# State Estimation, SLAM, and Obstacle Detection on a Low Powered Onboard Computer

Cole Gulino (cgulino)
Carnegie Mellon School of Computer Science MRSD c.o. 2016
5000 Forbes Ave, Pittsburgh, PA 15213
cole.gulino@gmail.com

## Abstract

*Planning, control, and state estimation is an integral part of all mobile robotics applications. This problem is exacerbated in a constantly dynamic system. A quadcopter must constantly maintain and update state in order to properly plan and control itself. This problem becomes even more difficult in an indoor area where you are GPS denied. This paper aims to identify and quantify the drawbacks of several different vision-based localization and state estimation systems. It further develops a framework for using vision to maneuver indoors with static obstacles without global state estimation real-time on a low-powered onboard computer.*

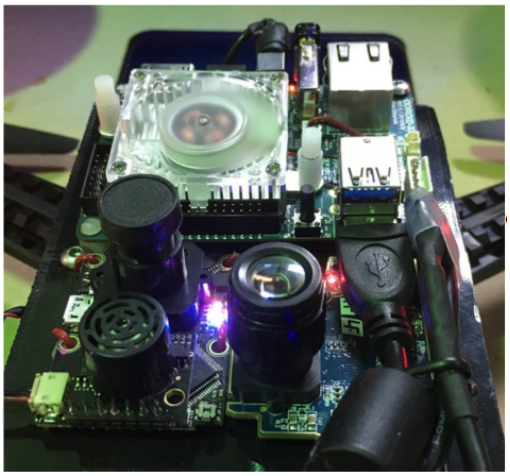

Figure 1. Vision Systems Connected to the On-Board Computer

## 1. Introduction

This paper will outline several vision-based methods for localization, state estimation, mapping, and obstacle detection real-time with a single-board computer on a quadcopter. The paper aims to layout the major drawbacks and strengths of each of these vision-based systems. All of the results were generated real-time without any off-board computation. The conclusion of the paper is a recommendation for an efficient way to plan locally around obstacles using vision-based methods on a quadcopter.

### 1.1. The HardwareSystem

The robot body is the Iris+ from 3DR [10]. The Iris+ is a four motor drone which contains the Pixhawk [6] flight controller. This flight controller contains an IMU with a gyroscope and an accelerometer. It also provides an IO system that can communicate with ROS through the MAVROS [3] package.

The vision system comprises of three different cameras: PX4FLOW [12], PlayStation Eye [9], and Asus Xtion Pro Live [11].

The PX4FLOW is a CMOS device that provides velocity updates through an I2C cable. It is designed to coordinate and work seamlessly with the Pixhawk [6] flight controller. It automatically provides velocity measurements to the Pixhawk's onboard Extended Kalman Filter in order to improve state estimation during flight.

The PlayStation Eye is an inexpensive digital camera that can run at 120 fps. This high frame rate makes it effective for running April Tag [14] recognition and pose estimation. The camera's low cost and light weight make it appropriate for low-powered/low-cost applications. The PlayStation Eye communicates to an on-board computer with USB.

The Xtion Pro Live is an RGB-D sensor made by Asus. The sensor is designed for a developer environment. It uses structured light in order to provide depth. By using this with OpenNI [15], I was able to get a dense 3D point cloud within the field of view of the camera.

Figure 1 shows an image of the hardware system.

Figure 2 shows an image of the Asus Xtion Pro Live.

I ran the system on two separate on-board computers in order to test out the effectiveness of each. We used the Min-

Figure 2. Asus Xtion Pro Live

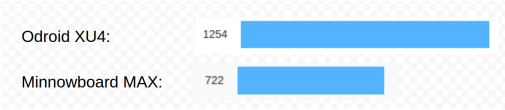


Figure 3. Benchmark Comparison of the Odroid XU4 and Minnowboard MAX

nowboard Max [4] and the Odroid XU4 [5].

Figure 3 [1] shows a benchmark test of the two systems.

The comparison between the two boards goes beyond just benchmark speed tests. Another major diversification between the systems is the Instruction Architectures. Odroid XU4 runs a paired down of Ubuntu on an ARM processor, while the Minnowboard MAX runs full Ubuntu on an x86 processor. The Odroid XU4 has the problem that there are many packages that will not compile on the ARM system. Specifically for my needs, RTAB-Map cannot compile on an ARM system.

The pro of the Odroid XU4 is that it combines a much faster processor with a light-weight operating system. This allows it to run significantly faster than the Minnowboard MAX in theory. In my experiments, I found that the Odroid XU4 did indeed run about twice as fast as the Minnowboard Max for many applications. Specifically, I was able to get an update rate of around 8 Hz for April Tag recognition on the Odroid XU4, while I was only getting between 3-4 Hz on the Minnowboard MAX.

### 1.2. Software System

The software system consists of multiple parts. Figure 4 shows a diagram of the software system.

The software system has a main code bank that feeds into the larger system that includes planning and control for the quadcopter. This system is fed information for state estimation, mapping, localization, and obstacle detection from the computer vision system sub-functions shown.

The first major subsystem is the optical flow system which is shown in the top row of Figure 4. The PX4FLOW sensor provides velocity estimates directly. The CMOS sensor processes the camera images and reports the velocity es-

vision software system (1).png

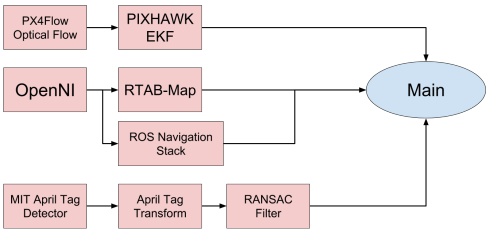


Figure 4. Diagram of the Computer Vision Software System

timates directly. This is then fed into the internal Extended Kalman Filter on the PIXHAWK flight controller.

The next major subsystem is on the second row of Figure 4. This system is responsible for localization, mapping, and obstacle detection. I used OpenNI in order to gather RGB and depth point cloud data from the Asus Xtion Pro Live camera. This data was then fed into the ROS Navigation Stack [7]. The same data was analyzed in two ways. First the data was used with the RTAB-Map [13] Package in order to generate a persistent map that could be used for Simultaneous Localization and Mapping. I also analyzed the data as a 2D cost-map. In order to accomplish this, I needed to provide a transform tree from global frame - body frame - camera frame - left and right stereo frame.

Figure 5 shows a visualization of the transform tree for my project.

The final subsystem involves localization using April Tags. Figure 6 shows an image of the April Tag [14] system used in my project.

I was required to do a transformation and a filtering on the April Tag information in order to get accurate readings from the April Tags.

## 2. Motivation

Quadcopters are a burgeoning market in the larger robotics industry. If quadcopters are going to move into larger areas of practical applications, there are a few major hurdles that must be overcome.

One of the major hurdles of quadcopters is that they are constantly in dynamic motion. Unlike most other robots, a quadcopter must be constantly controlled in order to maintain a single position, let alone plan in a cluttered environment.

The dual to the planning problem is the state estimation problem. Without accurate state estimation, no amount of control will make a difference.

For outdoor environments, the robot can utilize GPS data in order to get accurate real-time global position updates. Unfortunately, a lot of applications involve indoor flight. In an indoor setting, the robot must rely on other means of

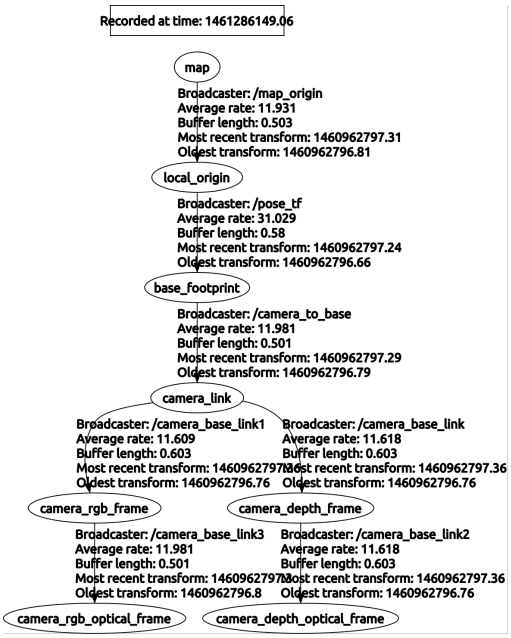


Figure 5. Transform Tree of My Robotic System


Figure 6. Images of MIT April Tags

localization and state estimation.

Beyond the technical limits of indoor state estimation, there is an obvious need to lower costs by using inexpensive visual sensors over more accurate but expensive LiDAR. In line with this, there are other power considerations that limit the kind of sensors and processors that you can run on the quadcopter.

The motivation for this project is to develop a reliable state estimation system on a low-powered/light-weight device with cheap visual sensors.

### 2.1. Related Work

Good results have been shown using LSD-SLAM [2] with a monocular camera. This provides a powerful and effective system using only a single camera.

This system has the advantage that it only requires a single monocular camera for a sensor. The disadvantage is that it is computationally intensive. Another major downfall of this system is that monocular vision is not as robust as stereo vision or time-of-flight sensors.

A team of researches at University of Pennsylvania and CMU's Nathan Michael have shown that you can get accurate state estimation using visual systems on a quadcopter [16].

This paper aims to use a visual-intertial system in order to provide state estimate on inexpensive visual sensors.

## 3. Optical Flow and Visual Odometry

As explained in the introduction, I used the PX4FLOW sensor in order to pass along direct velocity estimates of the quadcopter's motions to the internal Extended Kalman Filter of the PIXHAWK flight controller. Figure 4 describes the software system where the top row shows the optical flow and visual odometry subsystem.

The PX4FLOW sensor has a CMOS processor that runs the optical flow algorithm in order to provide an estimate of the velocity of the quadcopter at a moment in time. The sensor is quite efficient and provides 200 Hz update rate.

The paper [12] outlines the derivation of the velocity estimates.

The projected pixel coordinates

$$\mathbf{P} = \begin{bmatrix} X, Y, Z \end{bmatrix}^T$$

The relative motion between the camera and the projected pixel coordinates is given by

$$\mathbf{V} = -\mathbf{T} - \omega \times \mathbf{P}$$

Where here $\omega$ is the angular velocity and $\mathbf{T}$ is the translational component of the motion.

The paper[12] then outlines that taking the derivative with respect to time of both sides of the above equation leads to the relation between the velocity of $\mathbf{P}$ in the camera reference frame and the velocity of flow of

$$\mathbf{p} = f\frac{\mathbf{P}}{Z}$$

in the image plane

$$\frac{\textbf{flow}}{\Delta \text{time}} = \mathbf{v} = \mathrm{f}\frac{Z\mathbf{V} - V_z\mathbf{P}}{Z^2}$$

The paper then expresses these values in the x and y coordinates of the global frame. These can be written as

$$v_x = \frac{T_z x - T_x \mathrm{f}}{Z} - \omega_y \mathrm{f} + \omega_z y + \frac{\omega_x xy - \omega_y x^2}{\mathrm{f}}$$

$$v_y = \frac{T_z y - T_y \mathrm{f}}{Z} + \omega_x \mathrm{f} - \omega_z x + \frac{\omega_x y^2 - \omega_y xy}{\mathrm{f}}$$

These velocity estimates are passed along to the internal Extended Kalman Filter inside the PIXHAWK flight controller.

The Extended Kalman Filter is a popular filter that allows for fusing estimates from many sensors that are weighted by the confidence of the measurement (specified by the covariance).

The EKF maintains an estimate of the state even in the absence of sensor information by maintaining a simulation of the dynamics of the quadcopter.

When a new measurement comes in, it is taken into account and weighed along with the simulation estimates and measurement updates by their respective covariances.

Let us consider that the state is a function of the previous state, the control input, and some noise [8].

$$\mathbf{x}_k = f(\mathbf{x}_{k-1}, \mathbf{u}_k) + \mathbf{w}_k$$

And a measurement is a function of the state and some measurement noise.

$$\mathbf{z}_k = h(\mathbf{x}_k) + \mathbf{v}_k$$

When a new measurement comes in, we update the state and the covariance of the state based on the measurement.

If we let the following be the predicted state and covariance estimate before a measurement update.

$$\hat{\mathbf{x}}_{k|k-1} = f\left(\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{u}_k\right)$$

$$\mathbf{P}_{k|k-1} = \mathbf{F}_{k-1}\mathbf{P}_{k-1|k-1}\mathbf{F}_{k-1}^T + \mathbf{Q}_k$$

Where $\mathbf{F}$ is the linearization of the system, and $\mathbf{Q}$ is the covariance of the noise in the state.

The EKF algorithm specifies a gain $\mathbf{K}$ that provides near optimality for the state and covariance update.

$$\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}_k^T + \mathbf{R}_k$$

Where $\mathbf{H}$ is the Jacobian of the function that transforms the measurement into the state.

Then the state can be updated with the new measurement $\mathbf{y}_k = \mathbf{z}_k - h\left(\hat{\mathbf{x}}_{k|k-1}\right)$

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k\mathbf{y}_k$$

$$\mathbf{P}_{k|k} = \left(\mathbf{I} - \mathbf{K}_k\mathbf{H}_k\right)\mathbf{P}_{k|k-1}$$

For our system we consider a state to be a pose (position and orientation).

$$\mathbf{x} = \begin{bmatrix} x, y, z, q_x, q_y, q_z, q_w \end{bmatrix}$$

## 4. Localization with April Tags

The MIT April Tag package [14] provides a way to recognize and track position using the April Tag. The paper outlines their methods.

The April Tags are designed so that an edge detector using gradient magnitude and orientation in order to recognize known features. This highly controlled system allows for high accuracy in recognition and pose estimates from the tag.

This information is passed onto the camera's frame. In order to transform this information into the quadcopter's body frame, I needed to provide a translation and rotation between the camera frame and quadcopter's body frame.

The April Tag package provides information in the form

$$\mathbf{x} = \begin{bmatrix} x, y, z, roll, pitch, yaw \end{bmatrix}$$

This information is quite stable in $(x, y, z)$, but it is not as stable in $(roll, pitch, yaw)$.

From the state given by the April Tag, I was able to transform the visual information from the camera's frame to the body frame of the quadcopter.

$$\mathbf{x}_{camera}^{tag} = R^{-1}\mathbf{x}_{tag}^{camera}$$

The notation here says that I am converting between the tag in the camera frame to the camera in the tag frame.

This rotation matrix is quite noisy due to the poor estimates of the orientation estimate from the April Tag. In order to remedy this, I used RANSAC in order to filter out spurious results.

For RANSAC, I used Euclidean distance as the cost. I was directly comparing to a hyperplane in the 6 dimensional state.

## 5. SLAM and Obstacle Detection

Simultaneous Localization and Mapping is a popular method in robotics for providing accurate state estimation while creating a persistent map of the environment. Obstacle detection is a by-product of SLAM, but SLAM is not required to have obstacle detection.

I used the OpenNI [15] package in order to process the 3D point clouds from the Asus Xtion Pro Live [11].

For SLAM, I used the RTAB-Map [13] package with ROS (Robotic Operating System). RTAB-map uses a graph and node based system which processes RGB and depth images. It then takes these images and maintains a collection of visual words.

RTAB-map detects loop closures using visual words. It provides localization information with SURF features and RANSAC as a filtering method. Graph optimization is implemented with TORO Graph Optimization with poses and link transformations used as constraints.

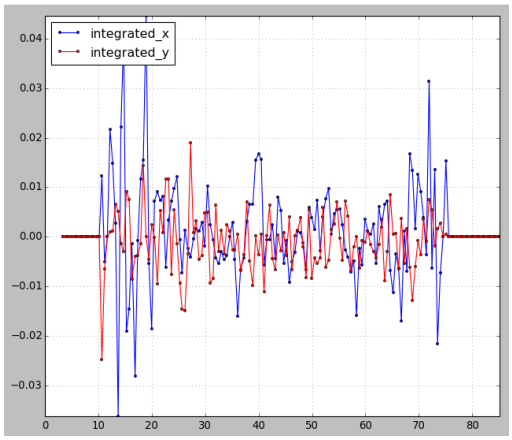


Figure 7. PX4FLOW Velocity Measurements

The RGB-D snesor is an Asus Xtion Pro Live that uses structured light with stereo in order to provide the depth information.

Obstacle detection is implemented within the ROS Navigation Stack by projecting the 3D point cloud data onto a 2D occupancy grid which provides a probabilistic based cost of an obstacle being present in the 2D projection of the seven dimensional state.

The occupancy fills based on the odometry information and the camera's point cloude provided by the user. The transform tree that provides the odometry and camera information are shown in Figure 5. These transforms place the camera information in the same frame as the body of the quadcopter. Obstacles persist for as long as the rolling window of the map (centered around the quadcopter's body frame) contains that obstacle's estimated position in the costmap window. Obstacles are removed by using ray-tracing and odometry information to determine if an object that was originally at a position is no longer there.

## 6. Results

This section will provide data and observations on how well each of the systems mentioned in sections 3-5 performed.

### 6.1. Optical Flow and Visual Odometry

As mentioned in section 3, optical flow provides velocity measurements for the quadcopter while in flight.

Figure 7 shows the velocity measurements from the optical flow sensor while the quadcopter was trying to maintain the same position.

This information has a high frequency which allows for consistent velocity measurement updates. Because this information is only velocity, there is a considerable drift that results from a linear drift associated with integrating random noise in an EKF.

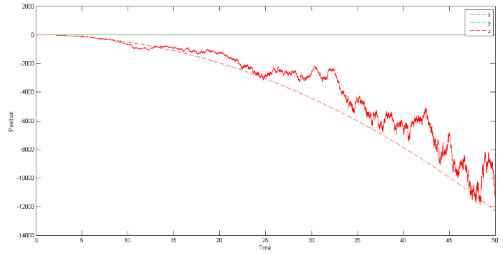

Figure 8. Linear Drift Associated with Velocity Updates

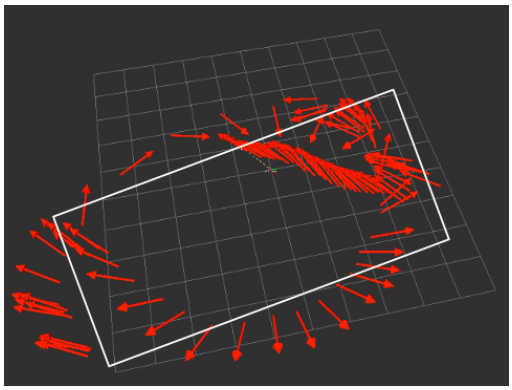

Figure 9. Odometry Information vs. Ground Truth

Figure 8 shows a simulation of the linear drift in state estimation when only using velocity estimates.

When placed into the EKF, the velocity estimates are converted to position estimates weighted by the covariance of the velocity estimates.

Figure 9 shows odometry estimates where the tip of the arrow is the position and the arrow is the direction of the velocity.

Optical flow and visual odometry can provide stable measurements at around 200 Hz effectively while in a steady configuration. The estimates are poorer in dynamic situations

### 6.2. Localization of April Tags

April Tag updates using the MIT package [14] can be gathered at around 8 Hz. This number goes down, however, as I run a filter that removes updates that are spurious. This lowers the frequency of the April Tag updates. It does, however improve the effectiveness of these updates.

Figure 10 shows the filtered vs. noisy updates.

These updates provide accurate direct state readings at low ( 1 Hz) frequencies.

### 6.3. SLAM and Obstacle Detection

As mentioned in another section, I used OpenNI to manipulate the point cloud from the RGB-D sensor, RTAB-


Figure 10. Filtered vs. Noisy Updates

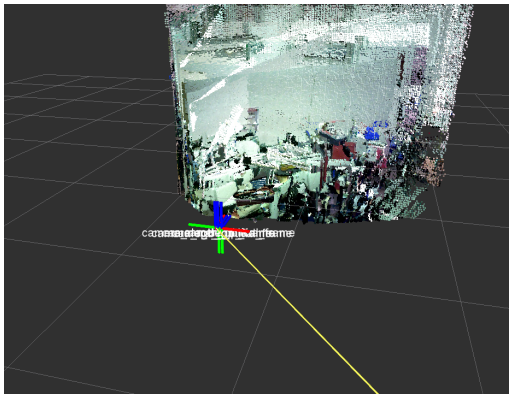

Figure 11. SLAM and Odometry Visualization

Map for SLAM, and ROS Navigation Stack for obstacle Detection.

In order to avoid the drift associated with integrating velocity measurement noise, I attempted to use SLAM in order to provide direct pose estimates. Figure 11 shows the output from SLAM with the estimate of the quad's odometry.

This information is quite effective but slow. I could only run this on the Minnowboard Max at around 0.5 Hz. This is completely ineffective for a dynamic quadcopter.

The next thing I tried was to just recognize obstacles. The mapping only ran at 0.5 Hz, but OpenNI could run at around 20 Hz with manipulating the point clouds. I decided to directly use the point cloud data to create a non-persistent costmap with the data.

Figure 12 shows the output of the costmap showing obstacles.

This costmap generation is very efficient only using ray

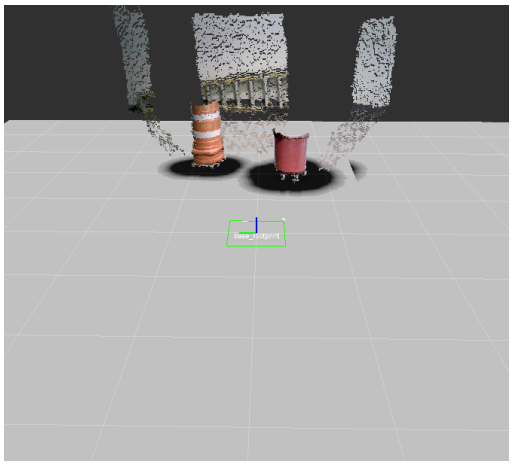

Figure 12. SLAM and Odometry Visualization

tracing to fill and clear the costmap. The costmap projects the 3D point cloud to a 2D occupancy grid. This grid is highly effective to plan through. By getting 20 Hz updates, I was able to consistently fill and clear the costmap in real-time and in flight.

## 7. Conclusion

After evaluating the effectiveness of several visual methods for state estimation, mapping, and obstacle detection, I have come to the conclusion about the effectiveness and practicality of these systems.

The main conclusion that I can draw is that SLAM is too computationally intensive to run on a low-powered board. The 0.5 Hz update rate is much too slow for a quadcopter environment. This could have been mitigated by cutting out the mapping aspect and only focusing on the visual odometry from the camera. I did not pursue this, because I was already using optical flow for visual velocity estimates.

The next conclusion I came to is that optical flow is only as good as the features you give it. By engineering the environment and floor, I was able to get accurate state estimation over the features. Whenever the features drastically changed, however, the system was prone to losing its state.

My conclusion is that in a low-powered/low-weight system, one can use local methods usch as optical flow and ray tracing in order to estimate ones state and detect obstacles.

By combining these vision systems, one can create a system around these while planning locally. For most applications, local planning is all that is needed for indoor environments. Furthermore, data could be collected while flying locally in order to perform mapping offline. If global localization is needed, a fixed beacon such as an April Tag can be used.

I conclude that local state estimation and obstacle detection methods are efficient and effective enough in a moder-

ately controlled environment. This provides a more useful system than mo-cap or direct visual ID.

## 8. Future Work

In the future, I plan on using this vision system to plan around static obstacles.

## References


[1] Geekbench. https://browser.primatelabs.com/android-benchmarks.

[2] Lsd slam. http://vision.in.tum.de/research/vslam/lsdslam.

[3] Mavlink ros wrapper. http://wiki.ros.org/mavros.

[4] The minnowboard.org foundation, minnowboard max. http://wiki.minnowboard.org/MinnowBoard_MAX.

[5] Odroid xu4. http://www.hardkernel.com/main/products/prdt_info.php?g_code=G143452239825.

[6] Pixhawk flight controller. https://pixhawk.org/choice .

[7] Ros navigation stack. http://wiki.ros.org/navigation.

[8] Wikipedia ekf. https://en.wikipedia.org/wiki/Extended_Kalman_filter.

[9] P. 3. Eye. Downward facing camera used for April Tag ID https://en.wikipedia.org/wiki/PlayStation_Eye.

[10] 3DR. Iris+. Quadcopter used for project https://3dr.com/kb/iris/.

[11] Asus. Xtion pro live. RGB-D Sensor https://www.asus.com/us/3D-Sensor/Xtion_PRO_LIVE/,.

[12] P. T. Dominik Honegger, Lorenz Meier and M. Pollefeys. An open source and open hardware embedded metric optical flow cmos camera for indoor and outdoor applications. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, May 2013.

[13] M. Labb. Online global loop closure detection for large-scale multi-session graph-based slam. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, Sept 2014.

[14] E. Olson. AprilTag: A robust and flexible visual fiducial system. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3400–3407. IEEE, May 2011.

[15] OpenNI. Ros wrapper for openni. Manipulates 3D Point Cloud http://wiki.ros.org/openni_launch.

[16] N. M. Shaojie Shen, Yash Mulgaonkar and V. Kumar. Vision-based state estimation and trajectory control towards high-speed flight with a quadrotor. In *Robotics: Science and Systems (RSS)*. RSS, June 2013.