Project Report - EE698G
Marker based localization of a quadrotor

Akshat Agarwal[1] and Siddharth Tanwar[2]

I. Objective

To implement a high level control pipeline on a quadrotor which can autonomously takeoff, track and hover over a marker, and land on it with high precision.

II. Motivation

With the recent proliferation of UAVs, there is an increasing need for giving quadrotors the ability to deploy themselves autonomously for extended periods of time. However, quads have very limited flight times due to high power consumption and limitations in existing battery technology. To be deployed over a large area/timeframe, they must be able to land with high precision on landing spots (either predecided or detected on-the-fly), and charge themselves. The mechanical docking of the chargers would obviously require very high precision in landing.

III. Review of the Reference Paper

We followed a paper [1] published recently in the International Conference for Robotics and Automation 2015 by Yang et al, which uses SR-UKF vision perception for precise autonomous quadrotor landing in both indoor and outdoor environments.

The paper targets landing a large quadrotor systems ($>$1.5 kg) while achieving landing precision within 5 centimeters. Such accuracy is required for autonomous long-term deployment of quads, since they have limited flight time and need to recharge after every 15-20 minutes, depending on the payload. Autonomously charging requires them to land on a specified location with such accuracy that allows the mechanical docking of the charging points. Moreover, landing on the ground is made difficult due to the ground effect [2], which is mitigated by using a small platform above the ground.

The system proposed by the authors uses one downward looking camera to detect the landing pad, and an optical flow sensor for measuring the speed of quadrotor relative to the ground. It also uses an IMU and GPS provided by the drone manufacturers (DJI). The quadrotor attitude control algorithm runs on the High Level Processor of the drone itself, processing IMU, GPS and optical flow sensor velocity data. The on-board computer handles the data from camera and executes the landing control algorithm, including SRUKF position estimation [3] and PID control.

*We would like to thank Prof. Gaurav Pandey for providing us the opportunity to do this project

[1]Akshat Agarwal is an undergraduate student in the Department of Electrical Engineering, Indian Institute of Technology Kanpur agarwalaks30@gmail.com

[2]Siddharth Tanwar is an undergraduate student in the Department of Electrical Engineering, Indian Institute of Technology Kanpur siddharthtanwar2996@gmail.com

A. Landing Pad Position Estimation

The landing pad needs to be detected first and then the relative position between the UAV and landing pad needs to be calculated.

- **Markers are generated and detected using the ArUco [4] marker library**. The detection and identification steps uses an image processing pipeline, and gives corner points of the detected marker as an output. Since the real size of marker is known, a correspondence between image points and real points is easily established and used to form a PnP problem, from which R and T are obtained.
- **Rotation Compensation**: The IMU has a dedicated gyroscope and hence measures rotation much more accurately than the PnP solver. Within the PnP solver, yaw angle is much more stable than pitch and roll, both of which are afflicted with noise. So the authors use IMU to compensate PnP rotation. Rotation compensation works by decomposing the rotation into a torsion and a tilt component, where the torsion only involves rotation around yaw axis, while tilt contains rotation around pitch and roll axes. So the tilt component of IMU is used to compensate the tilt component of PnP, and is multiplied with the torsion component of PnP rotation, resulting in a more precise rotation measure.
- **SRUKF position estimation**: Kalman filtering is used to reduce noise in yaw and translation estimates. SRUKF [3] has the advantage of a higher output rate over the UKF [5], while again avoiding the need to compute a Jacobian. The state consists of translation component T^{tb}, rotation (quaternion) R^{tb}, velocity of quadrotor v^t and the angular velocity of quadrotor ω^b. The process model is shown in Fig. 1. The symbol $\times$ means quaternion multiplication and a^t_{i-1} is obtained from IMU measurement. The measurement model is shown in Fig. 2 where T^{tb} and q^{tb} are obtained from PnP problem after rotation compensation. Linear acceleration and angular velocity is obtained from IMU. For velocity, the authors use two methods: 1) optical flow sensor, and 2) differentiate T^{tb}. While 2) is sensitive to noise, 1) may be incorrect if there are moving objects in the quad's field of view.

$$x_i = \begin{bmatrix} {T^{tb}}_i \\ {q^{tb}}_i \\ {v^t}_i \\ {\omega^b}_i \end{bmatrix} = \begin{bmatrix} {T^{tb}}_{i-1} + {v^t}_{i-1}\Delta t \\ {q^{tb}}_{i-1} \times q({\omega^b}_{i-1}\Delta t) \\ {v^t}_{i-1} + {a^t}_{i-1}\Delta t \\ {\omega^b}_{i-1} \end{bmatrix}$$

Fig. 1. Process Model

$$y_i = \begin{bmatrix} T^{tb} \\ q^{tb} \\ \frac{1}{w_1+w_2}[w_1({v^t}_i + a^t\Delta t) + w_2\frac{T^{tb}-{T^{tb}}_i}{\Delta t}] \\ \omega^b \end{bmatrix}$$

Fig. 2. Measurement Model

B. Landing Process

Landing involves position based visual servoing, with the landing sequence being: 1) Detect marker, 2) Obtain estimated position and orientation w.r.t. landing pad by SRUKF, 3) Rotate to target yaw angle 0, 4) Control to target position (0,0,50) that is eliminate xy-plane displacements, and 5) Control to target speed (0,0,-30) expressed in body frame b.

C. Results

With IMU data at 100Hz, marker detection at 50Hz, the controller generates output with frequency 50Hz. The system lands with a mean error of 3.1cm indoors, and 2.9cm outdoors.

IV. Our Methodology

We adapted the paper's methodology according to the resources available to us, both in terms of hardware and the algorithm used.

A. Hardware Setup

We used the Nayan quadrotor shown in Fig. 3, developed by AUS with an Odroid XU-4 on-board computer running Lubuntu 14.04. The quadrotor has a twin ARM cortex M4 processor with a Real-Time OS for the flight controller (High Level + Low Level Processor). It has the following sensors on it:

Fig. 3. Nayan Quadrotor

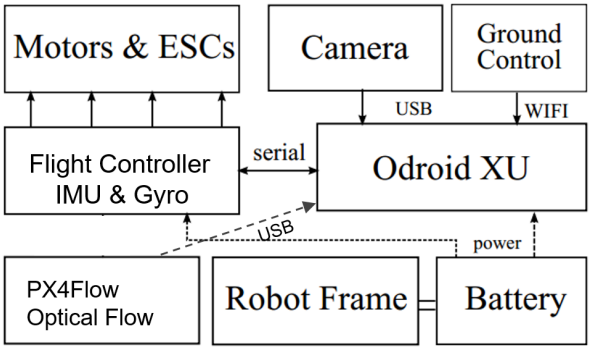

Fig. 4. Hardware architecture

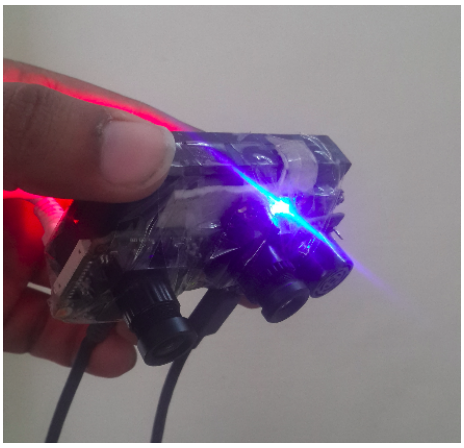

Fig. 5. Our testbed consisting of the Pixhawk flight controller, the mvBluefox camera and PX4Flow sensor

1) Monocular Camera: The mvBluefox USB 2.0 gray scale camera is installed facing downward. It has a maximum frame rate of 90Hz, but we used it at 60Hz since that was good enough for our purpose. Although it has adjustable gain and exposure, it performs well only in well illuminated environments, restricting our test times. It has a pre-built driver in ROS.
2) PX4Flow: This sensor has an optical flow sensor which gives x-,y- velocities at 400Hz, and a sonar which gives distance to ground. It is supposed to work in both indoor and outdoor environments, but gives good quality only in well lit ones. This too has a driver in ROS.
3) IMU: Provides linear acceleration using accelerometers (-8G to +8G) and angular rates using gyroscopes (max 2000deg/sec) to flight controller. Has an absolute reference frame towards true North, and is used to estimate orientation of the quad.

The hardware architecture is shown in Fig. 4. The Nayan quadrotor available to us gave erroneous values of IMU rotation, due to which we were unable to implement our code on that. Instead, we used a small test-bed consisting of the sensors listed above to record data and then test our localization algorithm on that. This test-bed can be seen in Fig. 5

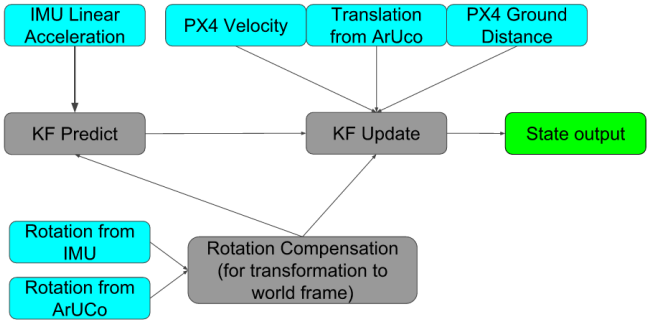

Fig. 6. Data flow through our pipeline: Blue represents input from sensor, gray represents processing step, and green represents output

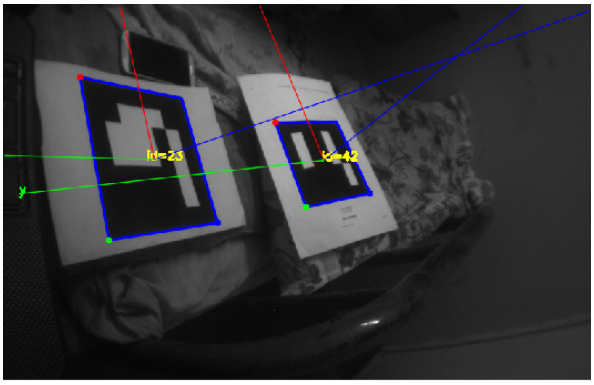

Fig. 7. ArUco markers being localized

B. Software

We used the Robot Operating System (ROS) [6] since it has pre-built packages for the sensors and also provides a great robust interface for inter-process communication. We have also used the ArUco marker library provided by Pal Robotics to detect marker and localize the camera's position with respect to the marker.

Due to difficulties in getting the SRUKF to converge properly, we used a simple Kalman Filter [7] after assuming a linear system model, where our state is $\mathbf{x} = [x\ y\ z\ x'\ y'\ z']$ where the primed coordinates indicate velocity. The motion model is assumed to be linear, taking action to be the acceleration derived from the quadrotor, and position is updated simply by taking $x_i = x_{i-1} + x'_{i-1}\Delta t$. The measurement model accepts inputs from ArUco and updates the translation components x,y and z. The optical flow data from PX4Flow sensor is used to update the x' and y' components. The sonar data is used to update the z component of state. We use rotation compensation to compensate PnP rotation with IMU rotation, and use the compensated value as the transformation to be applied to bring the sensor values from quadrotor's body frame to the Earth frame. Fig. 6 Shows the data flow in our algorithm.

V. Results

The ArUco marker localization can be seen in Fig. 7 where we see two markers being simultaneously tracked and localized by the test-bed. The result of localization of z-coordinate of the drone from our control pipeline can be seen in Fig. 8.

VI. Conclusions and Future Work

In this project, we built an architecture for implementing SRUKF, UKF and KF on the quadrotor. We have tracked pose of the quadrotor using the ArUco markers and using variants of the Kalman Filter. We conclude that sonar data is often unreliable and produces impulse noises, as can be seen in Fig. 8. Our Kalman Filter is enough to handle this and produces very good results. We also observe that camera data is reliable if and only if the ArUco marker stays completely in the camera's field of view. We also see that rotation compensation helps in producing a much better estimate of the quadrotor body to earth transformation, leading to greater localization precision.

Future work includes testing our localization on the actual quadrotor, and tune the control system to make it respond better to our code.

References

[1] S. Yang, J. Ying, Y. Lu, and Z. Li, "Precise quadrotor autonomous landing with srukf vision perception," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 2196–2201.

[2] G. J. Leishman, *Principles of helicopter aerodynamics with CD extra*. Cambridge university press, 2006.

[3] R. Van Der Merwe and E. A. Wan, "The square-root unscented kalman filter for state and parameter-estimation," in *Acoustics, Speech, and Signal Processing, 2001. Proceedings.(ICASSP'01). 2001 IEEE International Conference on*, vol. 6. IEEE, 2001, pp. 3461–3464.

[4] S. Garrido-Jurado, R. M. noz Salinas, F. Madrid-Cuevas, and M. Marín-Jiménez, "Automatic generation and detection of highly reliable fiducial markers under occlusion," *Pattern Recognition*, vol. 47, no. 6, pp. 2280 – 2292, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0031320314000235

[5] E. A. Wan and R. Van Der Merwe, "The unscented kalman filter for nonlinear estimation," in *Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000. AS-SPCC. The IEEE 2000*. Ieee, 2000, pp. 153–158.

[6] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2, 2009, p. 5.

[7] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Journal of basic Engineering*, vol. 82, no. 1, pp. 35–45, 1960.

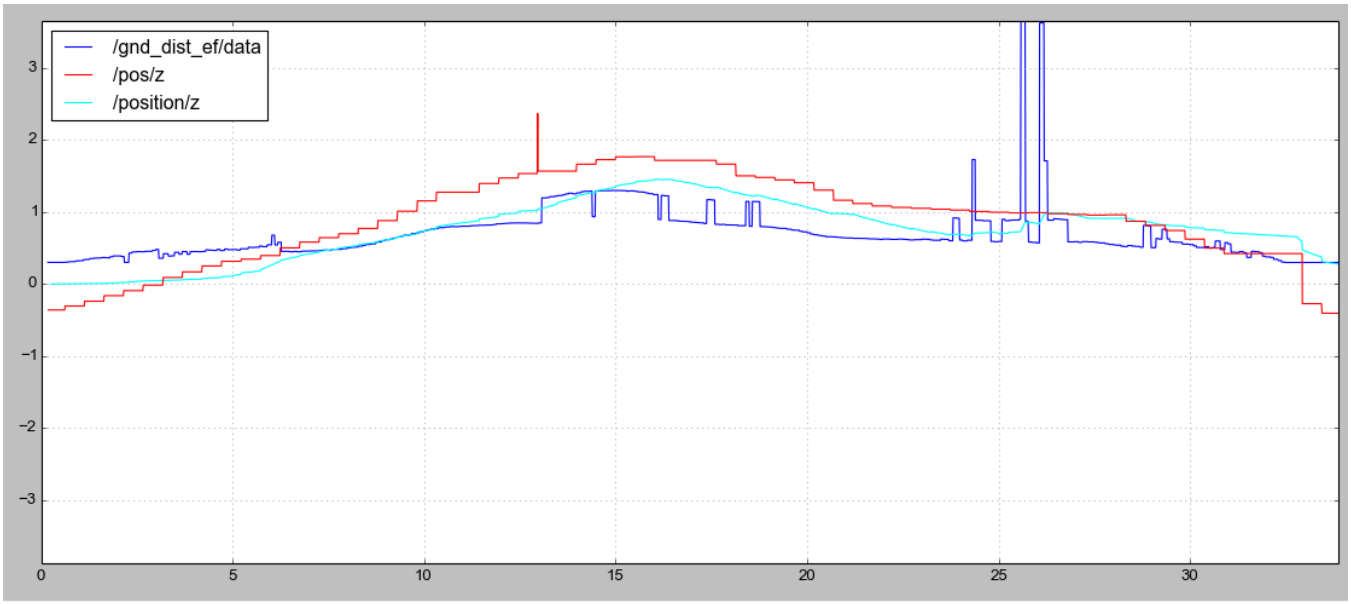

Fig. 8. Shows the result of z-coordinate localization using our Kalman Filter. The red line is z-coordinate from ArUco localization, dark blue line is from SONAR data, and the light blue line is what is produced after Kalman Filtering