Flowmo: Lightweight visual odometry

Author: Steph McArthur
Supervisor: Claude Sammut

Problem: Need fast, lightweight 3D odometry

Odometry tracks the movement of a robot. It can be used for controlling how the robot moves, figuring out where the robot is and to help build maps of the environment. Odometry can be calculated from a variety of sensors:

- A wheeled robot can use encoders to measure figure out how much the wheels have turned and use inverse kinematics to calculate the robots location.
- Lasers detect the distance of objects around the robot in a 2D plane. These scans can be aligned to estimate movement in a plane (2D).
- Kinect sensor generates a 3D point cloud in front of the robot which can be aligned to estimate movement in 3D.
- Cameras can be used to generate sparse or dense 3D point clouds for pose estimation.

When traversing outside, either with a robot traveling over hilly or bumpy ground or flying through the air, like the hexacopter to the right, the odometry needs to measure translation and rotation in 3D space. State of the art techniques that use cameras or lidars provide excellent odometry but are very CPU intensive to run in real time, making them not suitable to use on smaller or lower cost mobile platforms.

Approach

Flowmo: Lightweight visual odometry using raw frames and optical flow matching

Flowmo chooses a new frame to be a keyframe after the camera translates or rotates past a threshold. Each subsequent frame is compared to this keyframe to estimate the translation and rotation from the keyframe. The estimated transformation between each keyframe is summed to estimate the current location from the origin.

The flowchart above shows the steps in keyframe to current frame pose estimation:

1) Feature Detection: Features detected in the left frame with black dots, whilst on the right frame a line is drawn between the feature in the right frame and the location of the feature in the left frame.
2) Point Rectification: There are two main causes of distortion which will affect the triangulation of feature points.
   1) Radial distortion: This is most clearly seen in the left image above, where straight lines near the edges of the image are curved. The distortion is related to the distance from the center of the image and can be easily rectified.
   2) Rotational distortion: This is when the left and right image are not perfectly aligned. The epipolar line describes how a feature in the right image will move as the feature’s distance from the camera changes, whilst the position in the left image stays the same. Perfectly parallel cameras with a displacement difference in the x axis only will have a horizontal epipolar line. This is desired since it makes calculating the 3D position of the feature and removing bad point matching (points not along an epipolar line) simpler and faster.
3) 3D Projection: The location of current feature points in 3D is calculated before attempting to align them to the keyframe. Images points can be converted into 3D coordinates by the formulas on the right. The disparity (d) is the difference in x location between the left and right observations of a feature.
4) Alignment: The difference in pose between the current frame and keyframe is estimated by finding the transformation that moves the current features points so that when they are reprojected back into the keyframe the different between the features in the image space is minimized. The problem is solved using Ceres Solver, a non linear least squares solver.

Conclusions & Future work

Flowmo achieves its aim for being lightweight, allowing it to be successfully used on a hexacopter during flight and on a ground based robot. It does not yet achieve the accuracy of the best visual odometry systems but it shows reasonable odometry can be achieved with a fraction of current CPU requirements.

The biggest problems impacting the accuracy of flowmo, so would be good to look fixing at in future work, are:

- Sliding features: Features sometimes slide along lines in images (e.g. road lines), distorting the real movement.
- Detecting bad convergence: Poor alignments sometimes still converge, causing bad pose estimates to be included in the estimation.

The need for speed

Flowmo attempts to be faster than existing visual odometry methods by:

- Running on raw images: Lens distortion and misalignment between stereo cameras are often corrected before being used in the visual odometry algorithm. These distortions must be corrected to correctly triangulate feature points but it is an expensive operation (the true location of each pixel must be calculated then the image is interpolated to fix shearing or compression effects). Flowmo instead only corrects the locations of feature points, reducing the number of calculations from a few millions to a few hundred.

- Optical flow feature matching: Traditional methods run feature detection algorithms on left and right frames and then match those features between frames and to previous frames. Instead of this, flowmo uses optical flow to find features from the previous left frame and to find the same features in the current right frame. Optical flow matches left and right features by definition, also removing the need to run another algorithm to match found features. Optical flow runs three times faster than simple feature detectors (such as OpenCV’s GoodFeaturesToTrack algorithm).

Evaluation using the KITTI dataset

The accuracy and performance was evaluated using the KITTI dataset. KITTI was recorded by a car driving around highways, suburban and city streets and has a high quality ground truth recorded by DGPS.

Flowmo achieved an average of translation 10% error in the publicly available ground truths. This is pushed high by some datasets which contained scenes flowmo could not handle (e.g. highways). Some of the best tests have less than 3% translation error over 800m, which is much closer to many state of the art visual odometry algorithms which have <2% translation error. Performance wise, flowmo takes ~25ms to process a frame whilst ORBSLAM2, an accurate open source odometry took ~100ms on the same machine (which is comparable to the performance reported by the authors).