Improving DecaWave's Performance in V207

Joshua Springer

I. Introduction

This is a LaTeXtemplate document.[1]

The DecaWave positioning system in V207 offers a solution for ultrawide band radio-based positioning, via round-trip time of arrival distance estimation from a given tag to several anchors whose positions are known. It provides position estimation in 3 dimensions, but with low precision. Initial usage of the system to play the video game Pong revealed difficulty in maintaining a consistent position, which is problematic when trying to block the Pong ball from hitting the wall. this project attempts to improve the DecaWave system's precision by using multiple filtering methods: a Kalman filter, a particle filter, and a heading-aware filter that assumes the vehicle whose position is being estimated can only move forward/backward and rotate in the yaw dimension, and attempts to ignore perceived side-to-side movement. The data comes from 3 test cases, wherein a drone with one of the DecaWave tags, an IMU, a depth camera, and an optical flow sensor contribute to its positioning accuracy. Each of the proposed systems is a dead-reckoning system, meaning that it is subject to drift, and therefore depends on the DecaWave system's "global" nature to consistently remove the drift. The systems are designed purely to augment the DecaWave system, not to serve as a positioning system on their own.

II. Methods & Exploratory Tests

I have tried various methods for estimating the position of the drone system – particularly with the goal of determining the velocity of the drone in order to remove the bouncing that the DecaWave system exhibits. I initially tested these sensors in isolation to determine which is the most viable, and therefore where I should most apply my efforts. It is especially important to note that indoor environments are quite variable, and not all sensors are applicable to all environments. In this section I explain the exploratory tests to determine which sensors are viable in V207, and I discuss the possible factors influencing their behavior.

A. Optical Flow

The first sensor in the experiments is an ArkFlow optical flow sensor (See Figure 1), which determines x/y-velocity using pixel velocity in a downward-facing camera. This sensor is used in GPS-denied drone flight to extract velocity information from the relative movement of the terrain beneath the drone.

Fig. 1. The ArkFlow sensor mounted to the bottom of the drone.

Initial tests show that the sensor is able to perceive motion from a pixel environment with some variance. Its raw data is also in some abstract form that must be processed by the flight controller unit (FCU) before use. Therefore, the velocity estimate from this sensor came from the FCU directly. Ultimately, it also has difficulty determining its velocity from the very self-similar floor of V207. Figure 2 and 3 show the noisy velocity and position estimates respectively. The velocity signal shows a sharp discontinuity in $t \in [65, 74]$, and a large negative spike at $t = 50$, both of which do not correspond to the reality of the exploratory test. These are made clearer by the position estimate in Figure 3, which essentially shows teleportation. It is possible that this is not only due to the ArkFlow sensor, but also to the FCU's data fusion methods. However, given time limitations, this method was not explored further.

B. Depth Flow

A D455 stereo depth camera provides a relatively clean 3-dimensional view of the scene in front of the drone. Let $d_{r,i}$ represent the i^{th} *raw* 640x480 depth image generated from the D455 camera at index i. This is referred to as *raw* because its data is scalar multiples of some `depth_scale` s that is included as metadata, according to standard practices in the `pyrealsense2` library that connects Python to the sensor itself. We convert a *raw* depth image $d_{r,i}$ to a depth image d_i

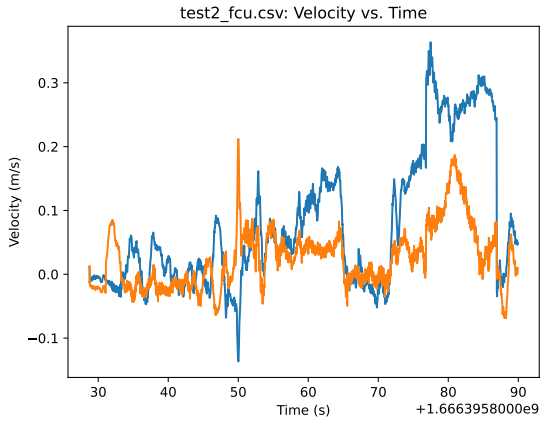

Fig. 2. The velocity estimate from the FCU, fusing the ArkFlow data with IMU. Blue and orange correspond to the x and y dimensions respectively.

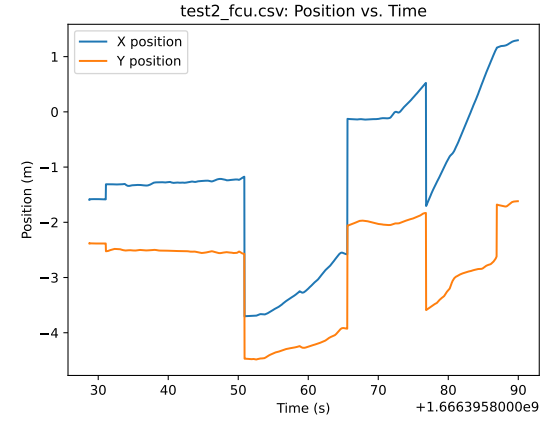

Fig. 3. The position estimate from the FCU, fusing the ArkFlow data with IMU. Blue and orange correspond to the x and y dimensions respectively.

simply by multiplying the image by the depth scale s after converting it to a Numpy array. So

$$d_i = d_{r,i} \cdot s$$

where d_i is a *depth image*, $d_{r,i}$ is a *raw depth image*, and s is a *depth scale*.

Then, the velocity at index i – v_i – is as such:

$$v_i = \frac{d_i - d_{i-1}}{\Delta t}$$

where $t = t_i - t_{i-1}$ is the difference in timestamps between d_i and d_{i-1} At this point, the pixel position of the data is no longer important, so I reshape the 2D array into a 1D array in order to operate on it more easily.

The edges around objects in the depth image are somewhat noisy and discontinuous, as shown in Figure 4,

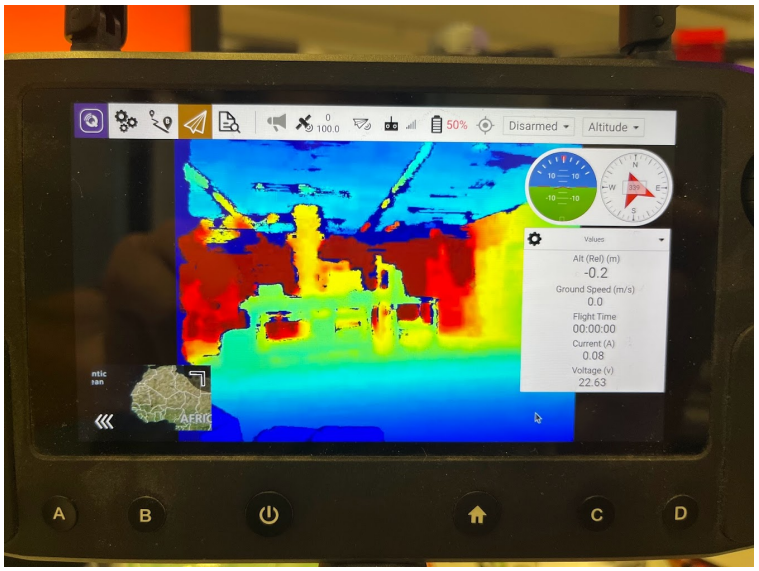

Fig. 4. A sample image from the D455.

and this can lead to high perceived pixel velocity in the depth dimension. To remove this particular type of noise, I simply remove all velocity estimates above a threshold of 1 m/s magnitude, decreasing the size of the velocity array v_i. The whole velocity vector can then be aggregated into a single measurement using a simple arithmetic mean.

To test the overall performance, I recorded the aggregated velocity measurement over time and plotted it for easy visualization.

For the first test, shown in Figure 5, I simply left the drone still and recorded the velocity over time. The plot shows relatively little noise – less than 5 cm/s. The second test, shown in Figure 6 correctly depicts discrete time segments of backwards and forwards motion and stillness. This method of forward/backward velocity estimation appeared promising enough to use in later experiments.

C. Heading Estimation

The D455 has an inertial measurement unit with a gyroscope and accelerometer, for estimation of angular velocity and orientation to the gravity vector.

References

[1] I.P. Freely. A small paper. *The journal of small papers*, -1, 1997. to appear.

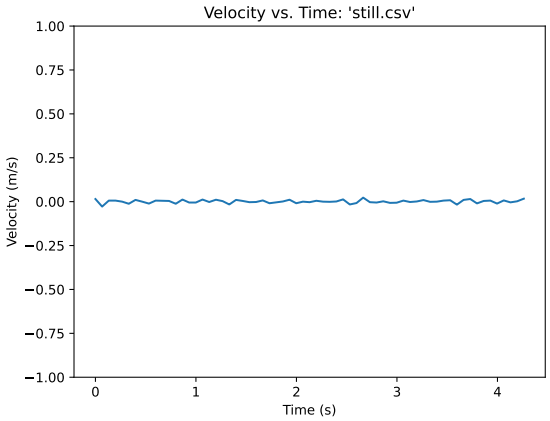

Fig. 5. The perceived velocity while the drone was sitting still.

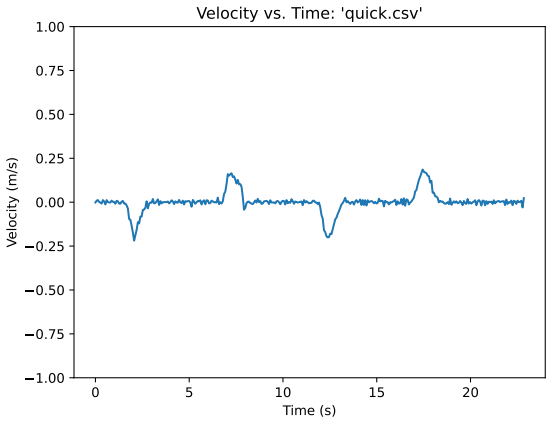

Fig. 6. The perceived velocity while I moved the drone backwards and forwards quickly twice, leaving it still for a short time between movements.

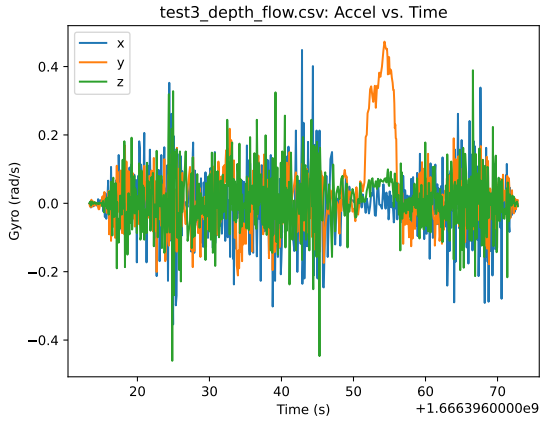

Fig. 7. A test of the gyroscope, showing a noise in x and z dimension, and a large, correct spike in the y dimension which corresponds to a rotation of the drone in the yaw dimension.