Visual Algorithms for Mobile Robots: Monocular Visual Odometry

Axel Barbelanne
ETH Zurich
abarbelanne@student.ethz.ch

Andre Gomes
ETH Zurich
agoncalve@student.ethz.ch

Jose Lavariega
ETH Zurich
jlavariega@student.ethz.ch

Simone Manni
ETH Zurich
mannis@student.ethz.ch

Video Link: https://youtu.be/ADyzvnv7kyI?si=oRDL_k6TlJ2qT2kr

***Abstract*—In this project, we implement a Visual Odometry(VO) pipeline that is applicable to three datasets. The pipeline uses SIFT features for bootstrapping and then switches to Harris corners. Keypoint tracking is done across consecutive frames. Pose estimation is handled through PNP and RANSAC approach. Our analysis reflects our observations through the development process, and we extend our discussion onto the final results. A discussion of the pipeline performance with and without the additional features is also provided. Futher improvements to the VO pipeline, as well as possible extensions and challenges for future iterations are also discussed. Our code is entirely written in Python. The implementation is not robust to scale drift, although local trajectories remain consistent. Overall ground truth shape is preserved.**

I. Introduction

In freely-moving autonomous platforms, a common building block is to estimate the platform's pose given some sensor suite. A common way to do so is by implementing a visual odometry pipeline. The pose of the robot is estimated from visual changes in the environment, captured typically by stereo cameras, LiDAR, RADAR, or some fusion of all of these. These sensor suites can be extensive and may include IMU or rotational angle encoders, which provide an additional element to the sensor fusion system to reliably obtain a pose. However, rotational encoders are not available for non-wheeled platforms, the usage of stereo camera and LiDAR systems is not encouraged for low-cost systems, and full state knowledge may not be obtainable by the robotic platform.

Our Visual Odometry pipeline uses a monocular approach for addressing the structure from motion problem. A motivating real-world example for a monocular approach is the handling of many low-cost platforms used in rescue operations. Rescue operations may require the robot to navigate through an unknown environment and create a rough map of its path as it searches for people after an avalanche. Due to the adversity of this environment, building a low cost robot becomes a design choice.

II. Preliminaries and Building Blocks

A. Keypoint Detection [2]

Keypoint detection is a fundamental feature of a Visual Odometry System since it is through detection and, afterwards tracking of these keypoints, that information about movement is extracted. An ideal keypoint detector should find distinctive and repeatable interest points, while being computationally efficient. In this implementation two different feature detectors are used.

The Scale Invariant Feature Transform (SIFT) algorithm is used for its highly repeatable and distinctive keypoints, which are desired features during the Bootstraping phase of the VO. However, due to the comparatively slow performance of this detector, the faster Harris Corner Detector is used during the remaining phase of the VO pipeline.

1) Scale Invariant Feature Transform [2]: The SIFT detector uses the DoG feature detector combined with a image pyramid to detect over multiple scales. A feature is selected when it is a maximum on the images and scale dimensions (3x3x3 pixel cube). A SIFT feature is a scale, rotation and affine-invariant, with the main downside that it is expensive to compute compared to other types of features. A SIFT feature is constructed from squared patch sizes of length 16.

1) To find the dominant orientation, the patch is multiplied by the Gaussian kernel and a Histogram of Gradients is created, weighted by the magnitudes.
2) From the HoG, obtain all dominant maxima, and create a different keypoint descriptor for each dominant orientation.
3) The patch is de-rotated using an interpolation method.
4) From the original 16x16 patch, compute a HoG for each 4x4 subpatch and concatenate all histograms into a single 1D vector of size 128 (4x4x8)
5) The 1D vector is scaled to have an L2 norm of value 1 to be invariant to affine and additive illumination.
6) To be scale invariant, the SIFT feature is assigned a specific scale, which is chosen as an extrema of a scale pyramid computed through the DoG kernel.

OpenCV implemenets a version of the SIFT detector.

2) Harris Corner Detector: The Harris Corner Detector aims to implement the Moravec detector in a more computationally efficient way. The Moravec Detector computes the Sum of Square Distances (SSD) for 4 distinct patch shifts ($\rightarrow$, $\downarrow$, $\searrow$, $\swarrow$), selects the minimum of those four values and compares it to a set threshold for corner, if it is larger, the pixel is considered a corner.

$$SSD(\Delta x, \Delta y) = \sum_{x,y \in P} \big(I(x,y) - I(x+\Delta x, y+\Delta y)\big)^2 \quad (1)$$

The Harris detector aims to remove the computationally expensive step of calculating the SSD 4 times, by linearly approximating the pixel intensity function:

$$SSD(\Delta x, \Delta y) \approx \sum_{x,y \in P} \big(I_x(x,y) \cdot \Delta x + I_y(x,y) \cdot \Delta y\big)^2$$
$$= \begin{bmatrix}\Delta x & \Delta y\end{bmatrix} M \begin{bmatrix}\Delta x \\ \Delta y\end{bmatrix}$$

Where the matrix M is given by

$$M = \begin{bmatrix}\Sigma I_x^2 & \Sigma I_x I_y \\ \Sigma I_x I_y & \Sigma I_y^2\end{bmatrix}$$

This linearized version defines a paraboloid, where the semi-axis are the directions of highest and lowest increase in SSD and are defined by the eigenvectors of the Second-Moment Matrix M. The eigenvalues describe the rate of increase of the SSD in the eigenvector direction. When a corner is present, it is expected that the SSD would increase at a significant rate in every direction, as such, a corneress function R is defined to capture this effect:

$$R = det(M) - \kappa * tr^2(M)$$

This corneress function is specific to the Harris Detector and it serves as a heuristic that attributes high values of R when both eigenvalues are large, and a low R value when one or both of the eigenvalues are small. This R function prevents the need for explicit eigenvalue calculation.

After computing the R function for the whole image, a non-maximum suppression is performed to prevent selection of close-by pixels. Finally, the N highest R value pixels are selected as corners.

B. Keypoint Matching

Keypoint matching is an important part of the bootstrapping phase, in which it is required to correctly perform 2D-2D matches between two image frames. In this VO system, this is only performed at the start of the pipeline and it utilizes the SIFT descriptors of the keypoints in each images to perform the matches. A brute force method is performed.

C. Feature Tracking [4]

To track individual features through the images sequence, the KLT tracker is utilized. This direct method allows for sub-pixel accuracy and fast performance when compared to matching feature descriptors between every frame.

The tracker works by defining a warping model, $W(p,x)$, finding the parameters p that minimize the SSD between a template image patch T and the warped patch in the image to track I:

$$SSD = \sum_{x \in T} \big(I(W(p,x)) - T(x)\big)^2 \tag{2}$$

As such, it is possible to track keypoints in a sequence of images by defining the template as a patch around each keypoint and then finding the p values that minimize the SSD for each keypoint patch. This enables computationally efficient tracking of keypoints. The p values that minimize the equation have the closed form solution:

$$\Delta p = H^{-1} \sum_{x \in T} [\nabla I \frac{\partial W}{\partial p}]^T [T(x) - I(W(p,x))]$$

where H is:

$$H = \sum_{x \in T} [\nabla I \frac{\partial W}{\partial p}]^T \sum_{x \in T} [\nabla I \frac{\partial W}{\partial p}]$$

The function cv2.calcOpticalFlowPyrLK applies and solves the Lucas Kanade Tracker method.

D. Prespective from 3 Points (P3P) and RANSAC [6] [7]

The P3P algorithm enables localization of a calibrated camera in the world frame when 3D landmarks and 2D-3D correspondences are known. This method works by solving a 4^{th} order polynomial derived from a system of three equations based on the law-of-cosines. Three point correspondences are used to reduce the number of possible solutions to a maximum of 4, and a fourth point is used to extract a unique solution. Given that the P3P algorithm only utilizes 3 + 1 point correspondences to find the location and orientation of the camera, it is highly susceptible to outlier correspondences. As such, a robust outlier detection system is fundamental for a reliable pose estimation.

P3P follows the following relation for 2D- 3D point correspondences:

$$s \begin{bmatrix}u \\ v \\ 1\end{bmatrix} = K[R|T] \begin{bmatrix}X \\ Y \\ Z \\ 1\end{bmatrix}$$

The Random Sample Consensus (RANSAC) algorithm is an iterative method to estimate the parameters of a given model. It works by randomly selecting the minimum number of points required to fit a predefined model (in this case, 3+1 points for a pose estimate using P3P), and then verifying and keeping track of which other data points are within a specified error threshold of the fitted model, these are considered inliners for this model estimation. The algorithm runs iteratively, selecting 3+1 random points, fitting a model, and computing the inliers. At the end of the iteration process, the model that recorded the highest number of inliners is selected, and the datapoints that do not fit this model are considered outliers and removed.

E. Essential Matrix

To recover the essential matrix from keypoints in two different frames, the following relations are used:

$$\lambda_1 p_1^i = K[I|0] \begin{bmatrix}X_w^i \\ Y_w^i \\ Z_w^i \\ 1\end{bmatrix}$$

$$\lambda_2 p_2^i = K[R|T] \begin{bmatrix} X_w^i \\ Y_w^i \\ Z_w^i \\ 1 \end{bmatrix}$$

Where K is the intrinsic parameters, and frame 1 is used as a reference to obtain the relative R and T of frame 2.

Enforcing epipolar geometry, the epipolar constraint equation for a calibrated camera is:

$$p_2^T E p_1 = 0$$

To solve for the essential matrix E ($[R|T]$), the singular value decomposition is used. E becomes decomposed into $E = U\Sigma V^T$ This gives four solutions to the T and R candidates:

$$\hat{T} = U \begin{bmatrix} 0 & \mp 1 & 0 \\ \pm 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \Sigma V^T$$

$$\hat{R} = U \begin{bmatrix} 0 & \mp 1 & 0 \\ \pm 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} V^T$$

And premultiply by K to get T and R. Since there are 4 solutions for R and T combinations, the correct choice is whichever combination where most of the points are in the front plane of both frames (those with positive depth)

In the code, the pipeline uses the cv2.findEssentialMat() from the OpenCV library.

F. Triangulation

For triangulation, we do singular value decomposition on the essential Matrix to solve the system of equations:

$$\lambda_1 p_1 = M_1 P, \lambda_2 p_2 = M_2 P$$

$$[p_1]_\times \cdot P = 0, [p_2]_\times \cdot P = 0$$

III. Technical Approach - Visual Odometry Pipeline

1) Overview: Implementation of a straightforward Visual Odometry (VO) pipeline, using the Open CV library, which provides some helper functions, mainly in the selection of feature detectors and matchers. A Brute Force Feature Matcher is used, with features initialized witha SIFT Feature detector. The scipy library provides functions for minimization and least squares. The Visual Odometry Class takes in as parameters: the feature detector (SIFT, during bootstrapping), the feature matcher (Brute Force Matcher), the intrinsics matrix that is given with the dataset, and a parameter indicative of the used dataset.

In (2), the basic intialization and feedback loop of the VO pipeline is shown, with the relevant elements. This figure is referenced throughout the writeup.

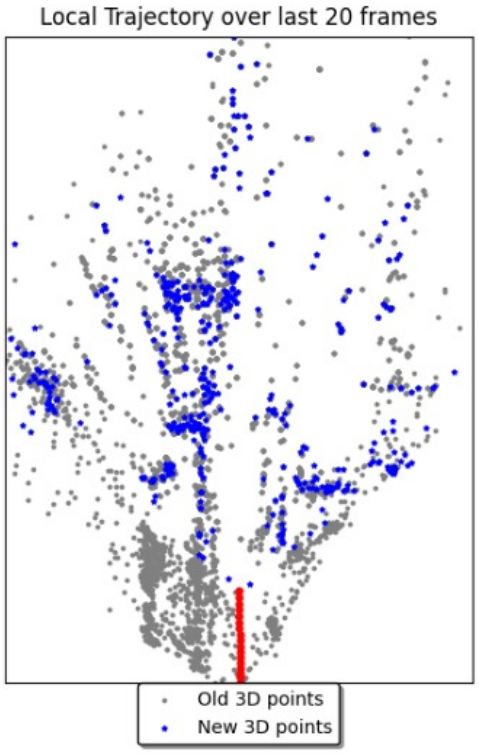

Fig. 1. Local trajectory for the kitti dataset, consistent in scale for the prior 20 frames. Tracked 3d Features are shown

A. Bootstrap (Initialization)

Reference: Figure 2 Our pipeline works by defining the start of the dataset as the origin of the world frame, and any motion afterwards traverses the camera through the world frame. The pose of the first image is treated as the origin of the world frame.

Upon initialization, parameters are set and a descriptor size of 128, which is hard-coded to comply with the SIFT descriptor. This hard-coded value is kept. Matrices that will be referenced throughout the pipeline are initialized: Matrices to store Keypoints (denoted henceforth as **S**) and Candidate Keypoints, (denoted henceforth as **C**) the first observtions of the candidate keypoints (denoted as **F**), and the observed 3D points and the camera Poses. (Denoted **P** and **T** respectively.)

The pipeline uses PNP and RANSAC to regress and obtain the transform back into our camera pose, with respect to the origin of the worldframe. Depending on the dataset used, the number of iterations for RANSAC changes. For the KITTI and Parking datasets: 80 iterations, while with the Malaga dataset: 150 iterations. Other parameters are a reprojection error limit of 1.0, and requiring a 0.99 confidence from RANSAC.

In the bootstrapping phase the pipeline finds candidate keypoints in the initial frames, begin tracking them and matching correspondences, and remove weak candidates. A transforma-

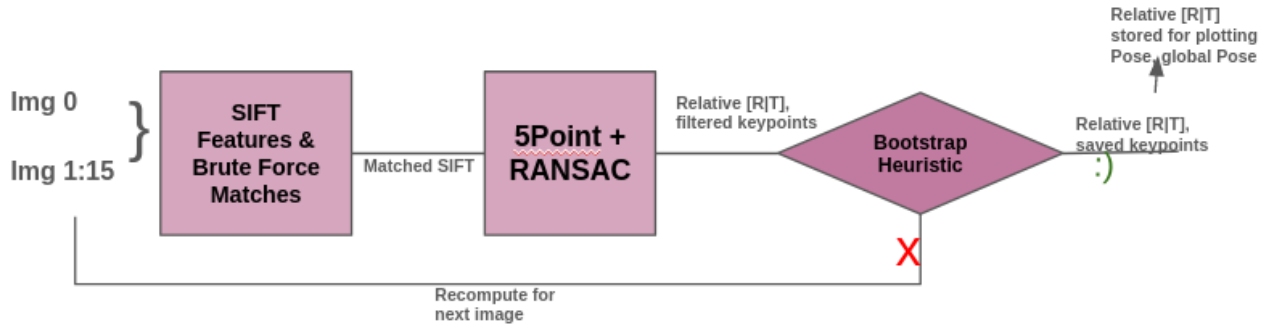

Fig. 2. Bootstrap Portion of the pipeline

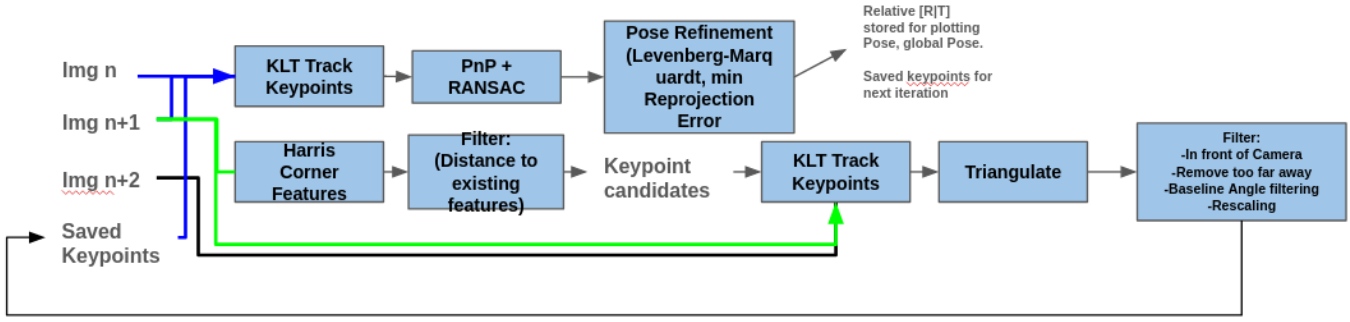

Fig. 3. Continuous Operation Portion of the pipeline

tion matrix is then inatialized, a point cloud populated, and initial features are stored.

1) Automatic Bootstrap: Automatic Bootstrap is a method to search for a good pointcloud initialization among the first 15 images in the dataset. A series of keypoints and descriptors is obtained for the first image in the dataset, and the method is terminated whenever the following heuristic is completed:

$$||\frac{T_i - T_0}{\frac{1}{n}\sum(depth)}|| \geq 0.10$$

[1]

Where $T_i - T_0$ is the distance between keyframes, and the denominator is the average estimated depth of all Keypoints as computed via transfomation matrix into the world frame. Here the notation T_i means the Transform of frame i. When the heuristic is satisfied, the candidate keypoints are stored **S** is updated with the keypoint values, and **P** with the keypoint poses (in the world frame). The scale is initialized to the mean of the norm of all the pose dimensions. Mean of (Norm X, Norm Y, Norm Z).

B. Continuous Operation

Reference: Figure 3

1) Tracking and Matching: Each subsequent frame is treated as a potential keyframe. While the authors recognize that there could be a performance boost by spacing out keyframes a bit more, there is an accuracy risk in doing so, or not being able to match features of a new frame to the old frame. It was also noted that the processing time for each frame was not significant (the pipeline processes frames at a rate on the order of 26 FPS). As a note, the biggest contributor to processing time performance was during plotting after every frame, and not through actually running the pipeline.

The KLT method is used to track existing keypoints from one frame to the next, which updates the keypoits **S** and the keypoint poses **P**. These two matrices are then fed through a PNP method with RANSAC to find all inliers and a preliminary position transform matrix (**T**). **T** is then refined through minimizing the reprojection error through the Levenberg-Marquardt method.

From the **T** matrix, the rotation and translation components are extracted to obtain the pose for that frame. The poses are saved and kept for plotting once the full dataset has been run through.

2) New Landmarks: In Continuous Operation, the pipeline switches to Harris Corner features. New Corner features are looked for in every keyframe, and are filtered out if they are too close to the previous keypoints. If the minimal distance to any keypoint is greater than a set threshold, then the new feature is kept as a candidate keypoint.

For candidate keypoints stored in **C** and **F** matrices, the

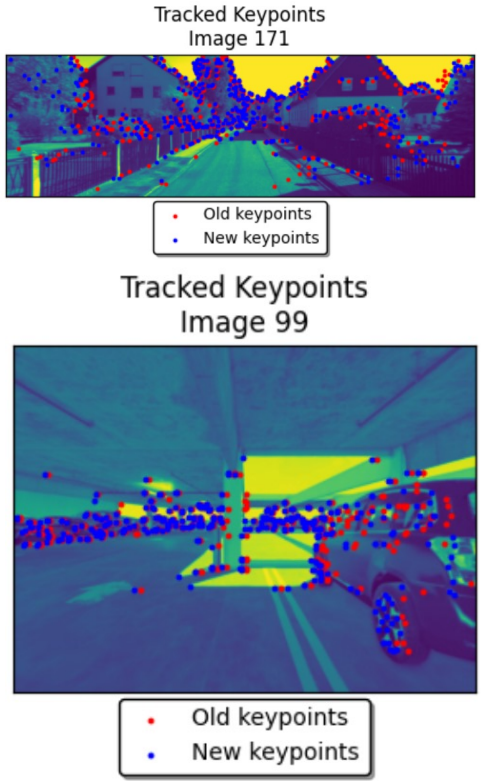

Fig. 4. New Landmarks

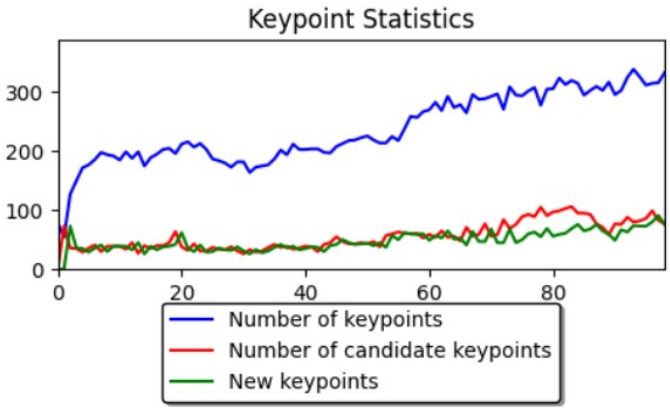

Fig. 5. Tracked landmarks over time. Full dataset run.

continuous operation portion of the pipeline triangulates and keeps only those with positive depth. The candidate keypoints are then rescaled and further pruned based on a distance from the camera, and pruned again from a difference in a baseline angle, which is specific to each dataset.

C. Improvements, Quick Extensions, and Discussion

With any VO pipeline, there comes the issue of scale drift. The results shown in the next subsection show the prevalence of the scale drift for the base implementation of the pipeline. There are multiple ways of addressing the scale drift problem, given a limited sensor suite. Some improvements the team thought and made progress towards implementing were:

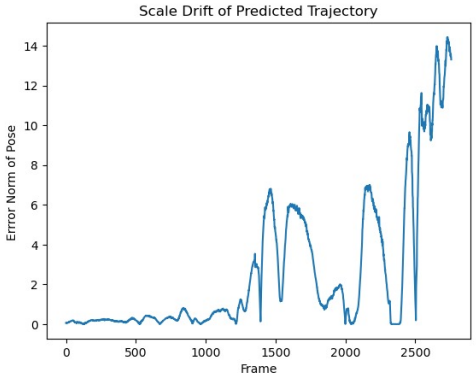

Fig. 6. Scale drift as moving through the dataset.

1) Correcting when revisiting known locations/ mapped locations [5]: The team made good progress towards building an visual vocabulary that would recognize a place on the dataset and compare it to ground truth. This with the objective of simulating a situation where the environemnt is already mapped, so that the actual pose is extracted.
2) Motion Model constraint: keep an a motion model given the constraints of what is known from the dataset: these are all cars, and they follow a dynamics model. The idea here was to implement a bicycle model of the car so that large turns would be compared against dynamic feasibility. And would be otherwise scaled down if the predicted motion from the image alone would be too great.
3) IMU Integration [3]: IMU data is readily available, and a thought was to integrate this to upgrade to a VIO pipeline. IMU measurements alone tend to drift with time, but they give an absolute scale, which is ambiguous when dealing with a monocular pipeline as during this project.
4) Bundle adjustment would be used to minimize the reprojection error by refining the 3D coordinates. This is a measure that would add robustness to the existing pipeline.

Regrettably, time constraints meant that these features were not fully implemented or never left the idea phase. The most progress was done towards the recognizing mapped environments and correcting case.

D. Performance and Results

The video submitted alongside this writeup shows the performance on all three trajectories in more detail. Refer to figures 4, 1, 6 and 5 for plots on identifying new landmarks, the local trajectory using the last 20 computed poses, scale drift as a function of how many images the system has seen in the dataset, and statistics on new features as they are added to the existing features and dropped as they exit the frame. A reach goal was to implement an absolute error metric that reorients the ground truth and compares trajectory drift from

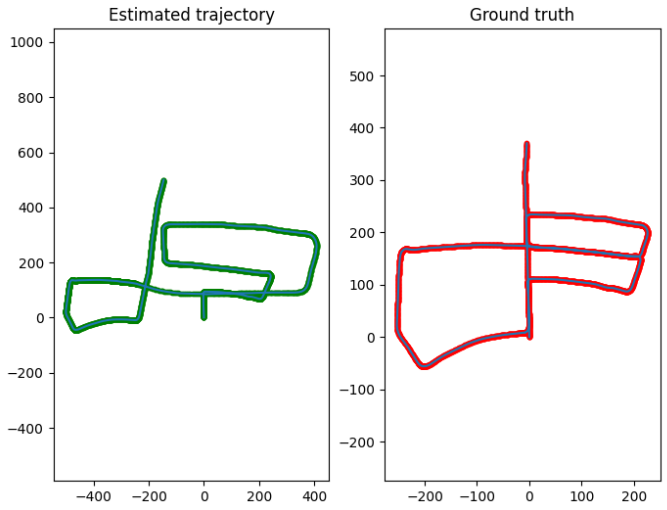

Fig. 7. Groundtruth of the Kitti dataset, compared to the estimated trajectory

the ground truth and 'correctness' of the trajectory. Due to time constraints, this was not possible, and not used, as it would have been used to compare different tracking methods. (Implementing current VO research as an additional feature).

IV. Distribution of Contributions

Initial passes of the code within the first couple of weeks from assignment release were written primarily by Andre and Simone, while the subsequent extensive debugging and integration was done equally across all members, as we became more available from our other classes. Not pictured in the commit history were multiple meetings at different hours to bounce ideas, try new approaches, or hunt down the bugs. The plotting interface for debugging was initially written by Jose, but extended in its final video submission and real-time form by Axel. Axel and Simone led the final integration effort. Andre and Jose wrote the report.

Acknowledgment

We would like to thank Prof. Scaramuzza and the teaching staff for an extensive and challenging yet fun course, for all the time dedicated making the labs, for their patience and understanding when answering our questions, and for providing us with the tools to build and inspect Visual Algorithms in the future.

References

[1] Scaramuzza,D. "Visual Algorithms for Mobile Robots Multiple View Geometry 4" UZH Course
[2] Scaramuzza,D. "Visual Algorithms for Mobile Robots Point Feature Detection and Matching" UZH Course
[3] Scaramuzza,D. "Visual Algorithms for Mobile Robots Visual Inertial Fusion" UZH Course
[4] Scaramuzza,D. "Visual Algorithms for Mobile Robots Tracking" UZH Course
[5] Scaramuzza,D. "Visual Algorithms for Mobile Robots Recognition" UZH Course
[6] Scaramuzza,D. "Visual Algorithms for Mobile Robots Camera Calibration" UZH Course
[7] Scaramuzza,D. "Visual Algorithms for Mobile Robots Filtering" UZH Course
[8] OpenCV Documentation, URL: https://docs.opencv.org/4.9.0/d6/d00/tutorial_py_root.html