Graph-based visual SLAM and visual odometry using an RGB-D camera

Liang Xu

This is a project report for CMSC-591 Robotic Vision. In this report, I will show my visual odometry(VO) and simultaneous localization and mapping (SLAM) result using graph optimization based on G2O framework. The presented visual SLAM algorithm can be separated into three parts: feature extraction, data association, and SLAM back-end. I used SIFT for feature detection and the Binary Robust Independent Elementary Features (BRIEF) as feature descriptor, which together provides a fast and stable feature extraction.

1 Introduction

The Simultaneous Localization and Mapping (SLAM) problem deals with building a map of an unknown environment and localizing a robot in that environment. Typically, SLAM is described in probabilistic terms. This work presents a graph-based visual SLAM approach, using an RGB-D camera as the only sensor. RGB-D camera provides out-of-the-box color to depth registration, which may be used to incorporate visual features into a three-dimensional map.

The Swiss Ranger 4000 3D, time-of-flight industrial camera provides high-resolution 3D image data in real time. The device relies on an image sensor technology capable of capturing 3D data sets (known as depth maps) in real time and is built into a single, compact, solid-state housing suited to harsh environments.

A visual graph-based SLAM algorithm builds maps of the environment by storing notable locations, including robot position and 3D image features observed at that location, as nodes in a pose graph. The nodes are linked if 3D image features between the nodes match. Edges then describe transformations between nodes, which are

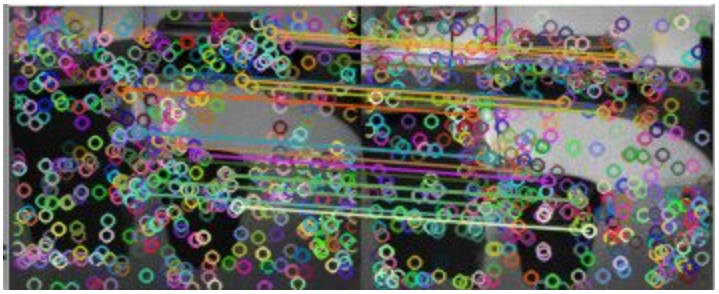
estimated based on the matching 3D image features. The relative representation of node-to-node transformations enables graph-based SLAM to cope with large measurement errors. Graph-based SLAM is further applicable for large-scale online mapping of three-dimensional environments.

2. Graph-based Visual SLAM

An overview of the graph-based visual SLAM framework is shown in the below figure. It contains the Feature Extraction, which uses the data from the RGB-D camera to extract features (observations) and transforms them into a 3D-space. These observations are used to determine visual odometry and, together with the odometry, based on the visual odometry information and locally stored feature information from the last measurement, when a new node should be added to the graph. If a new node is added to the graph, it searches for potential neighbor nodes and computes transformations to these potential neighbor nodes, integrating the new node into the existing graph and optimizing the graph using the general graph optimization framework g2o afterward.

2.1 Feature Extraction

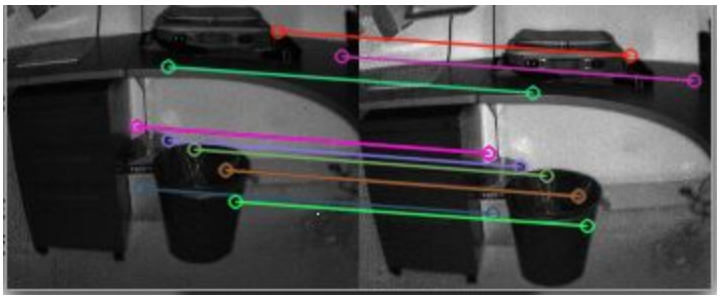
Visual SLAM highly relies on the quality of image features. Image features describe the environment and are used to compute transformations between different measurements. Requirements for good image features are the uniqueness of their descriptor as well as the invariance to contrast, scale, and rotation. In this project, SIFT feature and BRIEFdescriptor have been used.



Only take good features

2.2 Visual Odometry

3D to 3D pose estimation has been used based on the Aaron Paper[1]. For more detailed information please check the paper.



Inliers Feature points

TABLE I Pose Estimation Errors in Rotation Measurement

MV: (mean,stddev) TV: (roll, pitch, yaw)	Roll	Pitch	Yaw
(0, 3, 0)	mean: 0.04, stddev: 0.61	mean: -2.45, stddev: 1.93	mean: 0.62, stddev: 1.82
(0, 6, 0)	mean: -0.10, stddev: 0.37	mean: -5.80, stddev: 1.05	mean: 0.07, stddev: 0.78
(0, 9, 0)	mean: 0.04, stddev: 1.34	mean: -8.55, stddev: 2.72	mean: 0.12, stddev: 0.42
(0, 0, 3)	mean:-0.08, stddev 0.36	mean: -0.12, stddev: 0.18	mean: -2.94, stddev: 0.37
(0, 0, 6)	mean: 0.68, stddev: 0.68	mean: -0.39, stddev: 0.36	mean:-5.80 stddev:0.64
(0, 0, 9)	mean: 0.79, stddev: 0.89	mean: -0.15, stddev: 0.68	mean: -8.95, stddev: 0.77

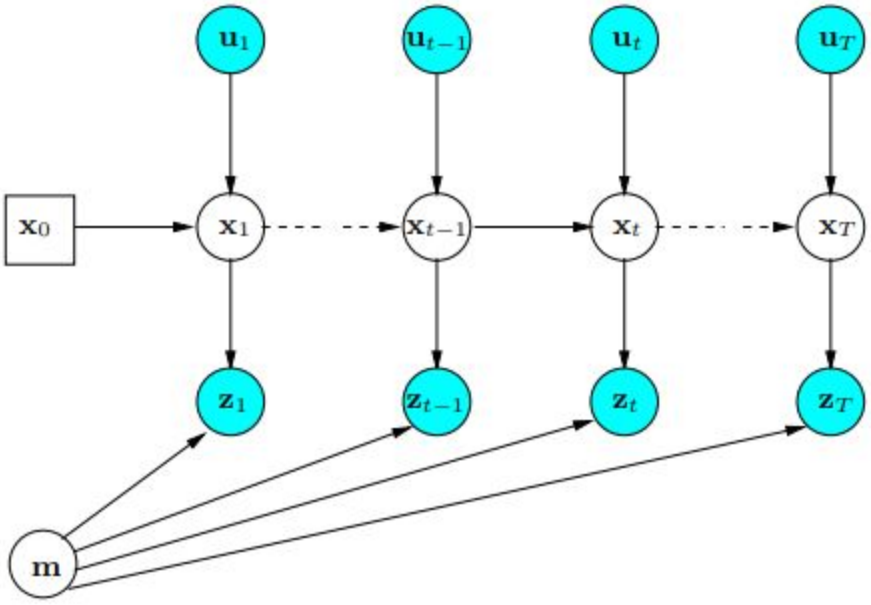
TABLE II Pose Estimation Errors in Translation Measurement

MV: (mean,stddev) TV: (X, Y, Z)	x (mm)	Y (mm)	Z(mm)
(0, 100, 0)	mean: 10.87, stddev: 2.45	mean: 112.68, stddev: 10.87	mean: 8.83, stddev: 1.48

(0, 200, 0)	mean: 6.34, stddev: 1.55	mean: 207.47, stddev: 8.57	mean: 9.44, stddev: 2.13
(0, 300, 0)	mean: 3.47, stddev: 2.32	mean: 310.12, stddev: 9.14	mean: 8.23, stddev: 2.08

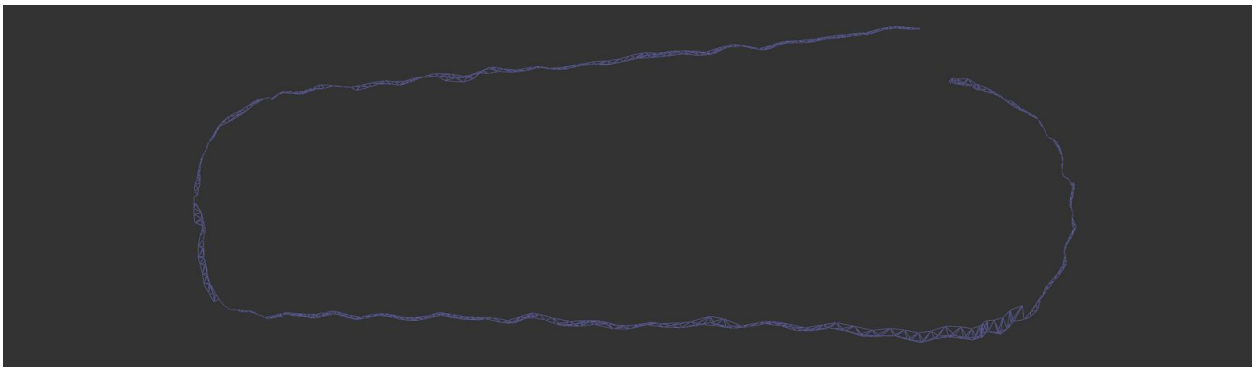
2.3 Graph SLAM

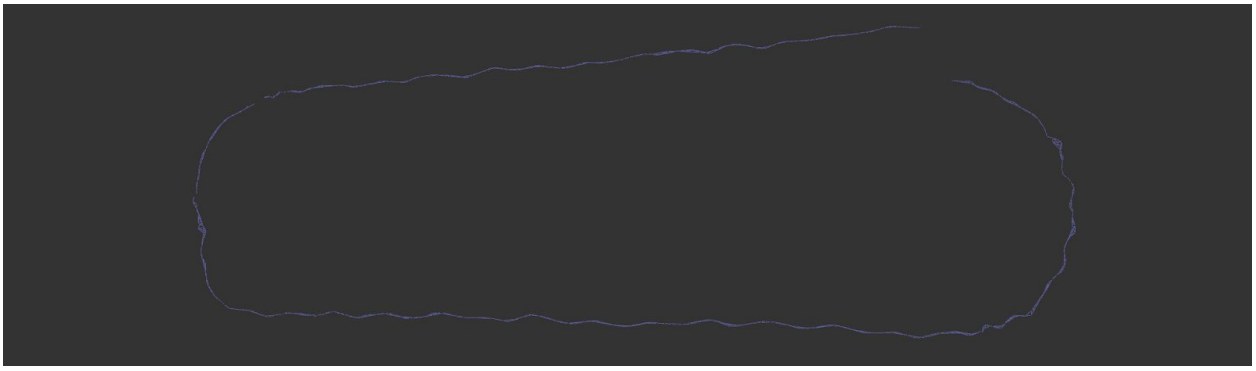
A graph-based SLAM approach constructs a simplified estimation problem by abstracting the raw sensor measurements. These raw measurements are replaced by the edges in the graph which can then be seen as “virtual measurements”. More in detail an edge between two nodes is labeled with a probability distribution over the relative locations of the two poses, conditioned to their mutual measurements.



SLAM as a Graph Problem

Final result:





After the optimized result:

iteration= 985	chi2= 76863572.797245	time= 0.0133458	cumTime= 11.2951	edges= 1573	schur= 0	lambda= 9966003.822841	levenbergIter= 2
iteration= 986	chi2= 76863527.558099	time= 0.00828025	cumTime= 11.3033	edges= 1573	schur= 0	lambda= 6644002.548561	levenbergIter= 1
iteration= 987	chi2= 76863424.655119	time= 0.0138711	cumTime= 11.3172	edges= 1573	schur= 0	lambda= 8858670.064748	levenbergIter= 2
iteration= 988	chi2= 76863210.840589	time= 0.0133166	cumTime= 11.3305	edges= 1573	schur= 0	lambda= 11811560.086330	levenbergIter= 2
iteration= 989	chi2= 76863134.113172	time= 0.0082046	cumTime= 11.3387	edges= 1573	schur= 0	lambda= 7874373.390887	levenbergIter= 1
iteration= 990	chi2= 76862973.346273	time= 0.013439	cumTime= 11.3522	edges= 1573	schur= 0	lambda= 10499164.521183	levenbergIter= 2
iteration= 991	chi2= 76862928.630211	time= 0.00829797	cumTime= 11.3605	edges= 1573	schur= 0	lambda= 6999443.014122	levenbergIter= 1
iteration= 992	chi2= 76862806.531483	time= 0.0134429	cumTime= 11.3739	edges= 1573	schur= 0	lambda= 9332590.685496	levenbergIter= 2
iteration= 993	chi2= 76862573.338035	time= 0.0132096	cumTime= 11.3871	edges= 1573	schur= 0	lambda= 12443454.247327	levenbergIter= 2
iteration= 994	chi2= 76862480.210007	time= 0.00822153	cumTime= 11.3953	edges= 1573	schur= 0	lambda= 8295636.164885	levenbergIter= 1
iteration= 995	chi2= 76862311.667708	time= 0.0150183	cumTime= 11.4104	edges= 1573	schur= 0	lambda= 11060848.219847	levenbergIter= 2
iteration= 996	chi2= 76862237.459048	time= 0.00838635	cumTime= 11.4187	edges= 1573	schur= 0	lambda= 7373898.813231	levenbergIter= 1
iteration= 997	chi2= 76862101.177105	time= 0.0142863	cumTime= 11.433	edges= 1573	schur= 0	lambda= 9831865.084308	levenbergIter= 2
iteration= 998	chi2= 76862069.035958	time= 0.00850012	cumTime= 11.4415	edges= 1573	schur= 0	lambda= 6554576.722872	levenbergIter= 1
iteration= 999	chi2= 76861974.211349	time= 0.0132217	cumTime= 11.4548	edges= 1573	schur= 0	lambda= 8739435.630496	levenbergIter= 2

Reference:

[1] Arun, K. Somani, Thomas S. Huang, and Steven D. Blostein. "Least-squares fitting of two 3-D point sets." *IEEE Transactions on pattern analysis and machine intelligence* 5 (1987): 698-700.