Dense Visual Odometry

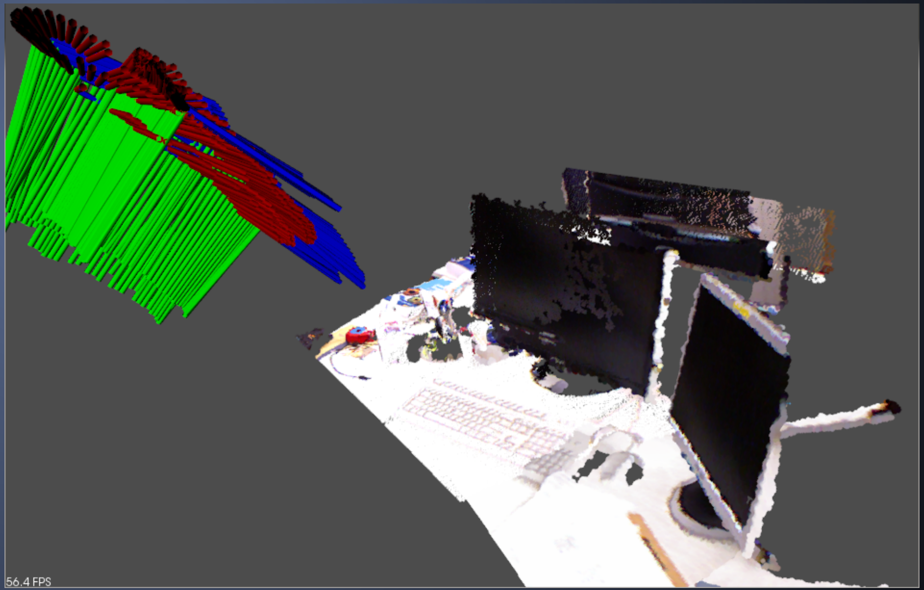
by
Poulami Chakrabarti
Shweta Shitole

Supervisor: Robert Maier

Contents

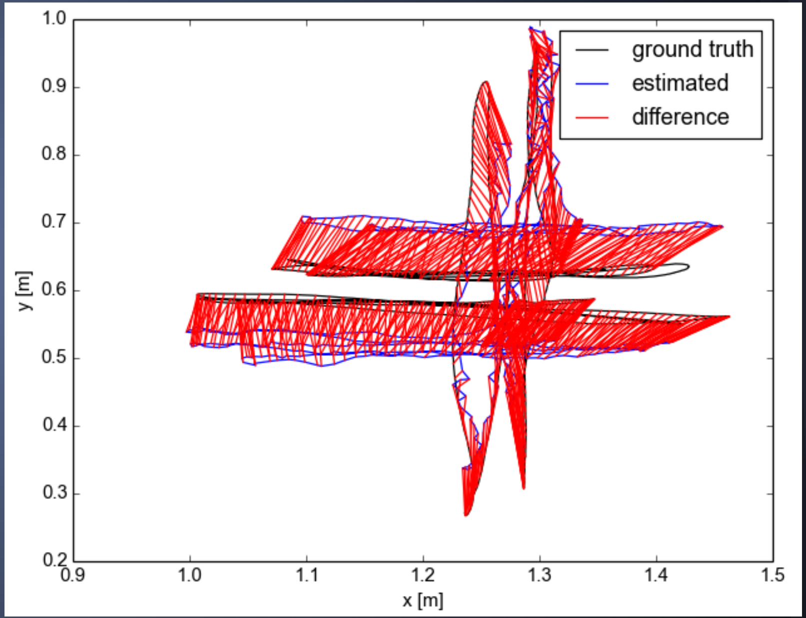
Dense Visual Odometry

- Determine the position and orientation of a robot by analyzing the associated camera images.
- No external reference is available.
- Classical approaches:
  - Visual features
  - Patch based approaches



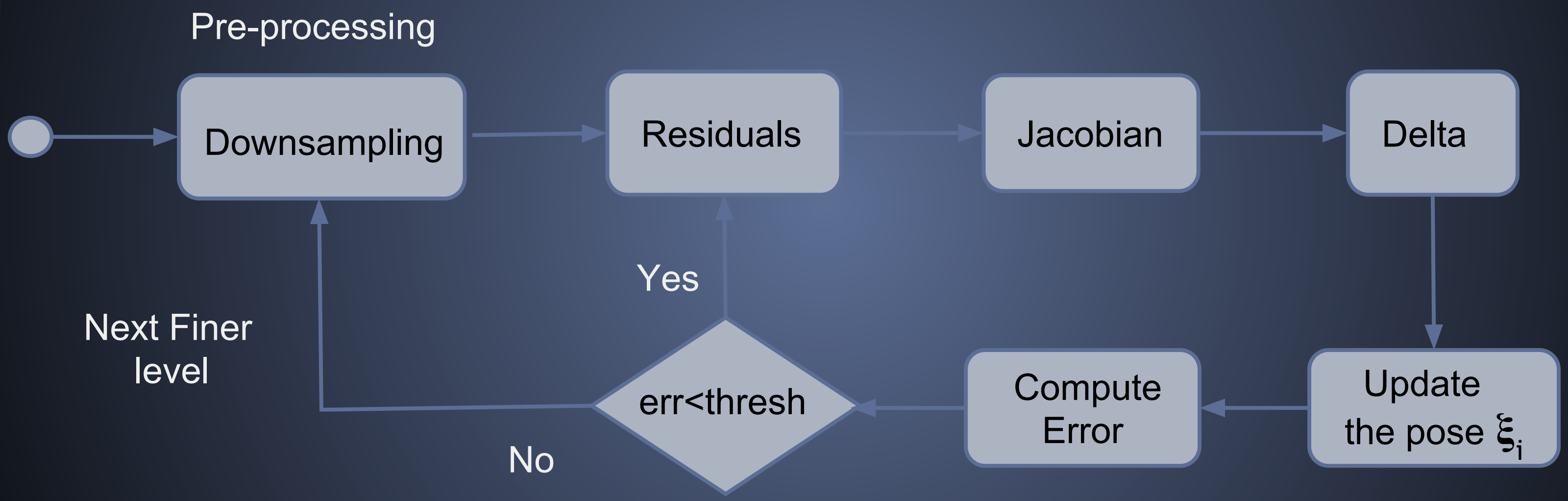
Project Outline

- ➔ Reference: CPU implementation[1]
- ➔ First Phase: Estimate the camera pose
- ➔ Second phase: benchmark datasets.
- ➔ Third phase: Evaluation
- ➔ Fourth phase: 3D visualization of the point cloud and trajectory.



[1] - Robust Odometry Estimation for RGB-D Cameras by Christian Kerl, J¨urgen Sturm, and Daniel Cremers

Flowchart



Pre-processing
Downsampling
Residuals
Jacobian
Delta
Update the pose ξ_i
Compute Error
err<thresh
Yes
No
Next Finer level

Preprocessing

➔ Uses the input image I and camera matrix K.

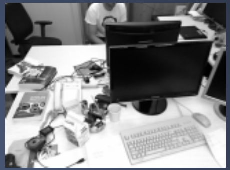
$$I_d(x, y) := 0.25 \sum_{x',y' \in O(x,y)} I(x', y')$$

where $O(x, y) = \{(2x, 2y), (2x + 1, 2y), (2x, 2y + 1), (2x + 1, 2y + 1)\}$.

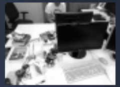
Downsampling

640 x 480

320 x 240

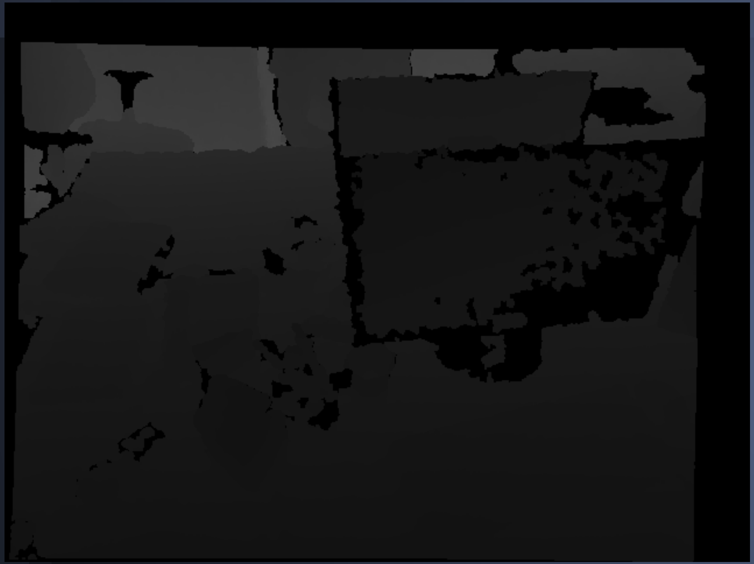


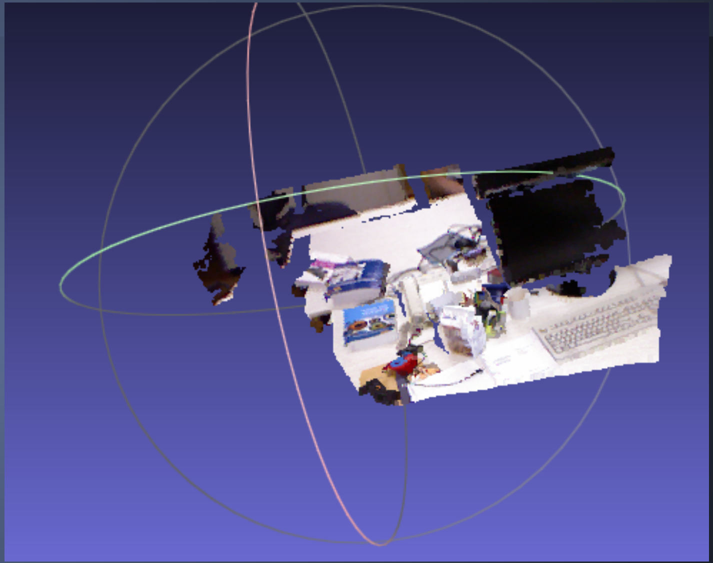
160 x 120



80 x 60

Point cloud





$$\omega(\mathbf{p}_i, d, \xi) := \pi(K(R_\xi K^{-1} \begin{pmatrix} dp_{i,x} \\ dp_{i,y} \\ d \end{pmatrix} + \mathbf{t}_\xi))$$

Residual Image

$$E(\xi) = \sum_{\mathbf{p}_i \in \Omega_{\text{ref}}} \underbrace{\left(I_{\text{ref}}(\mathbf{p}_i) - I(\omega(\mathbf{p}_i, D_{\text{ref}}(\mathbf{p}_i), \xi))\right)^2}_{=: r_i^2(\xi)}$$

Where, p_i :

$$\begin{aligned} \mathbf{p} &= \pi^{-1}(\mathbf{x}, Z_1(\mathbf{x})) \\ &= Z_1(\mathbf{x}) \left(\frac{u - c_x}{f_x}, \frac{v - c_y}{f_y}, 1 \right)^\top \end{aligned}$$

and

$$\pi(T(g, \mathbf{p})) = \left(\frac{f_x x}{z} + c_x, \frac{f_y y}{z} + c_y \right)^\top$$

Where, fx,fy,cx and cy are camera intrinsic parameters.

Texture Memory

Interpolation required for

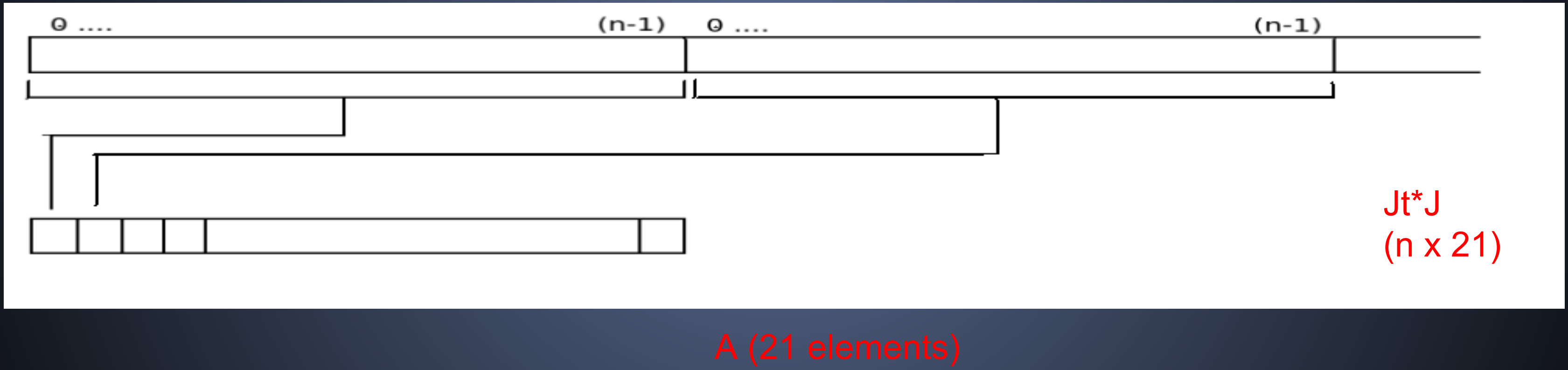
- residual image calculation
- Jacobian calculation

$$\left.\frac{\partial r_i(\epsilon \circ \xi^{(k)})}{\partial \epsilon}\right|_{\epsilon=0} = \frac{1}{z'}\begin{pmatrix}\nabla I_x f_x & \nabla I_y f_y\end{pmatrix}\begin{pmatrix}1 & 0 & -\frac{x'}{z'} & -\frac{x'y'}{z'} & (z' + \frac{x'^2}{z'}) & -y' \\ 0 & 1 & -\frac{y'}{z'} & -(z' + \frac{y'^2}{z'}) & \frac{x'y'}{z'} & x'\end{pmatrix}$$

= 1x6 row of $J_{\mathbf{r}}$

Parallel Reduction

- ➔ Shared memory is used.
- ➔ For J^t*J (Nx21) and J^t*residuals (Nx6).



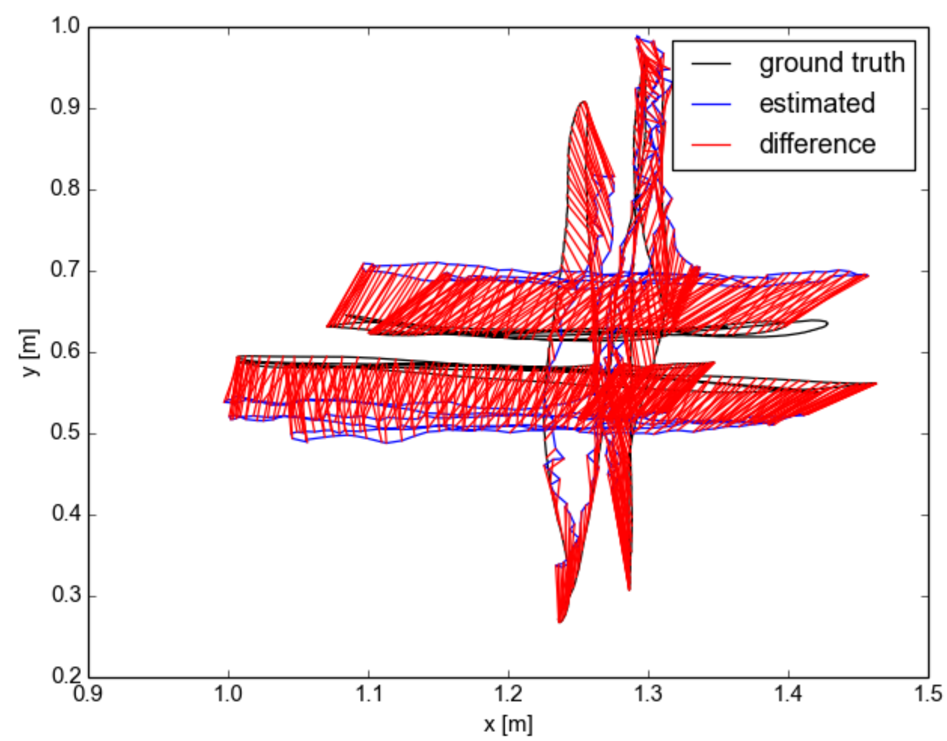
Gauss Newton Update

- ➔ Compute delta, $$\delta_\xi = -(J_{\mathbf{r}}^T J_{\mathbf{r}})^{-1} J_{\mathbf{r}}^T \mathbf{r}_0$$
- ➔ Update ξ , $\xi^{(i+1)}$ = delta * ξ^{i} , iterate till convergence .
- ➔ If the delta value exceeds threshold (>0.995) then go to the next finer level.

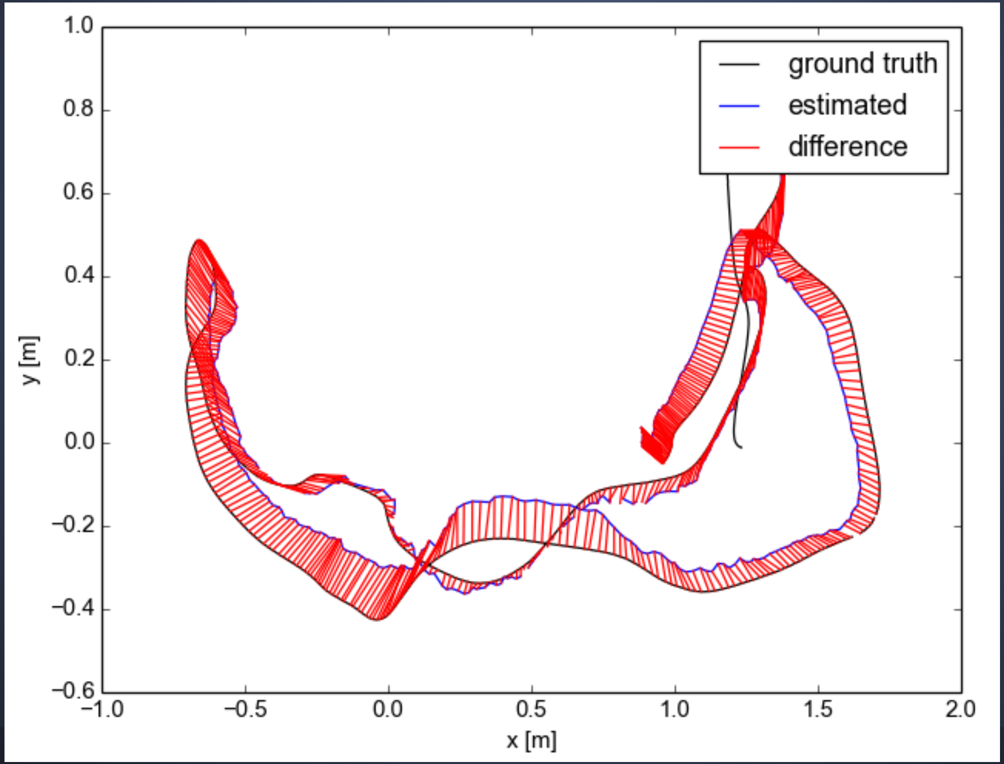
Problems Faced

- ➔ Memory Leaks
- ➔ Limited arguments to a kernel call (256 bytes)
- ➔ Boundary checks

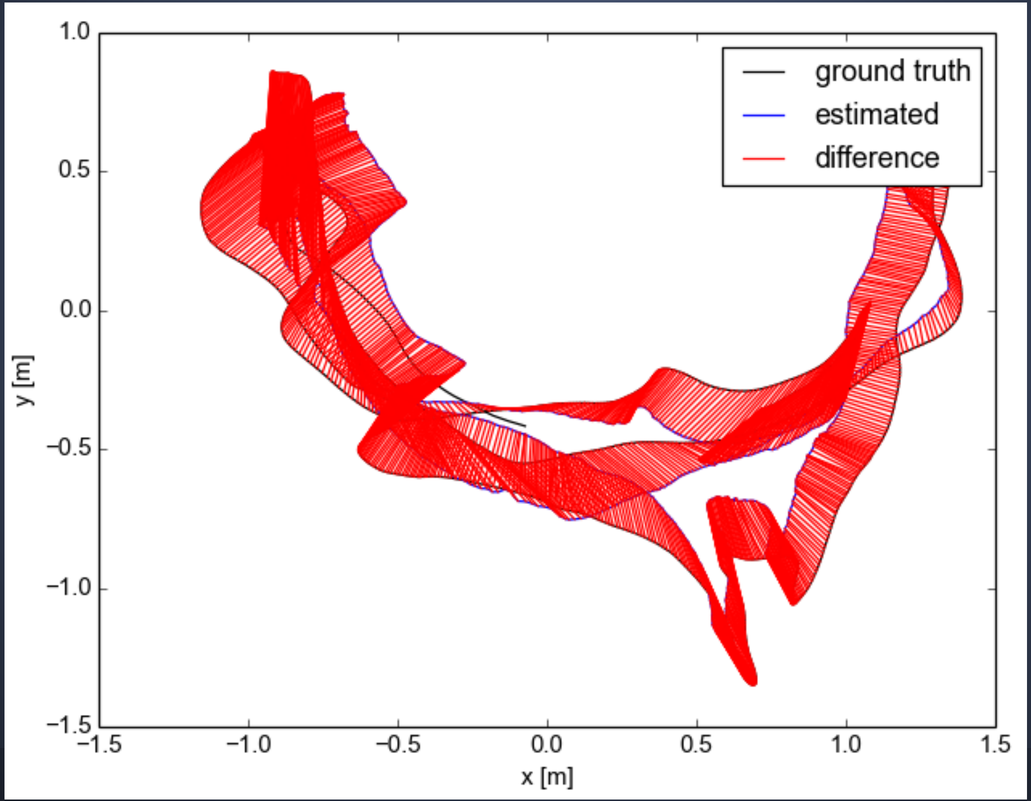
Results: fr1/xyz



Results: fr1/desk



Results: fr1/room



Further Work

- ➔ Code Optimization
- ➔ Cublas
- ➔ Real time implementation

Demo . . .

THANK YOU

QUESTIONS?