Visual Inertial Odometry

Aadhya Puttur, Alex Chiluisa

I. Introduction

Visual Inertial Odometry (VIO) is a method used to estimate the 3D position, orientation, and velocity of a moving vehicle with respect to an initial reference. This uses the input by only one or two cameras and one or more Inertial Measurement Units (IMU). This method is commonly used to navigate a robot (vehicle) without carrying a GPS or unreliable on it e.g. indoor navigation or navigation under a bridge [1], [2] Utilizing a stereo camera with a known distance we can obtain depth by matching features. However, with a moving robot, this can be a problem because of motion blur and matching features in general are very expensive. To solve this we need another measurement tool: IMU. This instrument measures the linear acceleration and angular acceleration of the robot. With an IMU, we can have accurate measurements of high speed and high acceleration which the camera lacks. Although, it has uncertainty when the acceleration is small because the signal-to-noise ratio decreases as the robot slows down which is why we use the camera. Therefore, both the IMU and cameras suffer from short-term drift and long-term drift respectively. The fusion of IMU and camera leads us to a multi-modal fusion problem which can be used to calculate the accurate pose of the camera to backtrack depth. We will be using tightly coupled visual inertial odometry because we focus on minimizing the error of the estimated poses to the ground truth poses Fig 1.

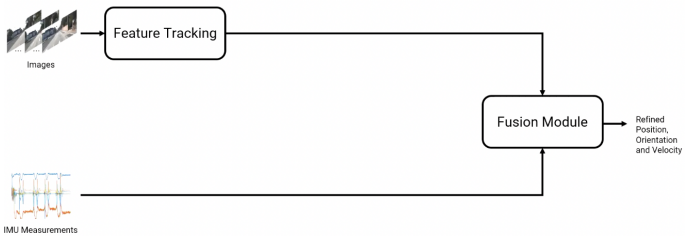

Fig. 1. VIO Tightly Coupled: State has position, orientation, IMU biases and 3D world point locations

II. Data

We will be using the Machine Hall 01 easy which is the subset of the EuRoC dataset to test our implementation. The data is represented in the form of $[time, x, y, z, qx, qy, qz, qw]$ which represents the camera pose obtained through a VI sensor. The data includes 2000 Hz IMU messages. The data is collected from perspective of a quadcopter. We utilize the ground truth trajectory of the quadcopter, which is through a sub-mm accurate Vicon Motion capture system, in order to aligned the estimate with the ground truth. Fig 2.

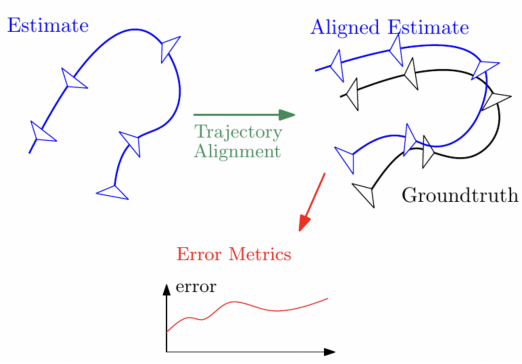

Fig. 2. Trajectory Evaluation for Visual-Inertial Odometry

III. Method

For this project, we are implementing Stereo Multi-State Constraint Kalman Filter (S-MSCKF), since we are using a stereo camera. We were given a starter code and we had to implement the following functions by using [1], [3]:

1) Initialize gravity and IMU bias
2) Batch Imu Processing
3) Process Model
4) Predict new State
5) State Augmentation
6) Add Feature Observations
7) Measurement Update
8) Predict New State

IV. Initialization

We first initialize the IMU bias and initial orientation from the first few IMU readings that we receive from the buffer. Then we initialize the gyro bias given the current angular and linear velocity. Gyroscopes are subject to bias instabilities in which the initial zero reading of the gyroscope will cause drift over time due to noise within the device. We estimate the gravity direction from IMU when the robot is stationary. Here we calculate the error of the IMU state. In order to perform this we must remove the bias from the measured

A. Chiluisa is with the Department of Robotics Engineering, Worcester Polytechnic Institute, Worcester, MA 01609, USA (e-mail: ajchiluisa@wpi.edu)

A. Puttur is with the Department of Computer Science, Worcester Polytechnic Institute, Worcester, MA 01609, USA (e-mail: aputtur@wpi.edu)

gyro and acceleration. Next we compute discrete transition and noise covariance matrix.

We define the IMU state Fig 3.

$$\mathbf{x}_I = \begin{pmatrix} {}^I_G\mathbf{q}^\top & \mathbf{b}_g^\top & {}^G\mathbf{v}_I^\top & \mathbf{b}_a^\top & {}^G\mathbf{p}_I^\top & {}^I_C\mathbf{q}^\top & {}^I\mathbf{p}_C^\top \end{pmatrix}^\top$$

Fig. 3. IMU State definition, first variable represents the rotation from the inertial frame to the body frame

Now you know the gravity direction. We can estimate the orientation by accumulating angular velocity from a gyrometer. We also calculate acceleration in the frames in the buffer to use that value to find the gravity.Then we initialize the initial orientation, so that the estimation is consistent with the inertial frame. We do this because we want to estimate the gravity direction from the IMU in those few frames in the buffer to get the orientation. To feed in messages we first need to process the IMU message given the time bound. We execute the process model, update the state information, and then repeat these steps until the time bound is reached.

V. Process Model

The MSCFK is an efficient method for light-weight sensor fusion. It is a two-step process where we do inertial propagation and measurement update Fig 5. With this method we are feature tracking. We create sliding window filter. The sliding window filter basically maintains a fixed window of features and camera poses. The EKF is using one camera and multiple feature points. MSCKF uses one feature and constrains many states. MSCKF has a fixed window and uses feature matching.

For the IMU we take pre-integration step where we only update the parts that depend on the IMU. Then you save the pre-integration in the buffer.

$$\begin{aligned} {}^I_G\dot{\hat{\mathbf{q}}} &= \frac{1}{2}\Omega(\hat{\boldsymbol{\omega}}){}^I_G\hat{\mathbf{q}}, \quad \dot{\hat{\mathbf{b}}}_g = \mathbf{0}_{3\times 1}, \\ {}^G\dot{\hat{\mathbf{v}}} &= C\left({}^I_G\hat{\mathbf{q}}\right)^\top \hat{\mathbf{a}} + {}^G\mathbf{g}, \\ \dot{\hat{b}}_a &= \mathbf{0}_{3\times 1}, \quad {}^G\dot{\hat{\mathbf{p}}}_I = {}^G\hat{\mathbf{v}}, \\ {}^I_C\dot{\hat{\mathbf{q}}} &= \mathbf{0}_{3\times 1}, \quad {}^I\dot{\hat{\mathbf{p}}}_C = \mathbf{0}_{3\times 1} \end{aligned}$$

Fig. 4. continuos dynamics of estimated IMU

We have the IMU measurements for angular velocity and acceleration with their biases removed Fig 4.

For the IMU Measurement Model we have $\omega_m = \omega + b_\omega + n_\omega$ and the accelerometer is $a_m = a + R^G g(rotatedgravity) + b_a + n_a$. The motion model or the state evolution model is the physics of the system with the quartonian evolution, point position, velocity, acceleration, and bias. We use the discrete state evolutional model to make it useful in the world. For the IMU we take pre-integration step where we only update the parts that depend on the IMU. Then you save the pre-integration in the buffer.

We then get the observational model of the camera by using distortion. The process model and measurement model don't have to run together since your process updates will be a lot more frequent than your measurement updates.

The MSCKF allows for updating features without inserting their estimates into the state vector. It does this by constraining multiple states with a single point. Although, it does not save point positions.

We estimate the new state of the IMU. We start by normalizing the measurement of the gyroscope and finding the omega matrix

$$\Omega(\widehat{w}) = \begin{pmatrix} -[\widehat{w}_x] & w \\ -w^T & 0 \end{pmatrix} \tag{1}$$

Then we apply a 4th-order Runge-Kutta numerical integration on eq. 1 on the paper [1] to propagate the estimated IMU state

The linearized continuous dynamics for the error IMU state follows $x_I = Fx_I + Gn_I$ We then also propogate the state covariance matrix and then update the state corresponds to null space.

Instead of propagating the entire state, we just need to propagate the error because is reduces computational complexity. $x_t = x \oplus \delta x$ The true state, x_t is the state we are estimating, nominal state (x) has no noise or perturbation. then we have the error state δx We are only propagating the error state which is the δ.

The state estimate is used to compute feature position estimate using triangulation. Feature position error is correlated with the error and we don't want to use that in the state. We want to decorrelate by performing projection of residual on left nullspace of feature Jacobian. Then we perform the EFK update.

The state augmentation is where we compute the state covariance matrix in equation (3) in the "MSCKF" paper. We first get the imu state, rotation from imu to cam0, and translation from cam0 to imu. We are only estimating the left camera pose because the extrinsic between the two cameras is already known. Next, we add a new camera state to the state server. When new images are received, the state should be augmented with the new camera state.

Marginalization is the removal of old or unwanted features. It is when the movement is greater than some amount in order to drop old frames or when the feature is not tracked anymore. As well as if the max number of camera poses are reached. We drop out old frames where in the buffer we only have a certain number of frames.

VI. Results

Using the viewer provided in the starter code, we simulate the trajectory using the "Machine Hall 01 easy" dataset. We used the camera and the IMU measurement while the robot track the trajectory and provides the estimated position. You can find a video with the expected results as "Output.mp4". Originally the video takes around an hour to complete the full trajectory, however, the attached to this report is sped up

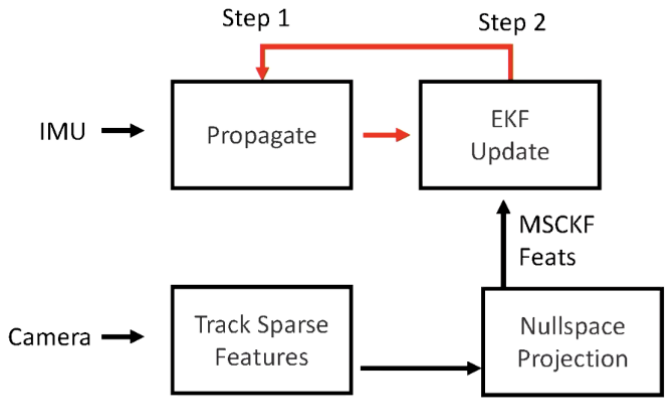

Fig. 5. MSCKF Estimation

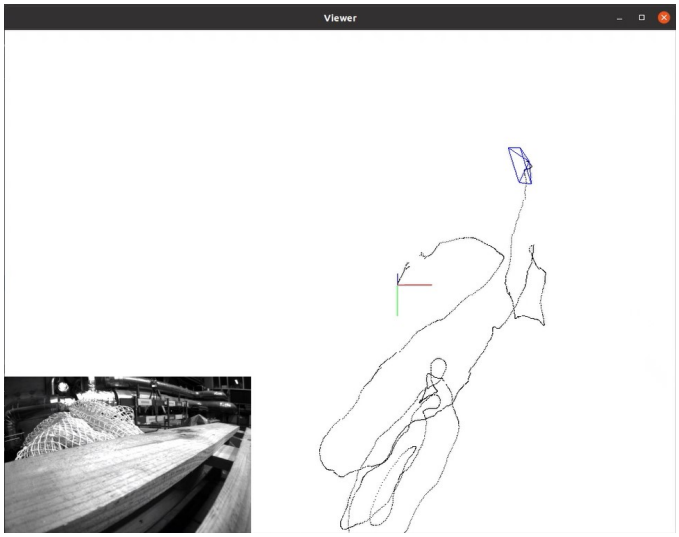

Fig. 6. Final Trajectory

20 times. Fig. 6 shows the final state of the vehicle and the final trajectory.

Moreover, we were requested to calculate the RMSE ATE using any toolbox kit, we selected to use the "rpg_trajectory_evaluation" [4]. First, we collected the estimated poses data in a .txt file within the msckf.py file called "stamped_traj_estimate.txt". Using this late file and the "stamped_groundtruth.tx" with a SE(3) alignment we determine the trajectory as Fig. 7 shown.

The absolute error values that we obtained are shown in Fig. 8. Regarding translation, the RMSE that we calculated is 8.04 [cm].

REFERENCES

[1] K. Sun, K. Mohta, B. Pfrommer, M. Watterson, S. Liu, Y. Mulgaonkar, C. J. Taylor, and V. Kumar, "Robust stereo visual inertial odometry for fast autonomous flight," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 965–972, 2018.
[2] https://docs.px4.io/main/en/computer_vision/visual_inertial_odometry.html.
[3] https://github.com/uoip/stereo_msckf.
[4] https://github.com/uzh-rpg/rpg_trajectory_evaluation.

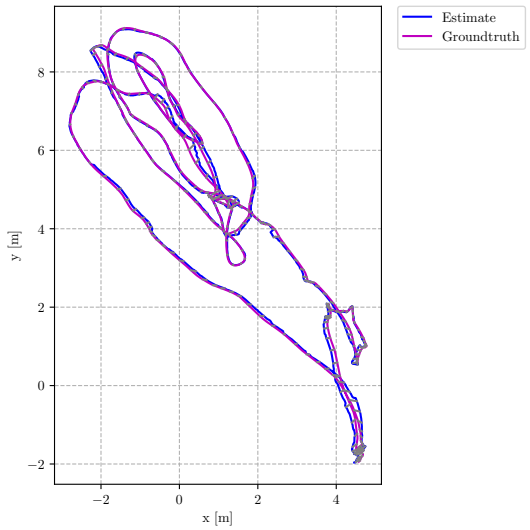

Fig. 7. Final Trajectory

Fig. 8. Absolute Errors