Visual Inertial Odometry

Component Overview

Visual Inertial Odometry (VIO) is a technique used to estimate a device's position and orientation by fusing data from a camera (visual) and an Inertial Measurement Unit (IMU). The IMU provides acceleration and angular velocity measurements, while the camera captures visual information to track movement over time. By combining these two sources of information, VIO achieves more accurate and robust motion tracking, compensating for the limitations of each sensor individually (e.g., IMU drift and visual ambiguity in low-light conditions).

System Integration

System integration in Visual Inertial Odometry (VIO) combines camera and IMU data to estimate motion by fusing visual feature tracking with high-frequency inertial measurements. The IMU data is pre-integrated to align with the slower camera frames, and non-linear optimization methods like the Levenberg-Marquardt are used to fuse the data, correcting IMU drift with visual information. Proper calibration ensures accurate sensor alignment, and the integrated VIO system provides real-time state estimates to higher-level systems like navigation and control in autonomous robots or AR devices.

Algorithms and Techniques

To simulate the Visual-Inertial Odometry (VIO) system, we will use the ETH Zurich drone racing dataset, which provides both camera and IMU data. The design process begins with testing Visual Odometry (VO). Keypoints are extracted from successive camera frames, and feature matching is used to establish correspondences. The 8-point algorithm is then applied to estimate the essential matrix, from which the camera's relative pose (rotation and translation) is derived. This gives an initial position estimate but can be affected by visual ambiguities, such as poor lighting or textureless environments.

Next, we handle the IMU data, which typically has a higher sampling rate than the camera. We will pre-integrate the IMU data to match the camera frame timestamps, while also modeling the inherent drift caused by sensor noise. The IMU state vector will include estimates for position, velocity, orientation, and biases for accelerometer and gyroscope readings, which are essential for drift correction.

Once both the visual and inertial measurements are processed, we fuse them using the **Levenberg-Marquardt optimization algorithm**. The optimization minimizes the residuals between the VO-derived positions and the IMU-integrated positions, compensating for errors in both systems. IMU drift is corrected by leveraging more reliable visual information, while visual estimation errors caused by lighting or motion blur are reduced by incorporating IMU data. This fusion provides a robust and accurate estimate of the drone's position and orientation over time, ensuring reliable performance even in challenging conditions.

Development Approach

Technologies and Tools

The primary technologies for development will include

- C++17, for coding
- Ceres Solver, for non-linear optimization
- OpenCV, for image processing
- The CMake, will be used for compiling and managing dependencies

External Libraries

We will leverage open-source libraries such as OpenCV for handling image processing tasks and the Ceres Solver for implementing the non-linear optimization component. These libraries are well-documented and widely used, ensuring reliability and ease of integration within the project.

Potential Risks

The **VIO** system's optimization process carries inherent risks, such as the potential for local minima, noise sensitivity in input data, poor convergence due to initialization issues, and computational inefficiencies with larger datasets. If optimization fails, the design allows for a simple replacement of the optimizer with an Extended Kalman Filter (EKF) module, maintaining the overall algorithm's integrity. This modular approach ensures that the optimizer can be swapped without altering other modules, preserving the system's functionality despite optimization challenges.

Process

First, Visual Odometry (VO) will be implemented using the ETH Zurich drone racing dataset, focusing on extracting keypoints, matching features, and estimating relative poses. This system will be thoroughly tested to address issues like noisy pose estimates. Following VO validation, IMU pre-integration will be performed, incorporating an IMU model that accounts for sensor biases, and this will also be rigorously tested. Once both pipelines are confirmed to work, they will be integrated into the optimizer for state estimation. If the optimization fails, we can seamlessly switch to an Extended Kalman Filter (EKF) module, ensuring minimal disruption to the overall algorithm.

Quality Assurance

Given the inherent susceptibility of the VIO system to multiple small errors, a rigorous quality assurance process will be implemented. Each function within the estimation modules will undergo thorough testing through unit tests, designed to cover as much functionality as possible. This approach will ensure that any potential issues are identified and addressed early in the development process, thereby enhancing the reliability and accuracy of the overall system.
Few examples:

- Keypoint Detection: Verify that the keypoint detection algorithm identifies the correct number of keypoints under varying lighting conditions using a controlled image set.
- IMU Preintegration: Confirm that IMU preintegration transforms measurements accurately by validating against expected outputs when simulating known motion.
- Reprojection Error: Test that the reprojection of 3D points into 2D results in minimal reprojection error compared to the actual 2D positions of those points in the image.

Final Deliverable

A software package that, when built and executed, will process real-time camera footage and IMU data to provide accurate VIO estimates, marking and updating the position and orientation of the robot in its environment. The final source code will include all unit tests (written using Google Test) passing with a code coverage exceeding 90%. Additionally, the code will be thoroughly documented (using Doxygen) and adhere to Google C++ style guidelines to ensure maintainability and clarity.

References

- https://github.com/HKUST-Aerial-Robotics/VINS-Mono
- https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8421746
- https://rpg.ifi.uzh.ch/docs/VO_Part_I_Scaramuzza.pdf