embeddedworld2022
Exhibition&Conference
... it's a smarter world

# Real Time Motion Tracking for Augmented Reality with TOF Camera and Vulkan Rendering

Marcel Wegmann
Institute of Embedded Systems
ZHAW School of Engineering
Winterthur, Switzerland

Prof. Dr. Matthias Rosenthal
Institute of Embedded Systems
ZHAW School of Engineering
Winterthur, Switzerland

***Abstract*— Augmented Reality is the concept of enhancing the real world with virtual objects or information with projections into a viewfinder or through specialized goggles. Simpler forms of Augmented Reality – like a heads-up display in a car – do not need to estimate the camera's motion, an object, or the user. However, more elaborate implementations of Augmented Reality need to track things and, more importantly, the camera's movement itself. The applications in which Augmented Reality could be leveraged range from social interaction over pedestrian navigation to various use cases in different professions. Multiple companies already have shown closed source or custom-tailored programming interfaces, either running on smartphones or shipped with industry-targeted goggles. The tracking of real-world objects or surfaces is possible with the provided interfaces, but the algorithms behind the different functions are corporate secrets. This paper describes an approach for an end-to-end pipeline in a prototype of an Augmented Reality platform without using commercial interfaces. A time-of-flight camera provides a depth-image that allows reconstruction of the recorded scene as a cloud of SIFT features. Frame-by-frame analysis of the point cloud estimates the camera's motion by highly parallel processing and a three-dimensional extension of the RANSAC algorithm. An accelerometer and a gyroscope provide additional data, fused with a Kalman filter to improve the motion estimation. A regular color camera acts as a viewfinder, and Vulkan renders the result to a monitor. Enhancing the matching quality of SIFT features between consecutive frames of a time-of-flight camera using a three-dimensional RANSAC algorithm led to over two times as many correct matches.**

***Keywords—AR, augmented reality, time-of-flight camera, tof, RANSAC, SIFT, Kalman Filter***

## I. Introduction

Augmented Reality - or AR – is the concept of projecting virtual objects into the real world. Phone screens, tablet computers, and specialized goggles render virtual objects over the camera image or display them on translucent screens. A form of Augmented Reality is a heads-up display, for example, in cars to project the current speed and navigation information to the windshield or in airplanes for comprehensive avionic information.

In contrast, Virtual Reality – or VR – limits itself to entirely virtual worlds, into which the user dives. While Virtual Reality hardware is already available through off-the-shelf goggles, which lets users meet other people and play games in virtual worlds, Augmented Reality is mainly limited to smartphone applications. Currently, it lacks specialized off-the-shelf hardware, other than niche products specialized for specific industries.

Augmented Reality faces numerous technical challenges. Projecting a virtual object – for example, a flowerpot – into the real world requires the system to recognize a table and find an unoccupied location. Apart from placing decoration or furniture, a pair of Augmented Reality goggles could project helpful information into the air. A mechanic could have virtual schematics or instructions floating beside his work, while another virtual monitor displays a video phone call with the customer. Hand detection and gesture control would enable interaction with virtual objects. Another example could be pedestrian navigation, projecting arrows to the street.

AR goggles need to react in real-time to any motion of the user's head. Any latency would break immersion as virtual objects lose the connection with their anchor point in the real world. A flowerpot would jump on the table, and arrows on the street would start to float and collide with walls. Fast and reliable motion tracking of the system itself is vital for avoiding visual glitches.

Time-of-Flight (TOF) cameras provide depth information on its image, by measuring the distance on each camera pixel. The information provided by a TOF camera allows reconstructing the scene and estimating the motion of the camera between two camera frames.

## II. Concept

For demonstration of the developed motion estimation algorithm, a full AR pipeline was created, containing video capturing, stream-processing, sensor-fusion, and 3D rendering, implemented on a Nvidia Jetson Xavier AGX platform. The stream-processing reconstructs the camera image into a 3D cloud from which the rigid motion from the prior frame is estimated. The sensor-fusion mixes the estimated motion from the TOF camera with data of an IMU for enhancing the accuracy.

## III. Motion Estimation from TOF image

The TOF camera captures both a black-and-white image and a corresponding depth map of the same frame. To gain linearity, lens- and radial correction are required and applied to the image and the depth map. For estimating the motion between concurrent camera frames, individual feature points in both B/W-images need to be matched. This study utilizes the SIFT algorithm for feature extraction, but any other feature-point extractor should work, if a method for brute-force matching generates a significant subset of correct feature pairs.

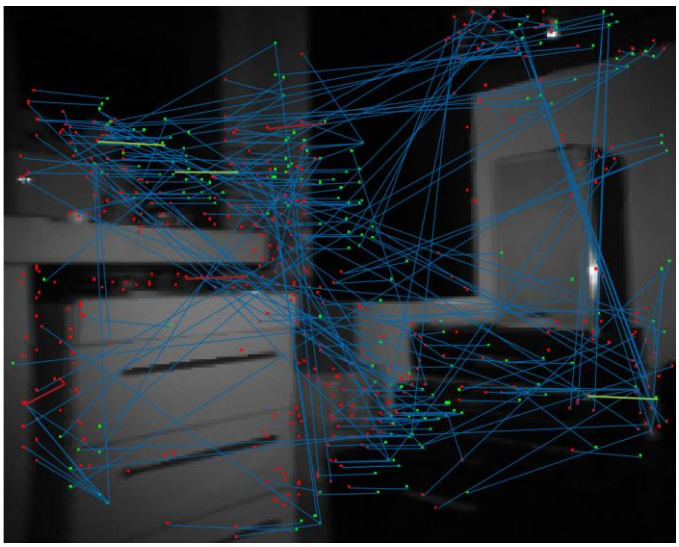

Fig. 1. Sample B/W-Image with drawn feature points, part of of a sequence in which the camera got rotated.
Green dots: Frame k-1
Red dots: Frame k
Lines: Brute-Force matches

As visible in Fig. 1, the brute-force matcher generated both correct and false matches, that need to be identified for reliable processing. Correct matches follow a rigid motion in 3D, which is a combination of rotation and translation. Identifying the correct matches also allows improving the matching quality of the bad matches, as gained information helps rematching the clouds.

### A. ToF depthmap to 3D transformation

Mapping the image feature point clouds into 3D space by utilizing the TOF depth map, allows analysis of the matches regarding the rigid motion. A TOF depth map contains radial data, that needs to be rectified first, either by knowing the angle for each pixel or by measuring the cosine individually for each pixel by a measurement on a flat surface.

After rectification, the de-projection of 2D feature points into a 3D map is linear, based on a distance calibration for the depth map and a focal length calibration on the TOF camera's optics.

### B. Rigid Motion from three points

In three-dimensional space, at least three matched point-pairs are required to determine the applied rotation and translation between two consecutive point clouds $P_k$ and $P_{k-1}$. The following method also allows calculating the rigid motion on more than three matches. The calculation is a multi-step algorithm described in the following, with $\overrightarrow{p_{i,k}}$ being the $i$-th point of the $k$-th cloud and n being the number of matched point-pairs.

1) *Calculate center points $\vec{c}$ of both point clouds:*

$$\overrightarrow{c_k} = \frac{\sum_{i=1}^{n} \overrightarrow{p_{i,k}}}{n} \qquad \overrightarrow{c_{k-1}} = \frac{\sum_{i=1}^{n} \overrightarrow{p_{i,k-1}}}{n}$$

2) *Calculate centered point clouds $Q_k$ and $Q_{k-1}$:*

$$\overrightarrow{q_{i,k}} = \overrightarrow{p_{i,k}} - \overrightarrow{c_k} \;;\; \overrightarrow{q_{i,k-1}} = \overrightarrow{p_{i,k-1}} - \overrightarrow{c_{k-1}}$$
$$i = 1, 2, 3, \dots, n$$

3) *Compute the covariance matrix of centered point clouds:*

$$S = \begin{bmatrix} \cdot & \cdot & \cdot & \\ q_{1,k} & q_{2,k} & q_{3,k} & \cdots \\ \cdot & \cdot & \cdot & \end{bmatrix} \cdot \begin{bmatrix} \cdot & \cdot & \cdot & \\ q_{1,k-1} & q_{2,k-1} & q_{3,k-1} & \cdots \\ \cdot & \cdot & \cdot & \end{bmatrix}^T$$

4) *Compute the singular value decomposition (SVD) of S:*

$$S = U\Sigma V^T$$

5) *Calculate the rotation matrix R:*

$$R = V \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & det(VU^T) \end{pmatrix} U^T$$

The term $det(VU^T)$ in the intermediate matrix corrects the result, if the SVD led to a reflection instead of a rotation. This would be numerically sound but would not reflect the real behavior.

6) *Compute the translation:*

$$\vec{t} = \overrightarrow{c_k} - R\overrightarrow{c_{k-1}}$$

The mathematical proof of this method can be found in [3].

### C. 3D RANSAC algorithm

Random Sample Consensus (RANSAC) is a well-known algorithm in 2D panoramic stitching to find good matches and allows correcting bad matches with new information. The standard approach uses regular images without depth map and allows finding the homology between two projections. Having brute-force matched features with 3D coordinates, and knowing that good matches fulfill a rigid motion, allows extending the RANSAC algorithm.

As three points are required for calculating a rigid motion in 3D space, to each point-pair two additional point-pairs get randomly assigned, ensuring that each point-pair gets checked at least once. On each group of three point-pairs, the SVD algorithm gets performed to find the rigid motion individually for each point-pair trio.

Each point-pair trio results in an individual rigid motion, that then gets tested on all brute-force matches. If a tested brute-force match fulfills the rigid motion calculated from the point-trio, it gets assigned to a list. The result with the most matches in the list wins the competition with the list containing the subset of correct matches. An additional calculation of the rigid motion using all the correct matches at once allows improving the estimated rigid motion from the brute-force matcher.

A significant portion of data points, namely all false matches, would be left out, if no further action would be taken. The rigid motion from the random sample consensus allows discarding all the brute-force matches and re-matching the data-points based on their position in 3D space. Due to noise and

changes in perspective, it is not guaranteed that every point can be assigned to a suitable counterpart. The rematched set of point-pairs allows further improvement of the rigid motion, by performing the calculation on this larger set of matches. The result of this final step is the optimal rigid motion to extract from this set of features and used as the output of the TOF motion extractor.

## IV. Sensor fusion with IMU

In any augmented reality platform, multiple sources for rotation and position estimation need to be combined, allowing more accurate readings. As a secondary data source, a low cost 6-axis IMU is used. One possibility to perform sensor fusion for motion estimation is the use of a Kalman filter. A Kalman filter is a model-based predictor-corrector algorithm, whose model describes the relations between multiple inputs and outputs. The used notation for the Kalman filter is as follows:

Prediction:

$$\vec{x}_{k|k-1} = F_{k-1}\vec{x}_{k-1}$$

$$P_{k|k-1} = F_{k-1}P_{k-1}F_{k-1}^T + Q_{k-1}$$

Correction:

$$K_k = P_{k|k-1}H_k^T(H_kP_{k|k-1}H_k^T + R_k)^{-1}$$

$$\vec{x}_k = \vec{x}_{k|k-1} + K_k(\vec{z}_k - H_k\vec{x}_{k|k-1})$$

$$P_k = (I - K_kH_k)P_{k|k-1}$$

In which $\vec{x}$ is the system-state vector, containing positions, velocities, and accelerations for all three dimensions as well as the orientation and the rotation speed. F is the state-transition-model, translating the prior state to the prediction. $P$ is the estimate covariance, $Q$ the covariance of the process noise, $K$ the Kalman gain, $H$ the observation model and $R$ the covariance of the observation noise. The vector $\vec{z}$ contains the sensor data of the current iteration.

Rotational motion information from the gyroscope and the TOF algorithm get transformed into quaternions, to only have four values and numerical stability in the Kalman filter's system-state vector. The required Hamilton Product got implemented in matrix form for the state-transition-model $F$.

$$\overrightarrow{x_{ori,k|k-1}} = F_{ori} \cdot \overrightarrow{x_{ori,k}} =$$

$$\begin{pmatrix} r_a \\ r_b \\ r_c \\ r_d \end{pmatrix}_{k|k-1} = \begin{bmatrix} \dot{r_a} & -\dot{r_b} & -\dot{r_c} & -\dot{r_d} \\ \dot{r_b} & \dot{r_a} & \dot{r_d} & -\dot{r_c} \\ \dot{r_c} & -\dot{r_d} & \dot{r_a} & \dot{r_b} \\ \dot{r_d} & \dot{r_c} & -\dot{r_b} & \dot{r_a} \end{bmatrix} \cdot \begin{pmatrix} r_a \\ r_b \\ r_c \\ r_d \end{pmatrix}_{k-1}$$

As both sensory inputs, gyroscope and TOF algorithm, provide the rotation speed, the correction step is duplicated, and the two inputs are chained one after another. For completeness, the rotation speed is also part of the system-state vector and translated in the state-transition-model with a 3x3 identity matrix.

For translation, the standard procedure is utilized for each dimension separately, estimating the position, velocity, and acceleration. This step is performed for each dimension once, as motion in one direction does not affect the motion in the other direction.

$$\overrightarrow{x_{trans,k|k-1}} = F_{trans} \cdot \overrightarrow{x_{trans,k}} =$$

$$\begin{pmatrix} p \\ v \\ a \end{pmatrix}_{k|k-1|} = \begin{bmatrix} 1 & \Delta t & \frac{\Delta t^2}{2} \\ 0 & 1 & \Delta t \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} p \\ v \\ a \end{pmatrix}_{k-1}$$

Merging the three 3x3 translation system-state matrices with the orientation and rotation speed matrices as sub-matrices, the resulting state-transition-model is of dimension 17x17.

$$F = \begin{bmatrix} F_{trans,x} & 0 & 0 & 0 & 0 \\ 0 & F_{trans,y} & 0 & 0 & 0 \\ 0 & 0 & F_{trans,z} & 0 & 0 \\ 0 & 0 & 0 & F_{ori} & 0 \\ 0 & 0 & 0 & 0 & I_{3x3} \end{bmatrix}$$

For translational motion, the piecewise white noise model was used for the process noise, while for the rotation, values got estimated heuristically.

## V. Implementation

The described methodology was implemented and tested on a Nvidia Jetson Xavier (8GB) system, using a PiEye Nimbus 3D camera as data source. The processing system features a 6-core ARM64 CPU and a 384-Core Volta GPU, that can be utilized with Nvidia CUDA. Its 8GB of LPDDR4x RAM can both be accessed by the CPU and GPU, allowing the use of shared variables to avoid time-costly memory copy commands. The Nvidia Jetson Xavier system is mounted on a custom baseboard, allowing an additional color camera to be used via FPDLink III.[4]

As the PiEye Nimbus 3D camera relies on a Raspberry Pi as its host system, a simple UDP/IP server-client socket between the Nvidia Jetson Xavier and the Raspberry Pi serves as video input. The TOF camera lens correction and the radial recalculation to achieve linearity in the data is solved by multiple lookup tables that get accessed by CUDA kernels. For the extraction and matching of SIFT features [1], and for the 3x3 matrix SVD on CUDA [2], third-party libraries have been used, the surrounding algorithmics were developed in C++ and CUDA.

The implementation parallelizes the algorithm, wherever possible using CUDA. The limited resolution of the used TOF camera limits the number of extracted features, so that – depending on the scene – around 300-500 parallel SVDs get performed. The estimated TOF motion is fused with the IMU data in the Kalman filter for the sake of having a complete pipeline. For demonstration, a virtual rectangle gets drawn into a viewfinder window using Vulkan. The rectangle reacts to the spatial position and orientation of the camera head, as it were a stationary object in the real-world space.

The processing time for the entire pipeline lies around 15ms, which would be sufficient for 60fps. Although, the frame rate of the TOF camera limits the system to 15-20fps. The performance may drop when using a higher resolution TOF camera.

## VI. RESULTS

### A. 3D RANSAC algorithm and rigid motion extraction

The 3D RANSAC feature matching was tested without motion, and against rotational motion as well in translational motion alongside and perpendicular to the optical axis. Across multiple frames during the motion, the total number of extracted features, the number of correctly paired brute-force matches and the number of RANSAC matches are compared in Fig. 2. Both regular image noise and depth noise of the TOF camera influence the matching performance negatively. The threshold is the size of a sphere, in which a matched feature is considered correct.

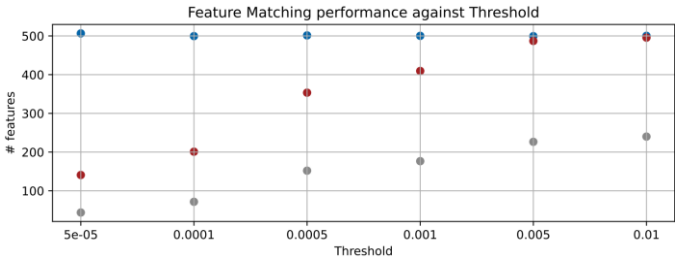


Fig. 2. Comparison of the feature matching performance between the brute-force matcher and the RANSAC algorithm without motion.
Blue dots: Total features
Red dots: RANSAC matches
Grey dots: correct Brute-Force matches

The chosen threshold of 0.0005 translates to a sphere of about 4.4cm in diameter of real-world space. On this threshold, the matching performance in motion was tested and the measurements listed in Fig.3.

TABLE I.

| **Measurement (avg)** | ***Rotation*** | ***Translation X*** | ***Translation X*** |
|---|---|---|---|
| Total Features | 435.9 | 400.0 | 488.4 |
| Brute-force Matches | 117.9 | 104.2 | 112.6 |
| RANSAC Matches | 280.2 | 262.5 | 284.3 |

Fig. 3. Per-Frame average perfocmance of the feature matching approaches

As visible in Fig. 2 And Fig. 3, the RANSAC algorithm consistently leads to an improvement of the matching quality of more than 100% compared to the brute-force matches. A direct comparison is shown in Fig 4, where in contrast to Fig. 1 the RANSAC features are shown.

The motion extraction of the TOF camera can easily be compared to the IMU by performing rotation different axis as shown in Fig 5. The rotation speed extraction roughly follows the gyroscope output but is tainted by more noise. Fig. 6 shows the extracted velocity of two consecutive translational motions – forth and back – in a single direction.

From the results, it becomes apparent that the camera noise of the TOF camera bleeds into the speed and rotation estimation. The low resolution of about 205x265 pixels of the lens corrected image and the slow frame rate of about 15-20fps negatively influence the result as well. As visible in Fig. 6, noise spikes negatively influence the integration results.

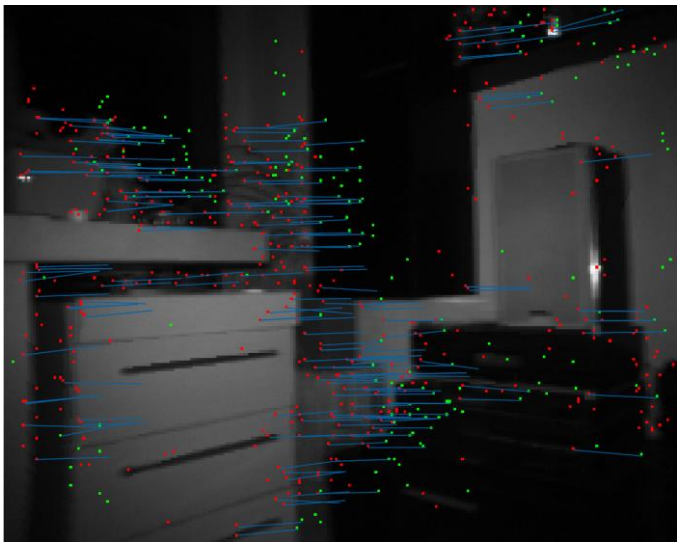

Fig. 4. Sample B/W-Image with drawn feature points, part of of a sequence in which the camera got rotated. Note that only features, whose have gotten matched by the brute-force-matcher got drawn, but there are more in the database. At the end of every line, there should've been a green dot.
Green dots: Frame k-1
Red dots: Frame k
Lines: RANSAC matches

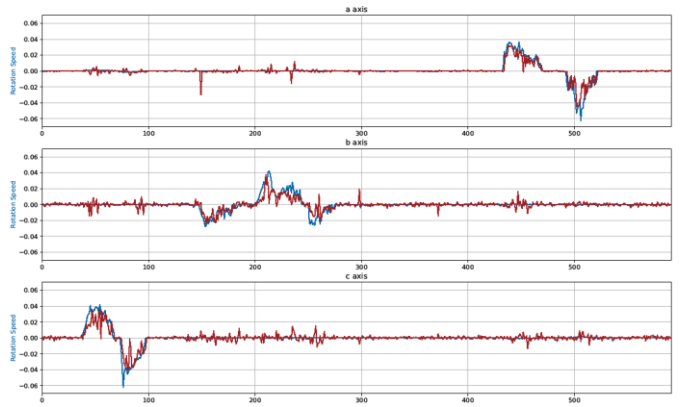


Fig. 5. Comparison of rotation output between the TOF algorithm (red) and the IMU (blue) when rotating the camera head in each axis.

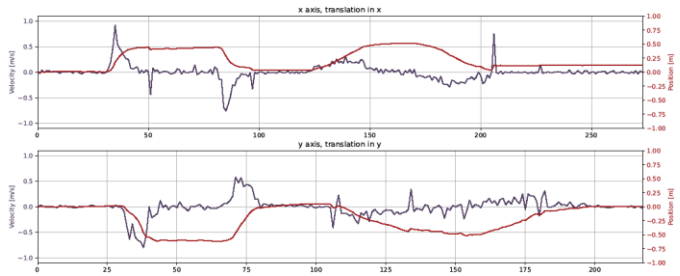


Fig. 6. Plot (purple) and integration (integration) of the translational motion in two separate directions. In each direction, a fast motion and a slow motion were performed and measured.

### B. *Kalman filter for sensor fusion*

The Kalman filter outputs were only tested in a rough manner, as there was no equipment available for reference measurements. Like with the results of the TOF camera algorithm, the camera head was rotated and translated around and along the three axes. Fig. 7 shows how the Kalman filter for rotation speed provides a stable rotation output. The rotation speed outputs follow the gyroscope more closely, than the TOF camera, which lies in the measured noise values of the two data sources.

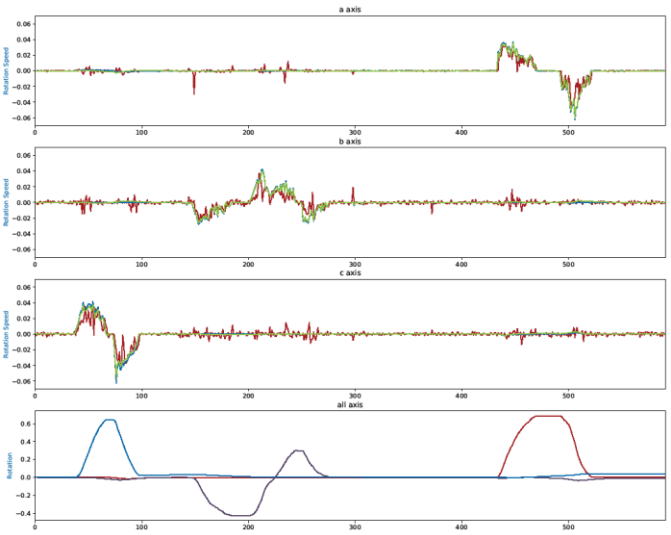


Fig. 7. Plot of the rotation speed (top three) and the rotation (bottom row) of the Kalman filter. Top three: Blue: Gyroscope, red: TOF rotation and green: Kalman Filter output.
Bottom row: red: x-axis, purple: y-axis, blue: z-axis.

For translational motion, the Kalman filter gets tainted by the hysteresis of the accelerometer. The generated offset provided by the hysteresis leads the acceleration to drift away, which leads the accelerometer's raw double integration for the position to quickly diverge, as seen in Fig. 8. The Kalman filter is able to correct that, but its output is worse than the raw integration of the TOF camera algorithm.

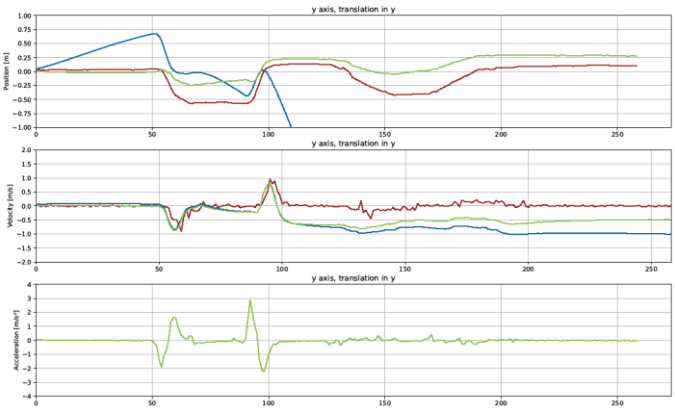


Fig. 8. Plot of the translational motion. On top the position, in the middle the velocity and on bottom the acceleration. In blue, the accelerometer and raw integrations for velocity and position. In red: ToF camera velocity output and its raw integration. In green: Kalman filter output.


## ACKNOWLEDGMENT

I am thankful to my supervisor, Prof. Dr. Matthias Rosenthal, for allowing this deep dive into Augmented Reality and the support given during this thesis. Furthermore, I am grateful for the support and advice from the ZHAW InES HPMM team. Special thanks, especially to Lukas Neuner, for his valuable inputs during this thesis and for proof-reading this document. Additional thanks are given to Alexey Gromov for his support in setting up the Jetson Xavier and proofreading this document.

I am thankful for the chance of gaining further experience in CUDA, Vulkan and for the time given to learn new topics in the math involved in three-dimensional rendering.

In addition, I thank all my friends and family members for giving support and motivation.


## REFERENCES


[1] M. Björkman, N. Bergström and D. Kragic, "Detecting, segmenting and tracking unknown objects using multi-label MRF inference", CVIU, 118, pp. 111-127, January 2014. ScienceDirect

[2] Ming Gao* and Xinlei Wang* and Kui Wu* and Andre Pradhana and Eftychios Sifakis and Cem Yuksel and Chenfanfu Jiang, "GPU Optimization of Material Point Methods" ACM Transactions on Graphics (Proceedings of SIGGRAPH ASIA 2018), vol. 37, Nr 6

[3] Olga Sorkine-Hornung and Michael Rabinovich. Ethz note: Least-squares rigid motion using svd, January 2017.

[4] Alexey Gromov. Optimal Platform for Embedded Supercomputers. Master's thesis, ZHAW, Zurich University of Applied Sciences, Switzerland, 2020