Stereo vision pipeline and Graph-SLAM implementation for FSAE driverless vehicle competition

Exam and Project Work report on
COMPUTER VISION AND IMAGE PROCESSING M 2024/2025

Gabriele Ceccolini

November 13, 2025

Contents

1 Objectives and Problem Description **3**
1.1 Navigation Environment and project scope . . . 3
1.2 Hardware Setup . . . 4
1.3 Software stack . . . 5

2 Stereo cone depth estimation **6**
2.1 Image Acquisition, Validation and Rectification . . . 7
2.2 Cone Object Detection with YOLO . . . 8
2.2.1 Custom YOLO training . . . 9
2.2.2 YOLO inference . . . 9
2.3 Multi-stage Stereo Matching of Detections . . . 10
2.3.1 Geometric Candidate Filtering . . . 10
2.3.2 Template Matching . . . 12
2.3.3 Feature-Based Correspondence with ORB . . . 13
2.3.4 Triangulation and Variance Estimation . . . 16
2.3.5 Coordinate System Transformation . . . 18
2.4 3D Position Estimation Validation . . . 19
2.5 Performance and Timing Analysis . . . 21

3 Simultaneous Localization and Mapping (SLAM) **23**
3.1 Graph-Based SLAM Formulation using g2o Framework . . . 23
3.1.1 Constraint Error Functions and objective function . . . 25
3.2 Graph Construction and Data Association . . . 27
3.2.1 Pose Node and Odometry Edge Creation . . . 27
3.2.2 Landmark Observation and Data Association . . . 28

3.3 Operational Modes and Graph Lifecycle 29
3.3.1 Operational Modes: Mapping vs. Localization 29
3.3.2 Graph Lifecycle: Managing Vertices for Efficiency 30
3.4 Optimization Strategies and Loop Closure 31
3.4.1 Incremental Optimization 31
3.4.2 Loop Closure and Full Optimization 31
3.5 System Visualization and Results 32

Conclusions **33**

Acknowledgments **34**

The algorithm and the threshold value were tuned heuristically by testing on multiple video streams, both recorded on-track and off-track, until the issue no longer appeared.

```
bool detect_shifted_frame(const cv::Mat &stereo_frame, double threshold)
    {
    int center = stereo_frame.cols / 2;
    cv::Mat stereo_frame_grayscale;
    cv::cvtColor(stereo_frame, stereo_frame_grayscale, cv::
        COLOR_BGR2GRAY);     // Convert to grayscale
    // Compute sum of absolute differences between center columns
    int sum_left = 0, sum_center = 0, sum_right = 0;
    for (int i=0; i<stereo_frame_grayscale.rows; i++) {
        sum_left += std::abs(stereo_frame_grayscale.at<uchar>(i, center
            - 1) - stereo_frame_grayscale.at<uchar>(i, center - 2));
        sum_center += std::abs(stereo_frame_grayscale.at<uchar>(i,
            center) - stereo_frame_grayscale.at<uchar>(i, center - 1));
        sum_right += std::abs(stereo_frame_grayscale.at<uchar>(i, center
            ) - stereo_frame_grayscale.at<uchar>(i, center + 1));
    }
    // Check if the center two columns have significantly higher
        difference compared to the left and right ones
    float center_weight = sum_center / static_cast<float>(sum_left +
        sum_center + sum_right);
    // If the center weight is around 33% it is similar to the besides
        ones, so there is no frame separation;
    // If there is no left-right frame separation, probably a frame
        shift occurred
    if (center_weight < threshold) {
        std::cout << "Detected frame shift: center_weight=" << std::
            setprecision(2) << center_weight << "%" << std::endl;
        return true;
    }
    return false;
}
```

Listing 1: Detection logic of shifted frames to discard

After validating the incoming frames, the node applies rectification to the stereo pair using calibration data provided by the manufacturer at production time. For additional safety, the calibration was later repeated using a dedicated software tool also supplied by the manufacturer.

The validated and rectified frames are published on `/camera_publish_topic` ready to be accessed by the following nodes.

The `detect_shifted_frame` function is implemented in `ros_frame_capture_node.cpp` [113-142].

2.2 Cone Object Detection with YOLO

As mentioned earlier, the first step of the pipeline consists of the 2D detection and classification of cones. For this task, a reliable yet fast object detection algorithm is required, capable of operating on a video stream in real time. The chosen model for this purpose is YOLOv11 for its well-known balance between speed and accuracy. The system is designed to run using either the NVIDIA TensorRT engine when deployed on NVIDIA GPUs, or the OpenVINO toolkit when executed on

Intel-based platforms. The desired backend must be selected prior to compiling the perception pipeline.

2.2.1 Custom YOLO training

YOLOv11 was fine-tuned starting from the default Ultralytics weights using the public FSOCO dataset (Formula Student Objects in Context). This dataset is a heterogeneous collection (see Fig. 7) of cone images, gathered both on and off the track over several years by different teams participating in the competition.

The model was trained for 300 epochs on a training set of 7,120 images (without augmentation), validated on 1,969 images, and tested on 689 images.

Class	Precision	Recall	mAP50	mAP50-95
All Classes	0.849	0.765	0.824	0.570
Blue Cone	0.922	0.806	0.896	0.615
Yellow Cone	0.926	0.794	0.892	0.608
Orange Cone	0.925	0.787	0.877	0.603
Large Orange Cone	0.868	0.873	0.908	0.710
Unknown Cone	0.603	0.566	0.547	0.315

Table 1: YOLOv11 performance on the FSOCO test set.

The fine-tuned YOLOv11 model achieves an overall mAP50 of 0.824, with precision and recall of 0.849 and 0.765, respectively, confirming strong general performance. Results are particularly robust for the main cone classes (Blue, Yellow, Orange), all with precision above 0.92 and mAP50 close to 0.90. Large Orange Cones are detected most accurately, while the Unknown class shows weaker performance, as expected due to its ambiguous nature. Overall, the detector proves highly effective for the intended task of identifying track boundary cones.

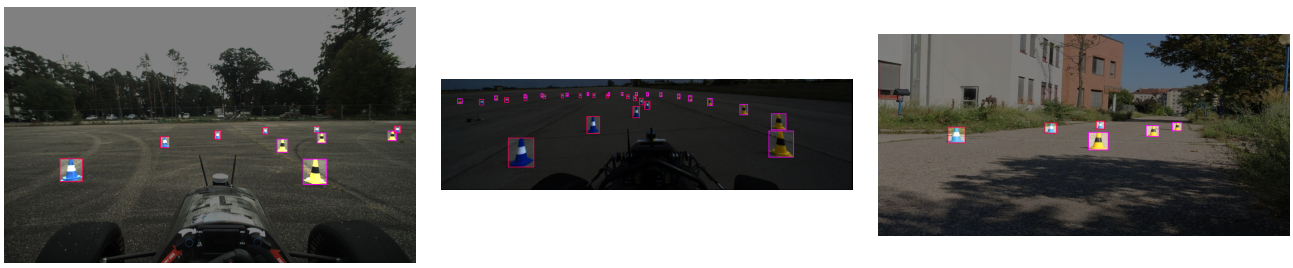

Figure 7: Examples from the FSOCO dataset.

2.2.2 YOLO inference

The detection process for each stereo pair proceeds as follows:

1. **Image preprocessing.** The raw left and right images are resized to the model's fixed input size (640×640) while preserving the original aspect ratio. Each image is scaled to fit and the remaining area is filled with black padding to avoid geometric distortion.

2. **Batched inference.** The two preprocessed images are stacked into a single batch of size two and passed to the inference backend (TensorRT on the ASU, or OpenVINO on Intel platforms). Processing both views in one pass is more efficient than running two separate inferences.

3. **Post-processing and filtering.** Raw detections are filtered by a confidence threshold and then refined with Non-Maximum Suppression (NMS) to remove overlapping boxes, retaining a single, best bounding box per cone (see Fig. 8).

The output of this stage is two independent lists of 2D bounding boxes—one for the left image and one for the right, each containing pixel coordinates, width and height, confidence, and class ID (e.g., blue cone, yellow cone). This data is directly used by the following section for stereo matching in the same ROS node.

Figure 8: YOLO output after NMS, with confidence numbers.

The YOLO cone detection logic is implemented in `ros_yolo_detector_node.cpp` [549-864].

2.3 Multi-stage Stereo Matching of Detections

Once the bounding boxes have been obtained for both the left and right images, the next and most critical step is to solve the association problem: determining which bounding box in one frame corresponds to which in the other.

The proposed pipeline adopts a multi-stage approach, where fast and lightweight geometric filters are first applied to eliminate clearly invalid candidate, followed by a more precise template matching step to refine the matches. Once a set of matched bounding boxes is established, local feature matching is optionally performed to obtain multiple correspondence points for a more accurate triangulation.

2.3.1 Geometric Candidate Filtering

As an initial step in the matching process, the algorithm quickly discards cone pairs that do not satisfy basic geometric constraints derived from the stereo camera setup. This pre-filtering phase constructs a list of plausible candidates for each bounding box in the left image before attempting a more computationally expensive template matching.

A bounding box in the right frame is considered a valid candidate only if all of the following conditions are met:

estimates are highly accurate and consistent; at 20 m, the mean error approaches 2 m, and the standard deviation exceeds 1 m, reflecting higher uncertainty at longer ranges.

- **Consistency of ORB vs. Center Point.** The ORB-based approach consistently shows lower standard deviation than the center-point method. For example, at 15 m the ORB method's deviation (0.68 m) is nearly half that of the center-point approach (1.16 m), confirming that multiple keypoints yield a more stable and noise-resistant estimate.

- **Sensor Limitation Confirmation.** The measured errors are consistent with the camera manufacturer's specified depth accuracy (less than 7% at 20 m, or roughly 1.4 m). Our results—showing mean errors around 1.7–2.0 m at 20 m—align with this physical limitation, confirming that the intrinsic precision of the stereo sensor defines the upper bound of achievable accuracy.

In conclusion, both methods achieve comparable average accuracy, but the ORB-based feature matching approach provides higher stability and reliability, especially at medium and long ranges. These characteristics make it the preferred configuration for the stereocamera pipeline used in the autonomous vehicle.

2.5 Performance and Timing Analysis

To assess the real-world performance of the stereocamera pipeline, a series of timing benchmarks was conducted using data recorded during a manually driven lap at the Rioveggio Karting Circuit. The experiments were executed on a high-performance workstation featuring an AMD Ryzen 9 6900HX CPU and an NVIDIA GeForce RTX 3080 Mobile GPU, which closely mirrors the onboard ASU hardware.

The pipeline was evaluated under four different configurations to analyze the impact of the inference engine (TensorRT vs. OpenVINO) and the feature matching strategy (center point vs. ORB features). Timing results, averaged over 1,870 frames, are summarized in Tables 3 and 4.

The results highlight several important aspects of the pipeline's performance:

- **Inference Engine Impact.** The inference engine is the dominant factor influencing total processing time. The GPU-accelerated TensorRT configuration is approximately 3.6 times faster than the CPU-based OpenVINO configuration (6.78 ms vs. 24.72 ms), confirming the critical role of GPU acceleration for real-time deep learning inference.

- **Feature Matching Overhead.** The choice of matching strategy introduces a clear trade-off between speed and robustness. Enabling ORB features increases total processing time by about 1.4 ms (TensorRT) and 1.8 ms (OpenVINO). This overhead arises because the feature matching and triangulation stages must handle multiple keypoints per cone instead of a single center point. The higher standard deviation observed in the feature matching stage reflects the variable number of ORB computations across frames.

- **Real-Time Capability.** All configurations operate well within the time budget for the 60 fps camera (16.7 ms per frame). The TensorRT pipeline achieves full processing in under 10 ms, leaving limited but sufficient room for other tasks such as SLAM and motion planning

algorithms. The OpenVINO configuration, while slower, still runs at an acceptable rate of around 37 Hz, making it a viable fallback option when GPU acceleration is unavailable.

Processing Stage	Center Point	ORB Features
Preprocessing	0.34 (0.08)	0.34 (0.07)
Inference	6.78 (2.45)	6.78 (2.05)
Postprocessing	0.38 (0.08)	0.38 (0.08)
BBox Matching	0.38 (0.27)	0.40 (0.28)
Feature Matching	0.00 (0.00)	1.41 (4.27)
Triangulation	0.02 (0.01)	0.11 (0.07)
Sending	0.04 (0.02)	0.04 (0.01)
Total	**7.95 (2.57)**	**9.46 (6.35)**

Table 3: Mean and standard deviation of processing times (ms) for the TensorRT configuration running on an NVIDIA RTX 3080 GPU.

Processing Stage	Center Point	ORB Features
Preprocessing	1.31 (0.19)	0.37 (0.08)
Inference	24.72 (1.61)	24.30 (1.25)
Postprocessing	0.59 (0.07)	0.60 (0.06)
BBox Matching	0.50 (0.36)	0.49 (0.35)
Feature Matching	0.00 (0.00)	1.76 (2.69)
Triangulation	0.03 (0.01)	0.15 (0.08)
Sending	0.05 (0.01)	0.05 (0.01)
Total	**27.20 (1.74)**	**27.72 (3.50)**

Table 4: Mean and standard deviation of processing times (ms) for the OpenVINO configuration running on an AMD Ryzen 9 CPU.

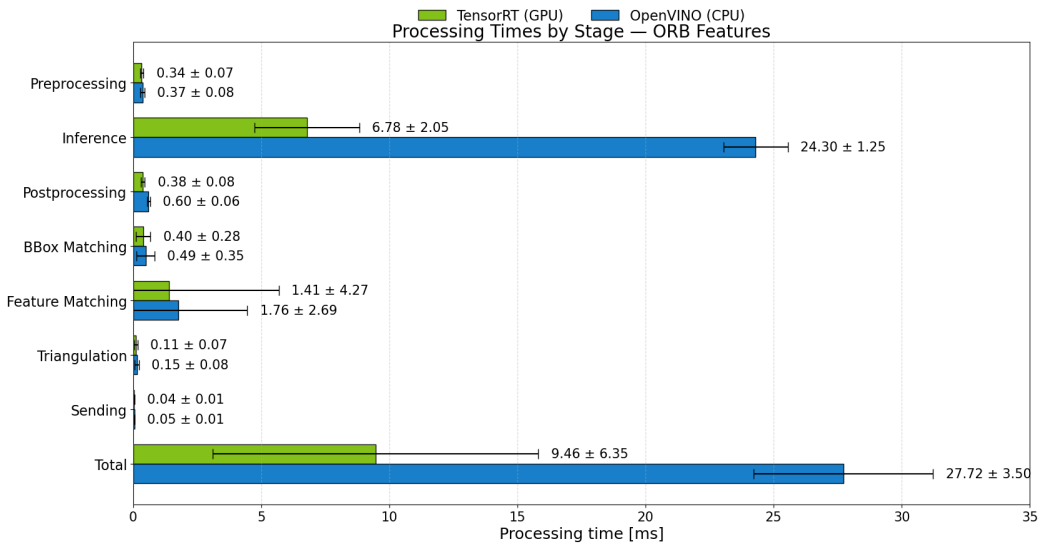

Figure 15: Comparison of average processing times across main pipeline stages for the TensorRT (GPU) and OpenVINO (CPU) configurations.

Acknowledgments

This project was developed in collaboration with the Unibo Motorsport Formula SAE team. I would like to specifically thank the following individuals for their contributions to the topics covered in this report:

- Federico Fusa, Beatrice Bottari, and Gabriele Pattarozzi for their work on the perception and computer vision stack.
- Matteo Fornaini and Alberto Genovese for their contributions to the SLAM system.

References

[1] Edoardo Fusa, *Pushing Cars' Limits: Exploring Autonomous Technologies in the Formula SAE Driverless Competition*, Tesi di Laurea Magistrale, Università di Bologna, 2025. https://amslaurea.unibo.it/id/eprint/35885/1/AI_Master_Thesis___Edoardo_Fusa.pdf

[2] Alessandra Tonin, *Design of a perception system for the Formula Student Driverless competition: from vehicle sensorization to SLAM*, Tesi di Laurea Magistrale, Università di Padova, 2023. https://thesis.unipd.it/retrieve/ed7c1b99-ece9-46fa-bec2-fd163208eb8c/Tonin_Alessandra.pdf

[3] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi, *You Only Look Once: Unified, Real-Time Object Detection*, In Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016. https://arxiv.org/abs/1506.02640

[4] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski, *ORB: an efficient alternative to SIFT or SURF*, In Proc. of the IEEE Int. Conf. on Computer Vision (ICCV), 2011.

[5] *Propagation of uncertainty*, Wikipedia https://en.wikipedia.org/wiki/Propagation_of_uncertainty

[6] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, *g2o: A General Framework for Graph Optimization*, In Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA), Shanghai, Cina, Maggio 2011. https://mengwenhe-cmu.github.io/Reading-Reports/Research/Localization/Graph_Optimization/g2o_A_General_Framework_for_Graph_Optimization/paper.pdf

[7] Giorgio Grisetti, Rainer Kümmerle, Hauke Strasdat, Kurt Konolige, *g2o: A general Framework for (Hyper) Graph Optimization*, Technical Report, 2025.