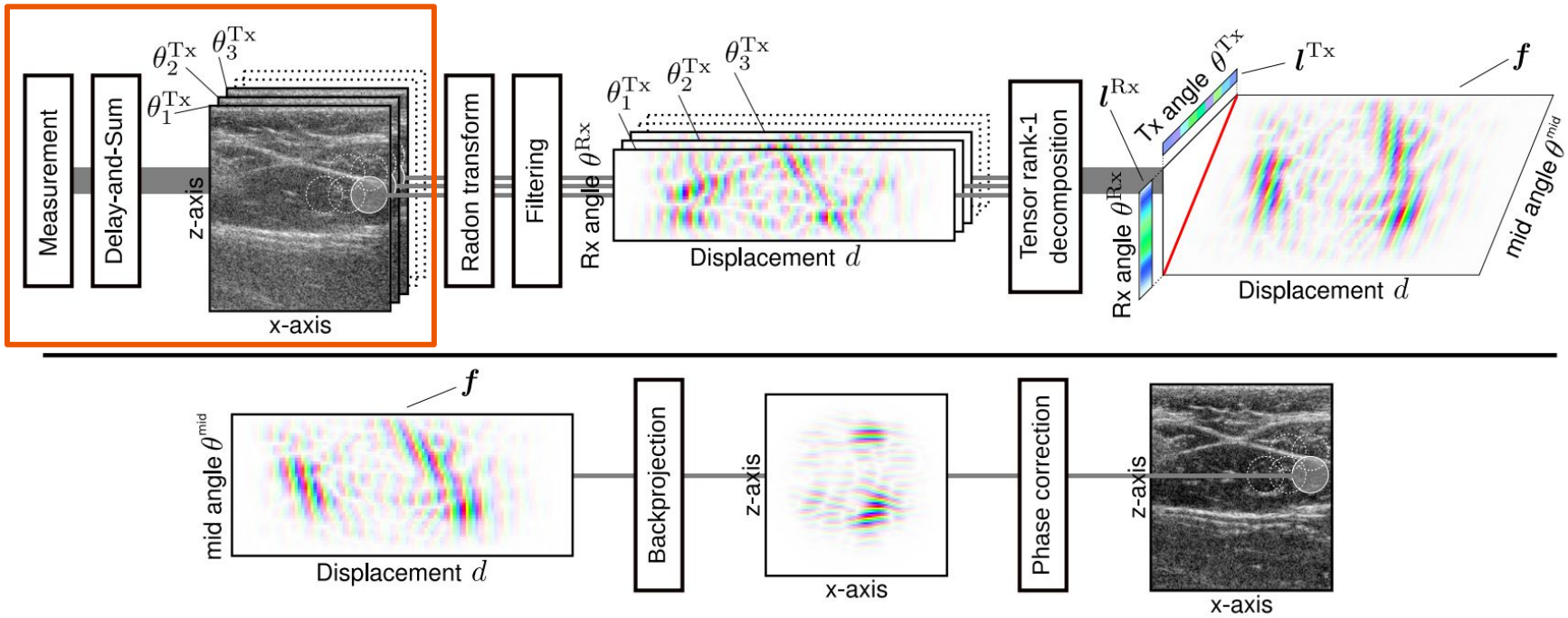


Figure. Pipeline de la méthode appliquée au Beamforming Adaptatif

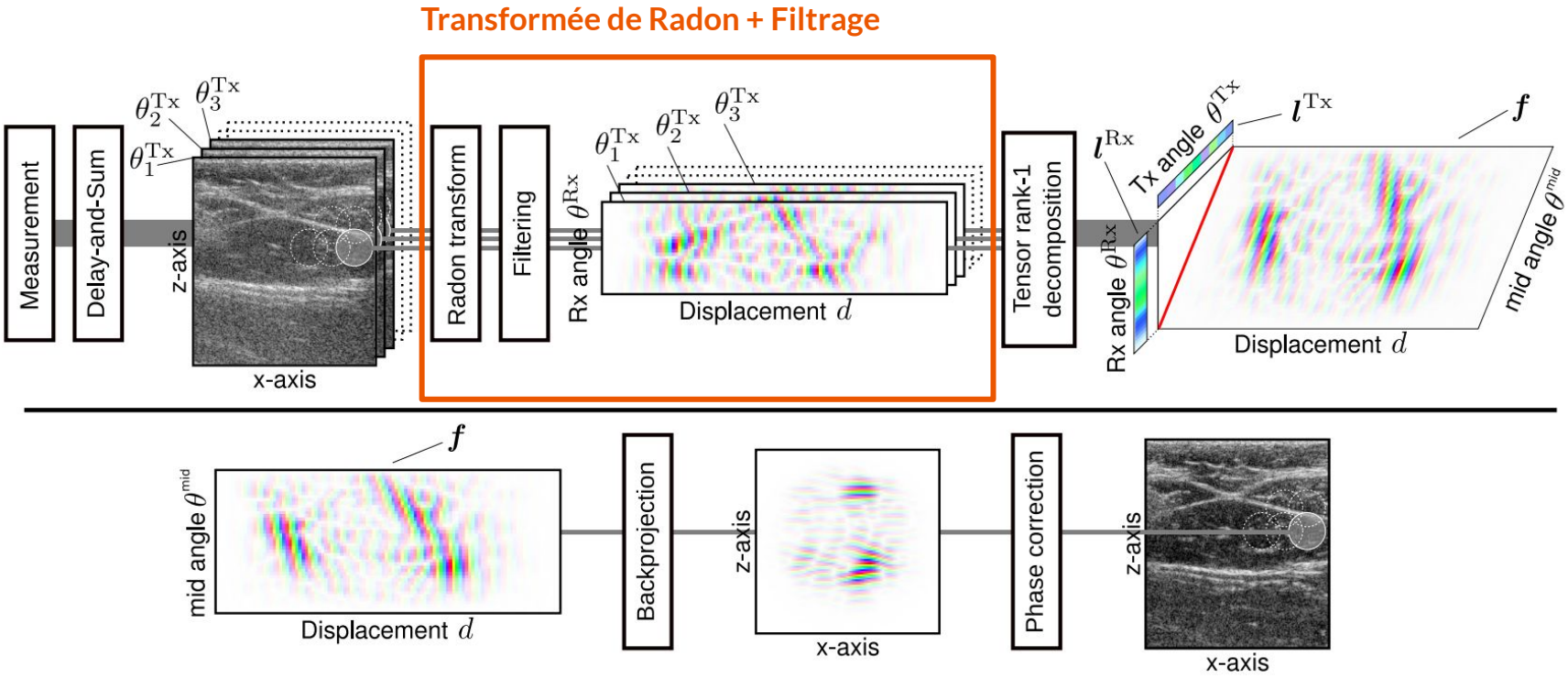


Figure. Pipeline de la méthode appliquée au Beamforming Adaptatif

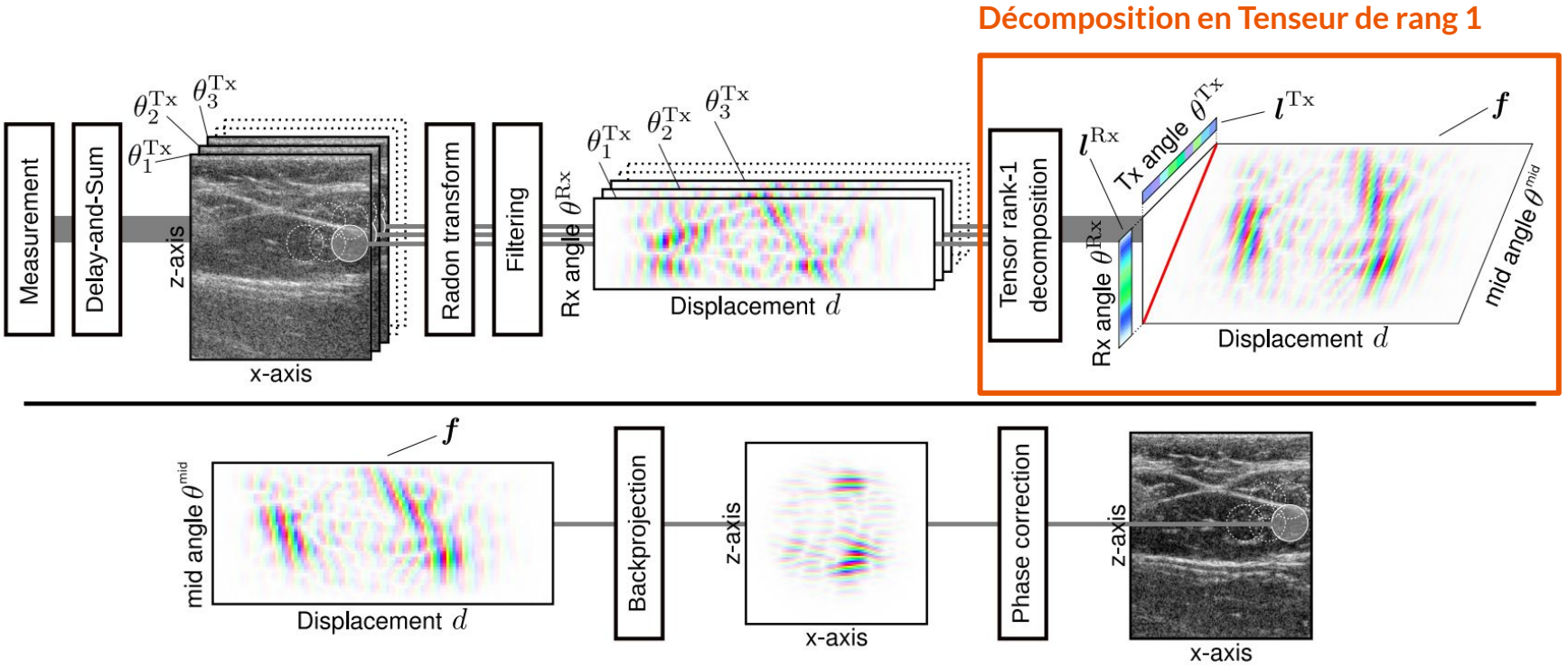
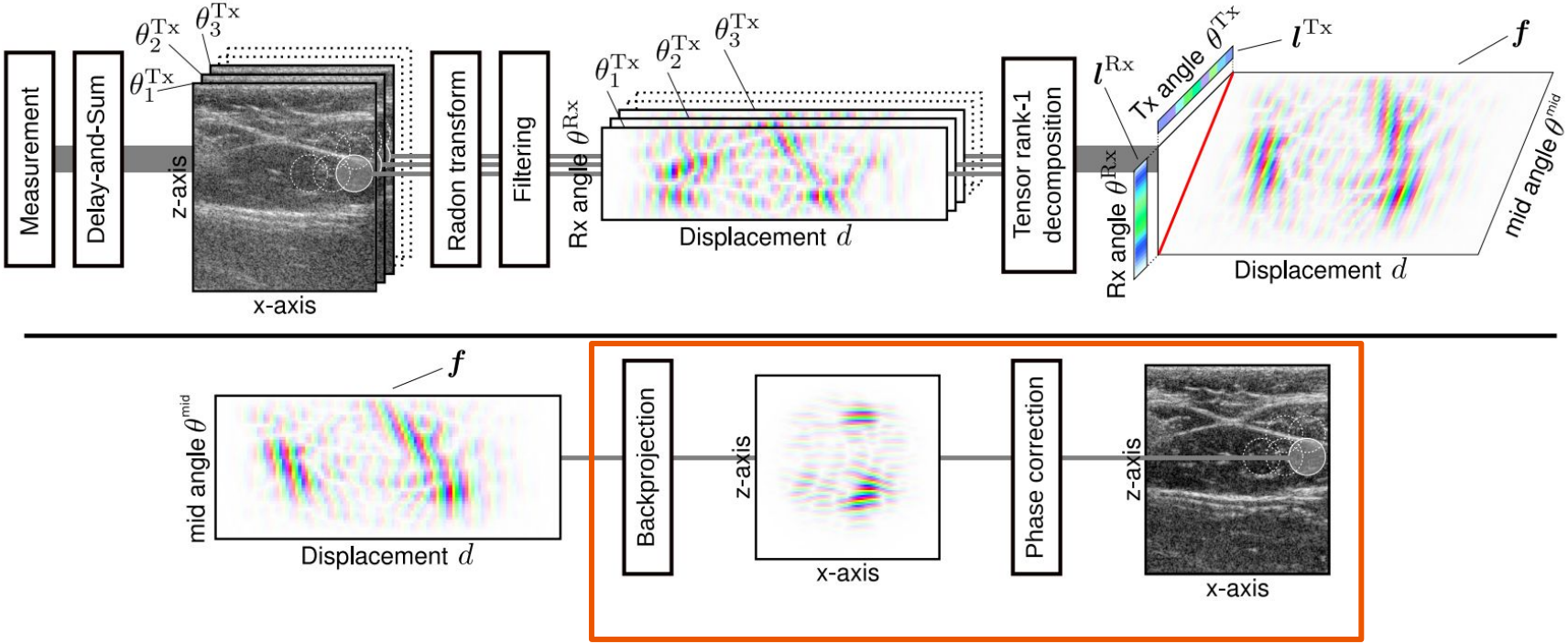


Figure. Pipeline de la méthode appliquée au Beamforming Adaptatif

Figure. Pipeline de la méthode appliquée au Beamforming Adaptatif



Reconstruction de l'image acquise à l'aide des données du Tenseur de rang 1

Initialement un fichier unique contenant l'intégralité du script :

- compute_corrected_image.py **:** contient l'entièreté du script + fonctions utilitaire

Code source modifié dans src :

• src/read_data.py : Visualise les données brutes à partir des matrices complexes.

• *src/compute_corrected_image.py :* Contient les implémentations du papier fourni par les auteurs.

• src/run_script.py : Point d'entrée local. Gère le support GPU/CPU (dépend de CUDA).

• src/run_notebook.ipynb : Point d'entrée Google Colab (GPU). Gère le support GPU/CPU (dépend de CUDA).

Fichier ajouté
Fichier modifié

```python
def main(folder_in, folder_out, is_gpu):
  """fonction principale de run_script.py"""

  # === Paramètres de preprocessing des données, de l'apodisation, de la transformée de Radon fenétrée,
  # de la decomposition tensorielle, du beamforming et du postprocessing === #
  # ...

  # chargement des données
    angles_tx, data, t_coord, c0, f0, x_sensor_list, angles_tx_raw = load_preprocess_data(folder_in,
                                angle_downsample_start, angle_downsample_factor, attenuation_tgc)

  # Compilations des fonctions CPU ou GPU
    comp_gpu(...) if is_gpu else comp_cpu(...)

  # on utilise soit les fonctions CPU soit les fonctions GPU

  # Beamforming
  beamformer = get_beamformer_npw_linear_transducer_Tukey_phase_screen_cpu(
            angles_tx, x_sensor_list, t_coord, 4, x_coord_beam, z_coord_beam, c0,
            apod_tukey_angle_max=max_angle, apod_tukey_cosine_frac=tukey_angle,
            return_one_per_angle=True)
  data_beamform = beamformer(data)

  # Storage for final patches
  patch_final_all = np.zeros((x_size_patch, z_size_patch, z_coord_small_out.shape[0],
    z_coord_small_out.shape[0]), dtype=np.complex64)
  n_iter_batch_x = int(np.ceil(x_size_patch / x_size_batch))

  start_batch_id = 0
```

Figure. Simplification de la fonction principale du script

```
for batch_id in range(n_iter_batch_x):
  x_size_batch_local = min(x_size_patch - start_batch_id, x_size_batch)

  # Select Patches
  patch_out = func_select_patch(data_beamform, window_val, x_size_batch_local, start_batch_id)

  # Radon Transform
  win_rad_filt = func_rx_radon(patch_out, mat_filt)

  # Decomposition
  f_real, f_imag = decomposition_cpu(win_rad_filt)

  # Backprojection
  patch_reconstructed = patch_backprojection(f_real, f_imag, window_val_out)

  # Store
  patch_final_all[start_batch_id:start_batch_id+x_size_batch_local] = patch_reconstructed
  start_batch_id += x_size_batch

 # === Reconstruction loop ===
shift_map = np.ones((x_size_patch, z_size_patch), dtype=np.complex64)

for iter_id in range(out_n_iter):
  img_temp = reconstruct_img_cpu(patch_final_all, shift_map)
  update_shift_cpu(patch_final_all, img_temp, shift_map)

  img_final_raw = reconstruct_img_cpu(patch_final_all, shift_map)
  norm_map = get_norm_cpu(window_val_out)
  img_out = norm_img_cpu(img_final_raw, norm_map)
```

Figure. Simplification de la fonction principale du script (suite)

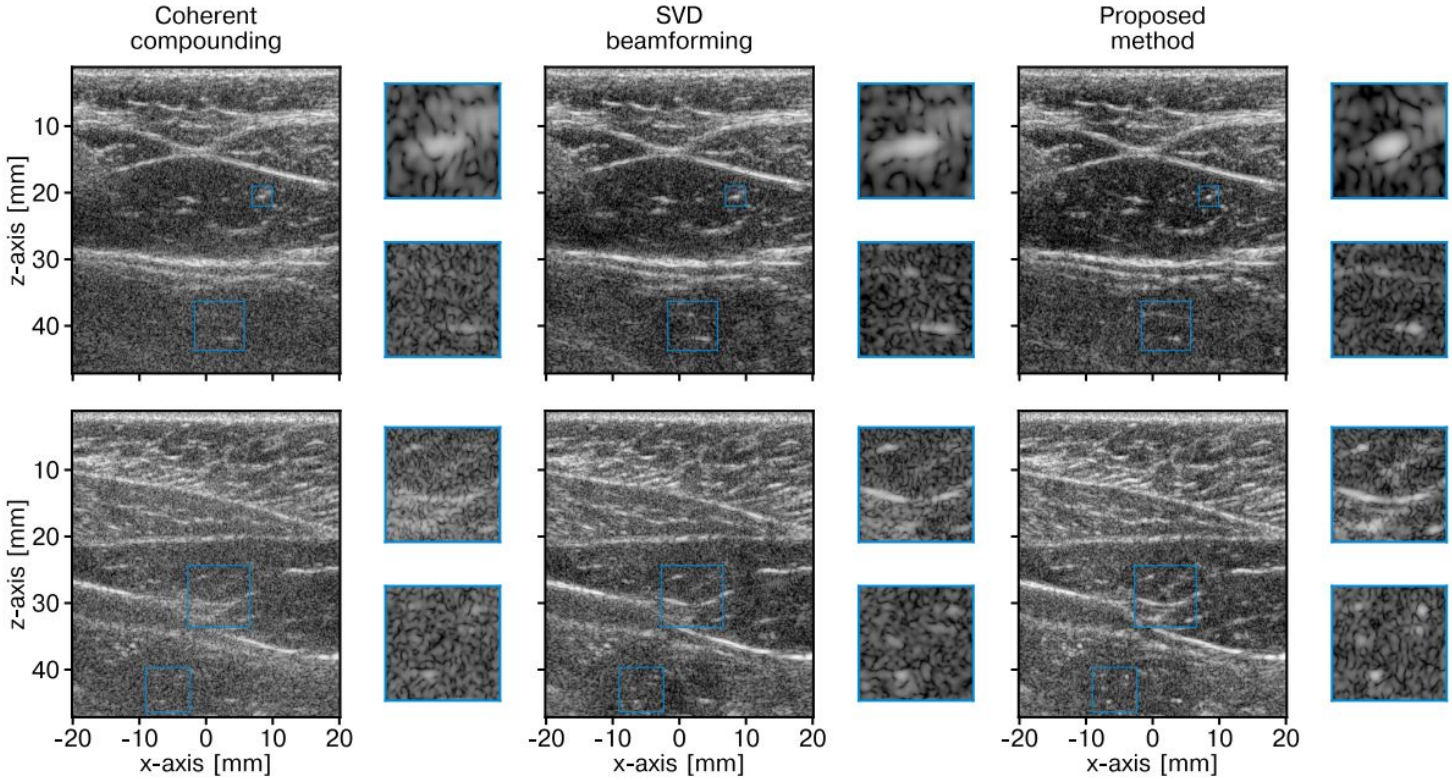


Figure. Résultats obtenus par les auteurs de l'article sur deux données US différentes. En particulier, la seconde correspond à la donnée que nous avons reproduit

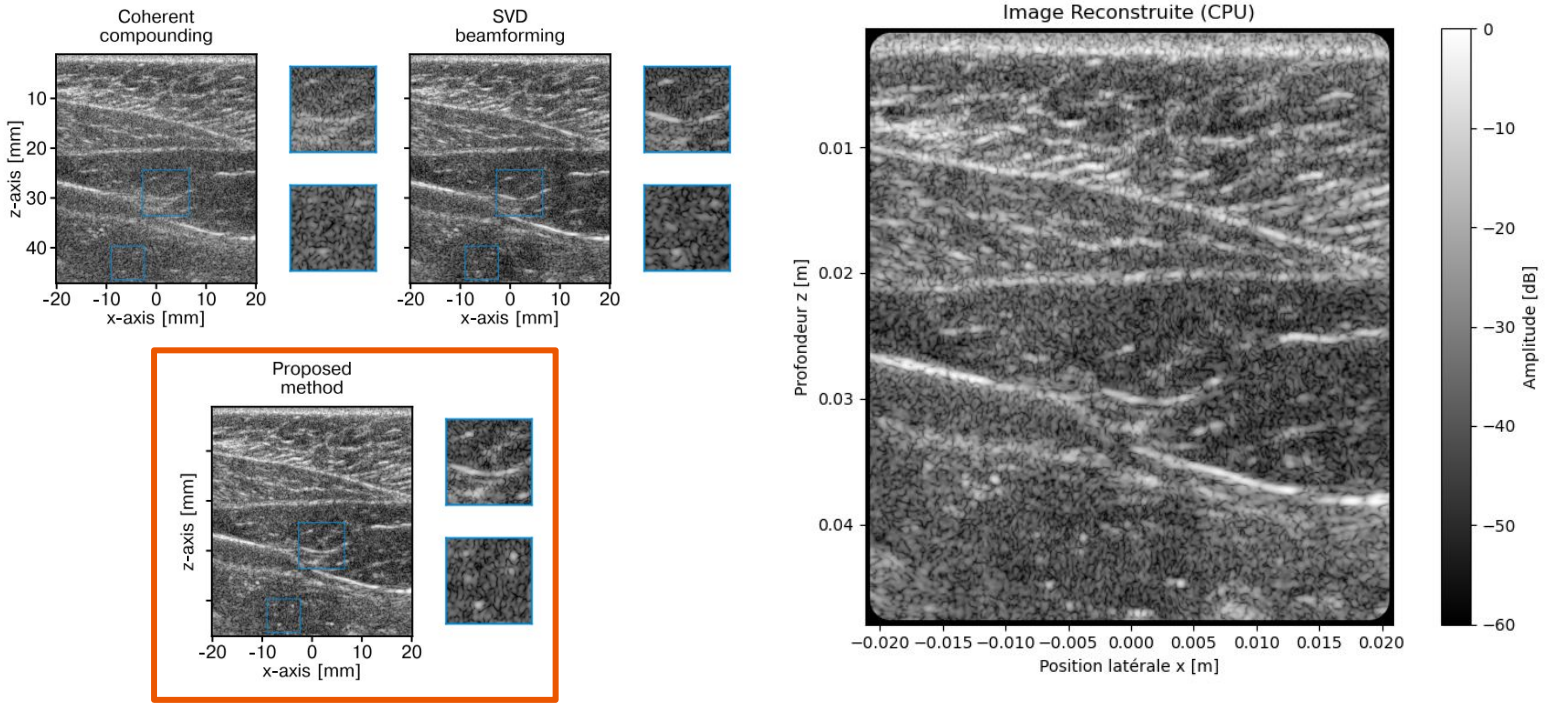


Figure. Comparaison entre nos résultats et ceux obtenus par les auteurs du papier. En particulier, le résultat **encadré** est le même que celui obtenu lors de notre test (*à droite*)

Conclusions et Observations :

- Validation des résultats par **analyse** du code et par **reproduction**
- Méthode donnant de bons résultats
- Supposition forte sur la **structure des signaux** réfléchis ainsi qu'une supposition que Tx/Rx sont décorrélés
- Impossibilité d'estimer la position absolue aux capteurs

Présentation basée sur l'article de recherche :

Windowed Radon Transform and Tensor Rank-1 Decomposition for Adaptive Beamforming in Ultrafast Ultrasound
S. Beuret and JP. Thiran, publié en Janvier 2024 au *Transaction on Medical Imaging Vol. 43*