Improving DecaWave's Performance in V207

Joshua Springer

I. Introduction

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.

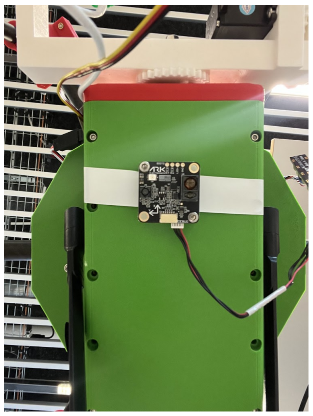
Fig. 1: The ArkFlow sensor mounted to the bottom of the drone.

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 (See Figure 9). 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 simply by multiplying the image by the depth scale s

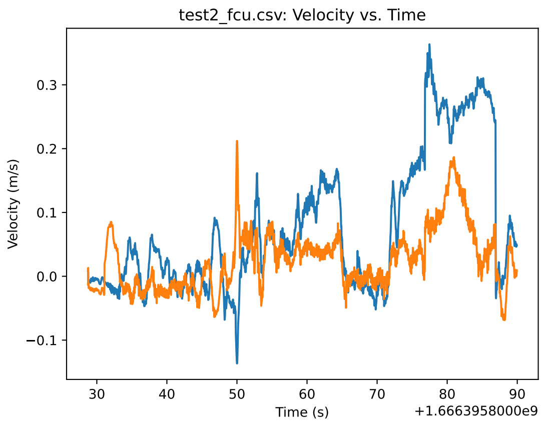

Fig. 2: The velocity estimate from the FCU, fusing the ArkFlow data with IMU. Blue and orange correspond to the x and y dimensions respectively.

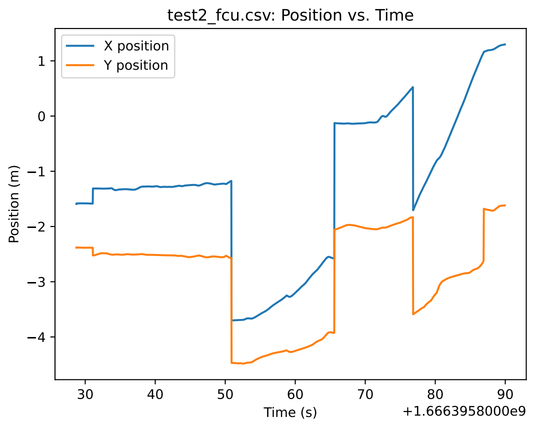

Fig. 3: The position estimate from the FCU, fusing the ArkFlow data with IMU. Blue and orange correspond to the x and y dimensions respectively.

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, and this can lead to

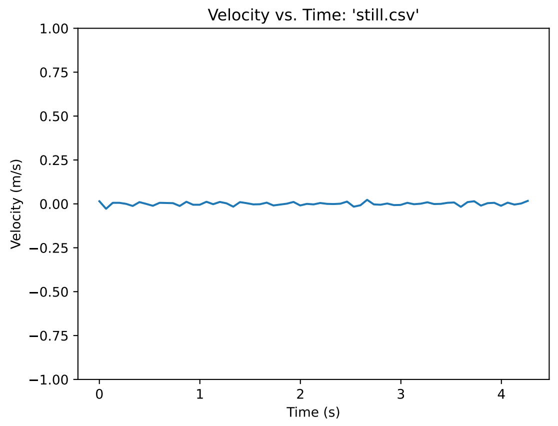

Fig. 4: The perceived velocity while the drone was sitting still.

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 4, 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 5 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. Given the setup of the testing platform (a pallet jack) one can make the warranted assumption that the platform must be moving forward/backward or rotating in the yaw dimension only. Moreover, during testing this is guaranteed by simply constraining the platform to simple movements that are not antagonistic to the system. With this assumption, we can use the heading of the drone as a justification to ignore erroneous motion if it corresponds to the impossible side-to-side motion of the cart.

I have made a Kalman filter to estimate the heading of the drone relative to its starting heading. Its properties are as below:

Initial state:

$$X_0 = \begin{pmatrix} 0 & 0 \end{pmatrix}$$

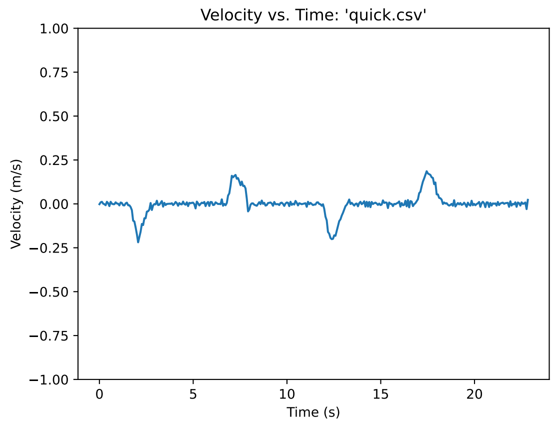

Fig. 5: The perceived velocity while I moved the drone backwards and forwards quickly twice, leaving it still for a short time between movements.

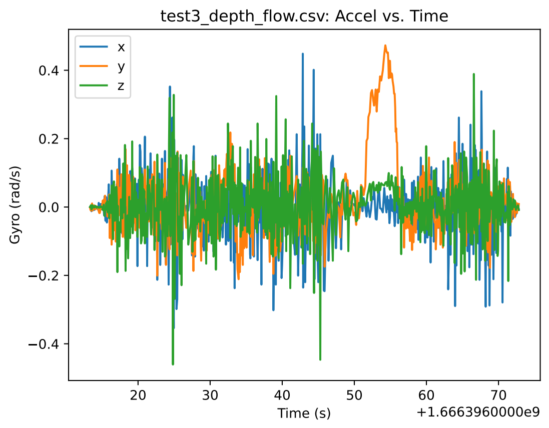

Fig. 6: A test of the gyroscope, showing a noise in x and z dimension, and a large, correct spike in the y dimension which corresponds to a rotation of the drone in the yaw dimension.

State transition matrix, with a camera framerate of 15 Hz $\rightarrow dt = 0.067s$:

$$F = \begin{pmatrix} 1 & \frac{1s}{15} \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0.067 \\ 0 & 1 \end{pmatrix}$$

Measurement model (the gyro measures only speed, and we integrate to get position):

$$H = \begin{pmatrix} 0 & 1 \end{pmatrix}$$

Covariance matrix:

$$F = \begin{pmatrix} 500 & 0 \\ 0 & 500 \end{pmatrix}$$

Measurement noise:

$$R = \begin{pmatrix} 1000 \end{pmatrix}$$

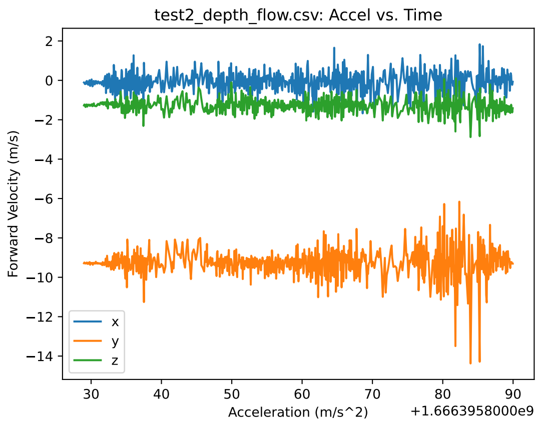

Fig. 7: Example accelerometer data.

D. Depth Camera Accelerometer

Figure 7 shows the raw accelerometer data from the D455, which seems to actually be clean, usable data. However, all it can really show is that the drone was *relatively* upright for the duration of the tests, with the Y axis reading roughly $-9.8\frac{m}{s^2}$, and the X axis hovering around $0\frac{m}{s^2}$. The Z axis is slightly negative as a result of the camera (but not the drone) being tilted slightly downward on its gimbal throughout the tests.

E. Filtering Methods

I filter the raw DecaWave data with both a Kalman filter and a sequential importance filter, to attempt to increase its precision. Each component of the position x, y uses a single, 1-dimensional filter. The Kalman filter definition is as below, and differs from the filter in Section II-C in that its measurement model specifies that the measurements corresponds to the position, rather than the velocity, and they are otherwise similar except that one is working in terms of angles and the other in terms of Cartesian coordinates.

Initial state:

$$X_0 = \begin{pmatrix} 0 & 0 \end{pmatrix}$$

State transition matrix, with a camera framerate of 15 Hz $\rightarrow dt = 0.067s$:

$$F = \begin{pmatrix} 1 & d_t \\ 0 & 1 \end{pmatrix}$$

Measurement model (the gyro measures only speed, and we integrate to get position):

$$H = \begin{pmatrix} 1 & 0 \end{pmatrix}$$

Covariance matrix:

$$F = \begin{pmatrix} 5000 & 0 \\ 0 & 5000 \end{pmatrix}$$

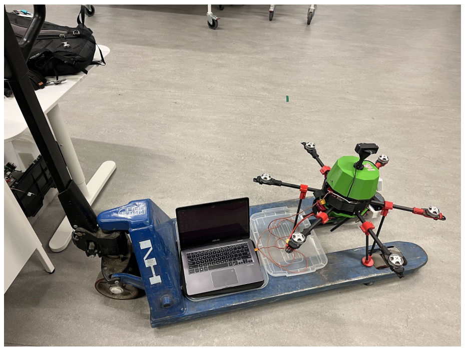
Fig. 8: The testing platform, in all its glory.

Measurement noise:

$$R = \begin{pmatrix} 500 \end{pmatrix}$$

The sequential importance filter is implemented exactly as in the file `sis.py` provided with the course, but with $n = 500$ particles instead of 250.

F. Platform & Data Collection

Figure 8 shows the data collection platform. The pallet jack exposes the floor to the optical flow sensor while allowing the drone to remain level and unobstructed.

I have marked the landmarks on the floor of V207, beginning 1.8m from the wall going into the department of computer science. They appear in 15x1.0m intervals, marked with pieces of electrical tape on the floor. The distances are measured with a 3.0m, class 2 measuring tape (although manually timestamping the landmarks in real time likely introduces more error than the small one of the measuring tape). At the end of the 15 meter set of points, the track turns towards the windows 90 degrees clockwise, and continues for 5 meters. The 90 degree line was marked using a laser joining to the original line using a square, as shown in Figure 9.

Error rates for the systems are determined using ground truth markers that are manually timestamped as the drone is pushed along its path. The Cartesian distance between the ground truth and the estimated position is the error at each ground truth point, considering only components x, y.

III. Results

A. Test Case Descriptions & Raw DecaWave Data

The raw DecaWave data is shown in Figure **??**. During tests 1 and 3, the data collection platform moved from $(x, y) \approx (5, 1.8)$ to $(x, y) \approx (12.5, 15)$ along an L-shaped path with a single clockwise $90°$ turn. During test 2, the drone followed the same path in the opposite direction. This most positional variance appears in the top left of the path, where the path is both near to a wall and to the wall of the convex hull bounding the anchors

Fig. 9: Marking the adjoining line, and showing the self-similarity of the floor.

for the DecaWave system. Test 1 is representative of the other 2 tests, and they have therefore been left for conciseness.

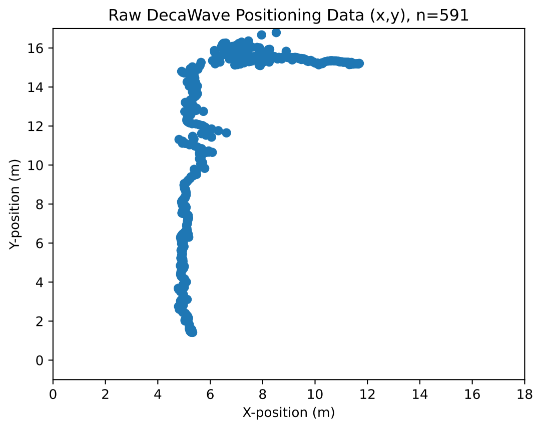

Fig. 10: Test 1 – a representative example.

B. Heading Estimation

Figures 11, 12, and 11 show the angular velocity measured by the gyroscope, as well as the smoothed angular velocity and estimated angular position as measured by the Kalman Filter. Figure 11 correctly shows a clockwise turn of $90° \approx 1.578$ rad. Figure 12 correctly shows the opposite – a counter-clockwise turn of $-90° \approx -1.578$ rad. Figure 13 shows a seemingly

Test	μ_{d_t}	σ_{d_t}
Test 1	0.0704	0.0209
Test 2	0.0822	0.0405
Test 3	0.0776	0.0360

TABLE I: Statistical values for actual d_t in each test case.

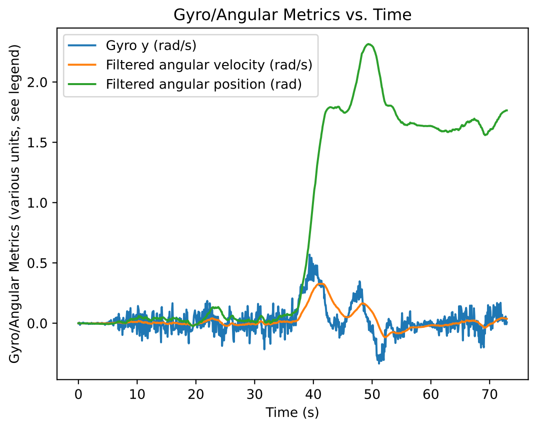

Fig. 11: Heading for test 1, with raw gyro data (angular velocity), and angular velocity and position as estimated by the Kalman filter.

erroneous clockwise turn of about $45° \approx 0.79$ rad, which needs investigation.

The `filterpy` implementation of the Kalman filter behaves badly (giving no output), if the value for d_t is adjusted at each timestep in F, and I therefore had to use only a constant 0.067s. Thinking that this could be an explanation for the inaccuracy in Figure 13, I analyzed the *actual* difference in time between frames for mean and standard deviation, as in Table I. However, this shows that Test 3 actually has medium values for both mean and stddev. The fact that Tests 2 and 3 have the most different values for mean and stddev mean that inconsistency in d_t should be more likely to major differences in performance.

C. Depth Flow Velocity Estimation

Figure 14 shows the raw depth flow data (forward velocity only). Unfortunately it seems that the changing scenery has decreased the usefulness of this data. The velocity estimate depends on pixels mostly belonging to the same objects over time, which was the case during the initial tests, where the drone was only moving forward and backward over a small range (with no rotation) facing a wall that was far away. In these test cases, the drone had nearby obstacles close to its left and right sides, and as it moved past them, they had a high pixel velocity in the x-direction, which is one factor in the noisy, unreliable data.

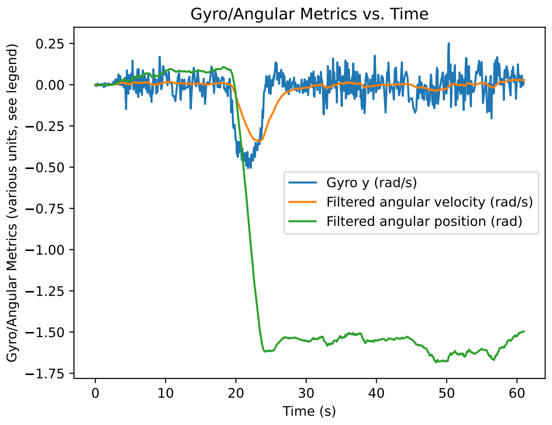

Fig. 12: Heading for test 2, with raw gyro data (angular velocity), and angular velocity and position as estimated by the Kalman filter.

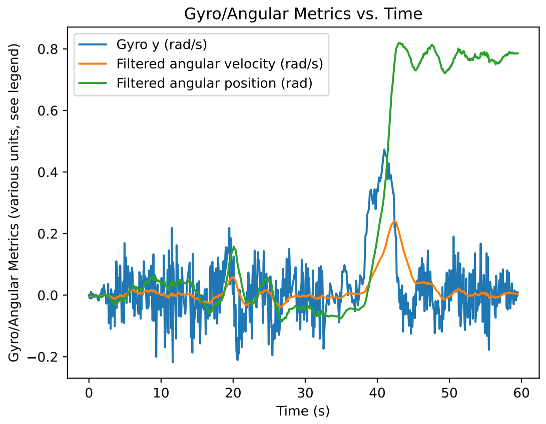

Fig. 13: Heading for test 3, with raw gyro data (angular velocity), and angular velocity and position as estimated by the Kalman filter.

D. Filtering Methods

I have plotted the raw, filtered, and ground truth values from the DecaWave experiments in Figure 15. They are organized by color, such that the most purple points are the oldest and occurred at the start of the experiment, and the most yellow points are the newest and occurred at the end of the experiment. The small points (sometimes obscured) represent the raw data, the larger points represent the filtered data, and the Xs represent the ground truth. Each of the systems' begin at an assumed position of $(0, 0)$, leading to the curved, extraneous purple tail in all of the test cases.

Figure 16 shows the performance of the 2-dimensional particle filter on the raw DecaWave data.

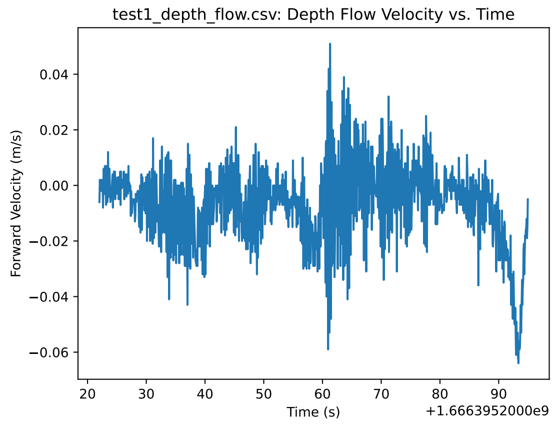

Fig. 14: The raw depth flow data visualized for Test Case 1 – a representative example.

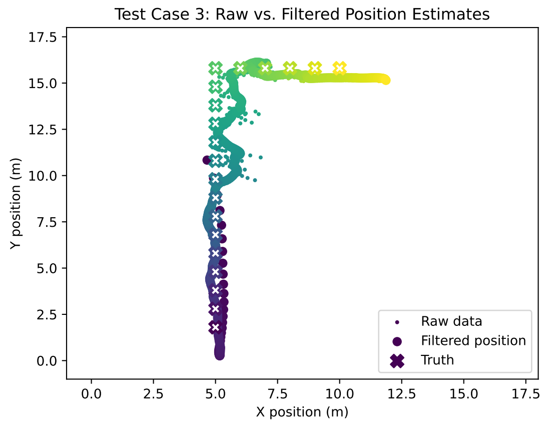

Fig. 15: The raw, filtered, and ground truth values for the DecaWave system in Test Case 3.

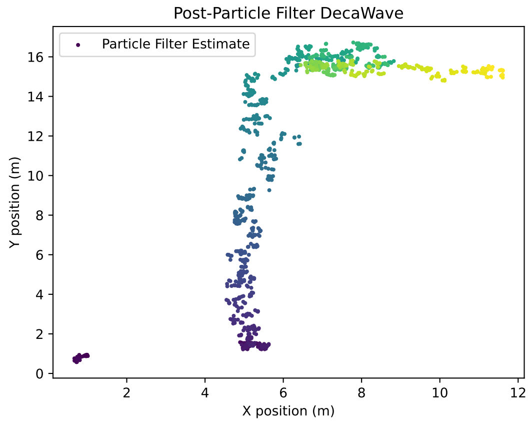

Fig. 16: Particle filter visualization for Test Case 1

	DecaWave	DecaWave + Kalman Filter	DecaWave + Particle Filter
Test Case 1	0.6000	0.6562	0.4080
Test Case 2	2.4653	2.3597	1.4205
Test Case 3	0.7304	0.7040	0.9668

TABLE II: The average errors (m) of each system in each test case.

E. *Error Rates*

Table II shows the error rates for each of the working systems. The raw DecaWave did not have the best (lowest) error in any test case, no system outperformed both competitors in all cases. The particle filter gives the lowest error in test case 1, and the Kalman filter gives the lowest error in the other test cases.

IV. Discussion

The most tried and true methods – filtering – were those that were somewhat successful in this project. They slightly decreased the error of the raw DecaWave system, although they did add in artifacts such as the purple "tail" in Figure 15, which likely occur as a result of the filter estimate having to converge to the correct value after starting at an initial value of $(x, y) = (0, 0)$. Kalman filtering also predicted near-correct values for the system's heading using dead reckoning in test cases 1 and 2, though it had some difficulty in test case 3 for a hitherto undiagnosed reason.

Some peripheral sensors that did not work in this setting could still be applied to future projects. While the depth sensor works to accurately estimate the drone's forward velocity in non-antagonistic conditions, it has trouble in scenes where there are a lot of obstacles moving close to the camera, and when the yaw orientation of the camera changes a lot, because it assumes that each pixel labels the same object from frame to frame. It could be made smarter in order to fix this issue. The optical flow sensor has been shown to work well on surfaces that are less self-similar than the floor in V207, so this could also be applied to the position estimate if the floor had a more varied visual appearance.

Finally, the heading-aware position estimator was not implementable given time limitations and other duties, but this is still a conceptually viable system.

V. Conclusions & Future Work

Kalman filtering and particle filtering/sequential importance filtering can improve the performance of the DecaWave system in V207. Optical flow in V207 is difficult because the floor is too self-similar. Depth flow velocity estimation is possible but needs to be more sophisticated than implemented here in order to handle obstacles that move in the frame. Future work includes making the environment more suitable for optical flow, making the depth flow velocity estimation more sophisticated, and implementing the heading-aware Kalman filtering.