Incremental Registration of RGB-D Images

Ivan Dryanovski, Carlos Jaramillo and Jizhong Xiao

Contents

- Introduction and problem statement
- Edge detection
  - Detection and filtering
  - Edge descriptors
- Registration
  - High-frequency loop
  - Low-frequency loop
- Results
- Current work

Introduction: The Problem

Problem description:

- Estimate the 6-DoF pose of a RGB-D camera in freehand motion
- High-frequency update described

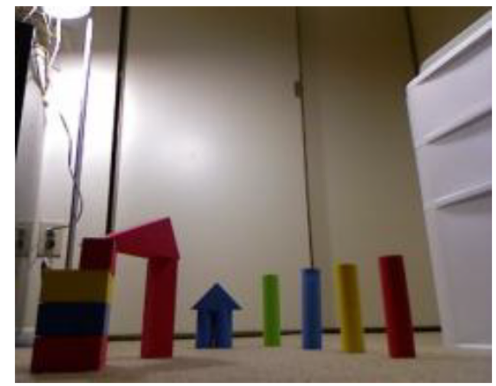
Data input:

- Scans from an RGB-D camera (such as Kinect).
- We assume RGB and Depth images are already registered

Applications:

A fast, robust pose estimation is useful for:

- Control
- 3D Mapping & SLAM



via http://www.roborealm.com

Introduction: The Approach

Overview of the approach:

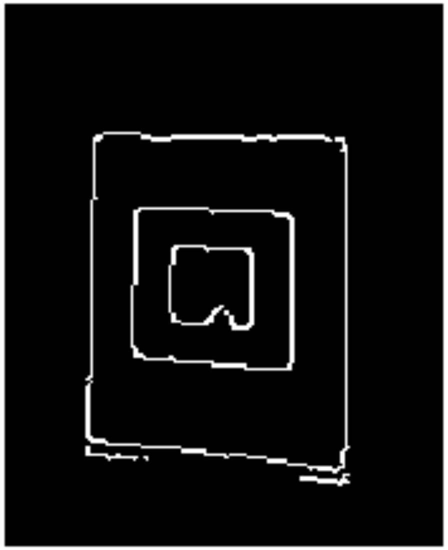
- Hi-freq. loop – operates on sparse data
  - Detect edges
  - Filter edges
  - Classify edges
  - Perform Edge-ICP
- Lo-freq. loop – operates on dense data
  - Use output of hi-freq. loop as estimation for motion
  - Perform point-to-plane ICP

Edge detection

1) Convert RGB image to gray-scale

2) Perform Gaussian blur filtering to remove noise

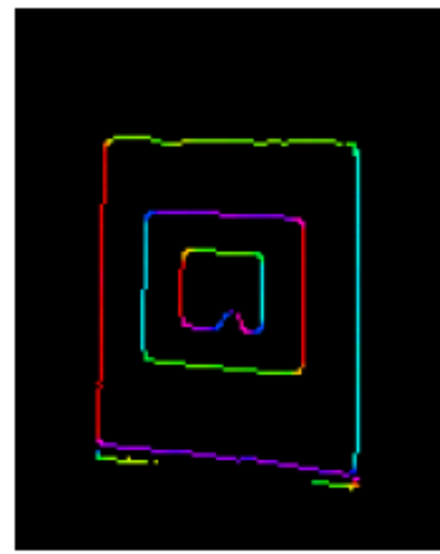
3) Perform Canny edge detection to locate edges



Edge classification

Each pixel belonging to an edge is classified by the edge orientation

Orientation is computed using the gradients along the x- and y-dimensions.



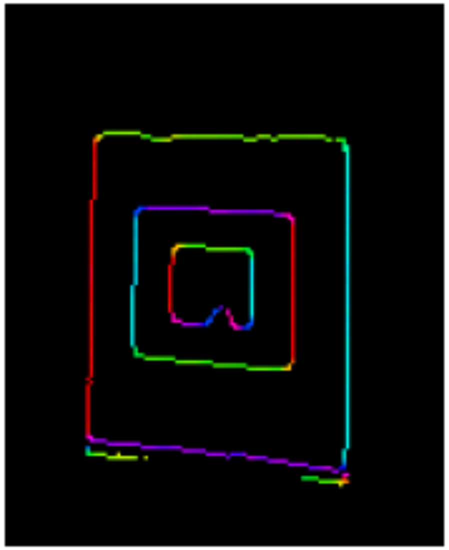
Edge classification

Each pixel belonging to an edge is classified by the edge orientation

Orientation is computed using the gradients along the x- and y-dimensions.

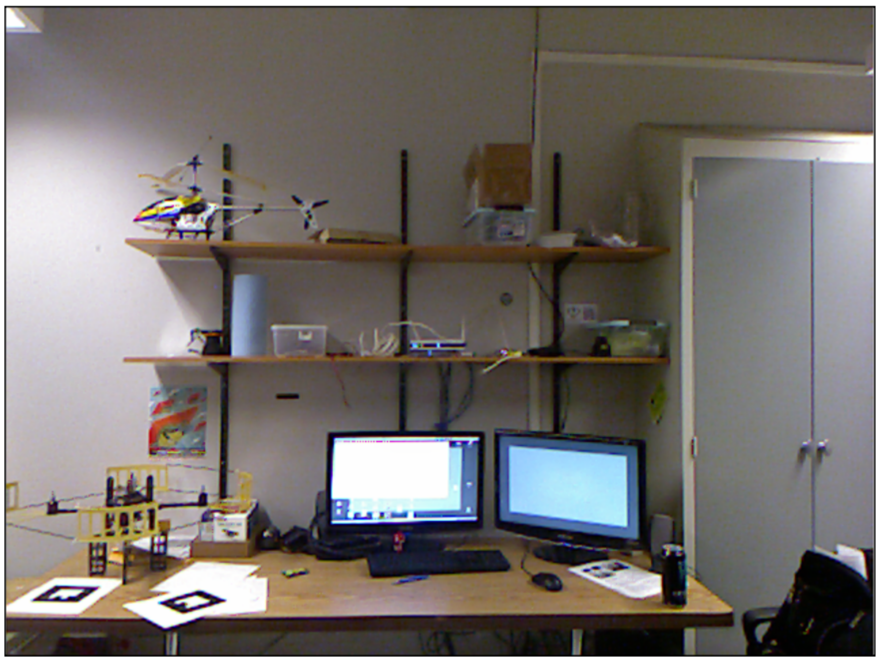
$$\theta = \mathrm{atan}(\frac{dy}{dx})$$

Where *dy* and *dx* are computed using a Sobel operator



Detection + classification

Example output for a typical image:

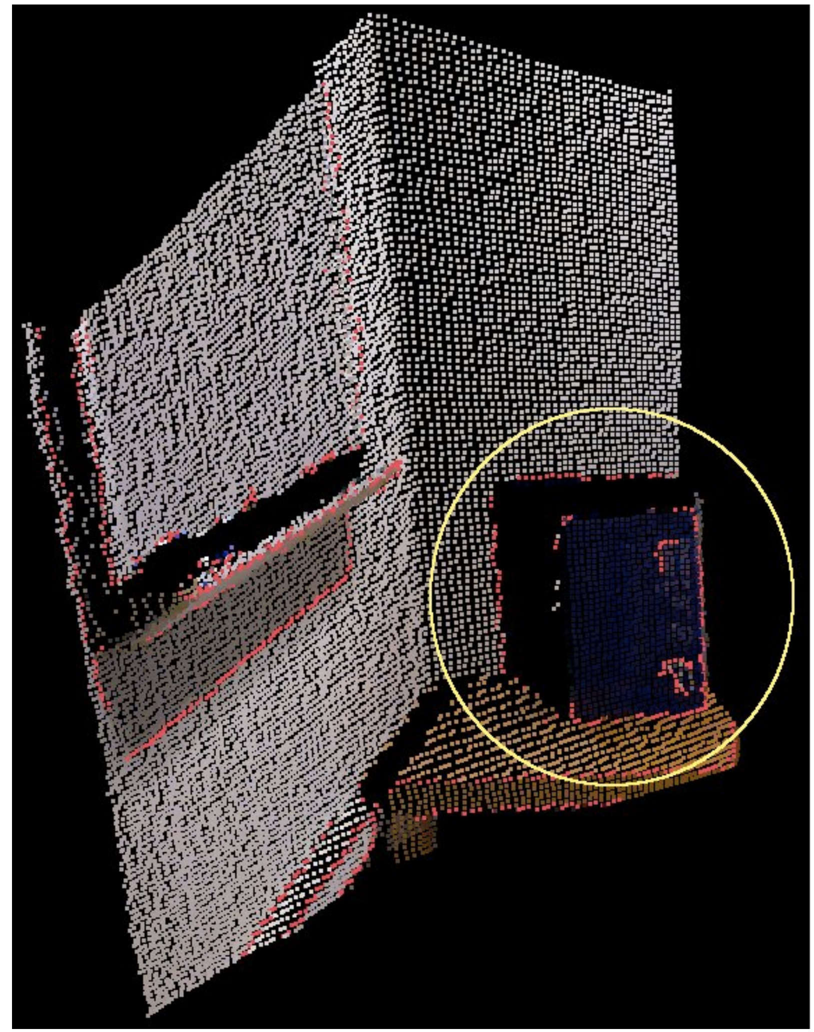


Edge filtering – why is it needed?

We create sparse point cloud from all the pixels belonging to edges (see image - pink pixels)

Problem: edge sometimes appears on the object background instead of object foreground (see image – area in circle)

However, we want edges which are *pose-invariant*.



Edge filtering algorithm

Problem:

Edge sometimes appears on the object background instead of object foreground

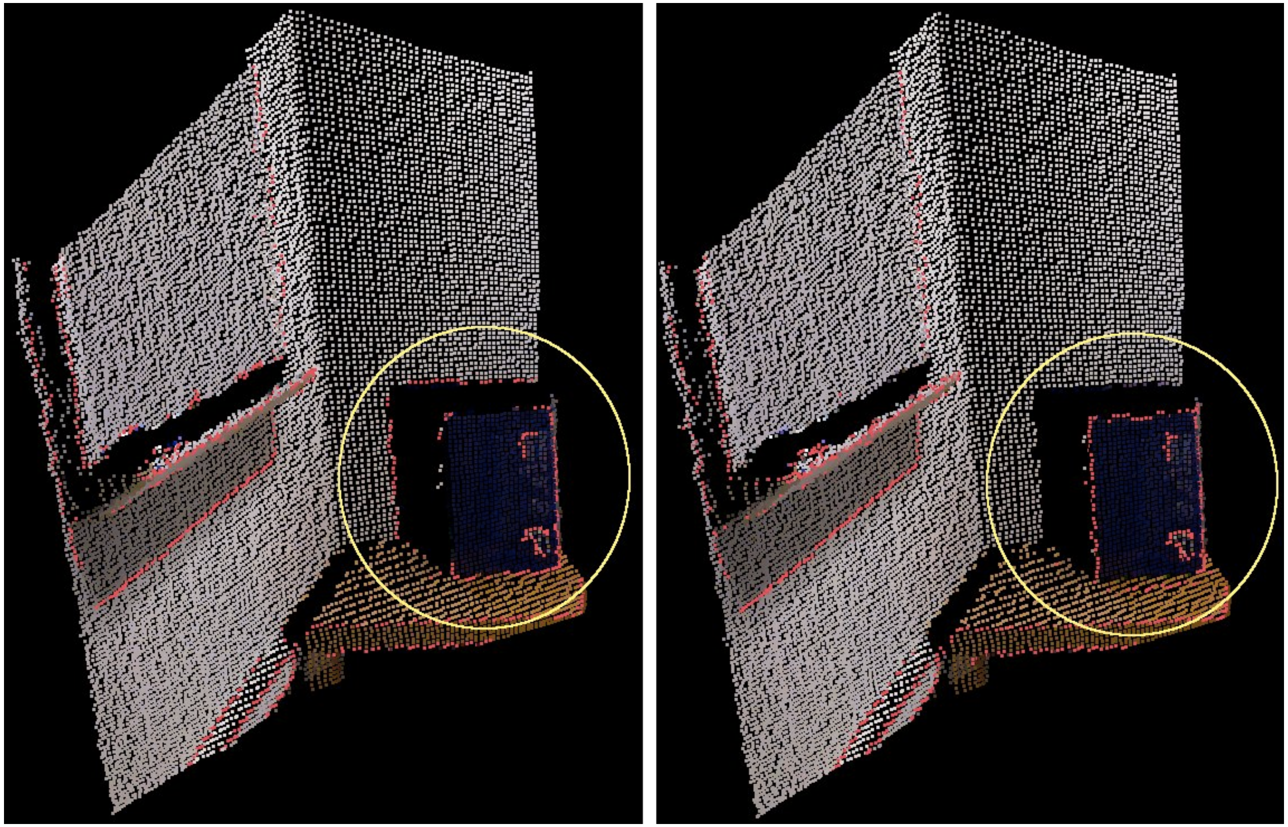
Solution:

Perform a local search around each edge pixel (window size of 3x3 or 5x5) in the Depth image

If there exists a significant jump from high depth value to low depth value, use the pixel with the lower depth value (closer to the camera).

Detection + classification

Example output of depth filtering:



Edges on book correctly determined, other edges remain the same.

Registration: high frequency loop

Edge-ICP algorithm:

Register sparse point clouds of edge points using ICP

Nearest neighbor correspondences are computed using Euclidean distance in 3D.

For each nearest neighbor found, we check if the θ values are the similar (within 30 degrees)

If not, we look for the next nearest neighbor.

This helps prune out false correspondences between edges

Registration: low frequency loop

The low-frequency loop runs parallel to the high-frequency loop in a separate thread

We use Generalized ICP (G-ICPP) to register the dense 3D data

Normally, GICP is slower. We use the estimated transform from the high-freq. loop as input to GICP

Result: low-freq. loop refines results from high-freq. loop.

Results:

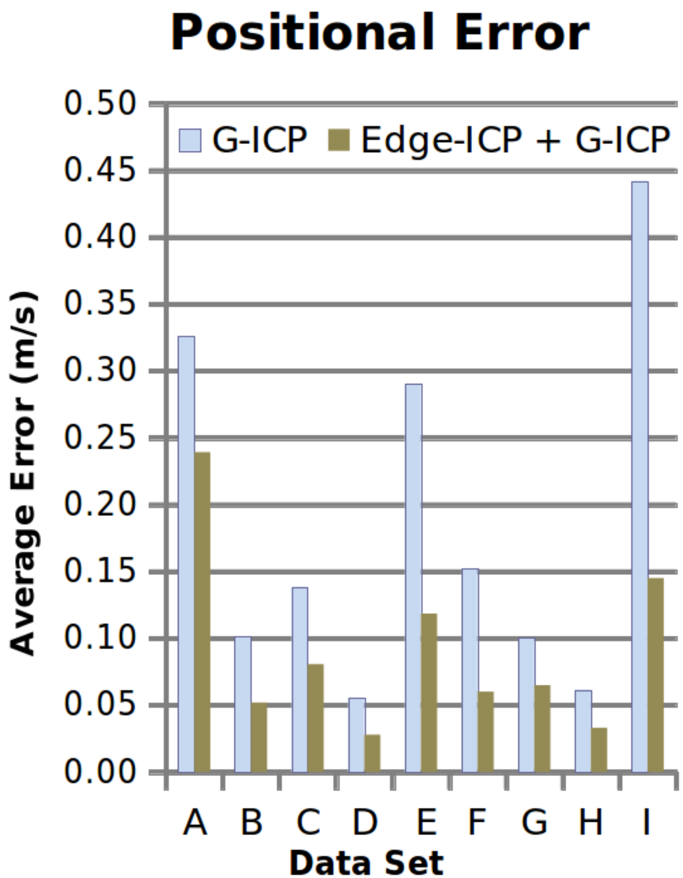
We use an RGB-D datasets with ground truth from a VICON motion-capture system.

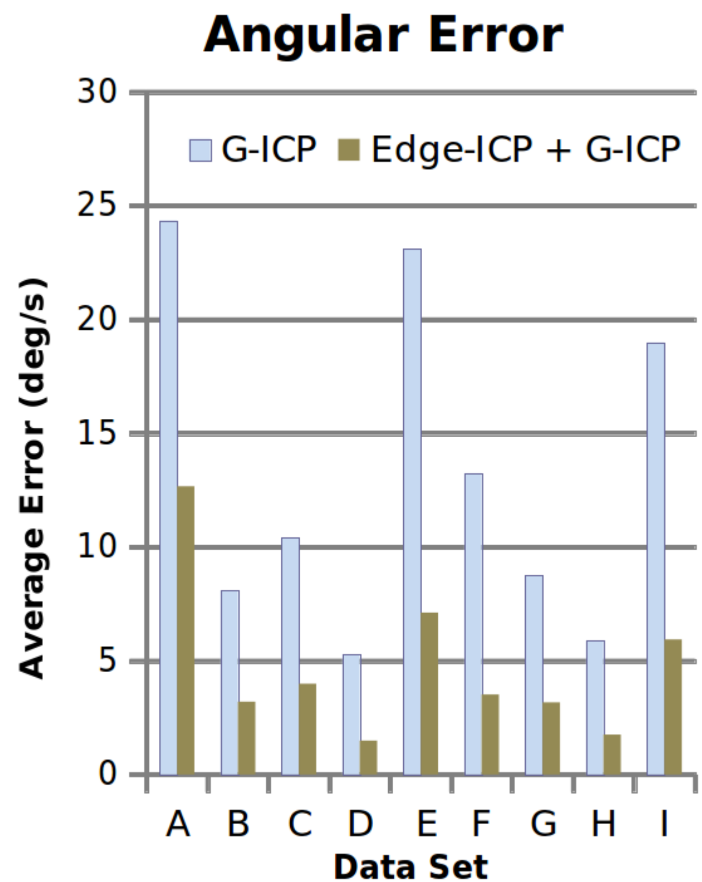
Compare error from dual-loop approach(Edge-ICP + GICP) vs. single-loop (GICP) registration.

Measure average angular error and translational error per each frame.

Results:

How does the dual-loop approach compare vs. regular GICP?





Current work:

1) We are considering variety of features for the high-frequency loop:

- Canny edges
- ORB
- SURF

Evaluation of which offers better tradeoff between speed and accuracy

2) Incorporating a full SLAM system which can deal with loop closure

3) Open-source release, targeted for robotics applications.

Thank you!