Embedded Computer Vision System for Practical Image-Based Visual Relative Formation Control

Ian Michel, Bayu Jayawardhana, Ridho Rosa.

Abstract— **Abstract— Multi-agent robotic formation control traditionally depends on high-fidelity sensors (e.g., LiDAR) and external positioning (GPS), which constrain scalability and fail in GPS-denied environments. Image-Based Visual Relative (IBVR) control proposes a monocular-camera-only alternative by inferring inter-robot distances from apparent object size in the image plane, but has to date lacked real-world validation. This work presents the design, implementation, and experimental evaluation of an embedded IBVR perception–actuation pipeline. A ROS-based data collection framework recorded synchronized camera images and ground-truth poses, supporting automated 2D and 3D bounding-box annotation for model training. An Ultralytics YOLO detector and a lightweight 3D bounding-box regression network were trained on a high-performance cluster. Deployed on an NVIDIA Jetson Orin Nano, the perception stack operates at approximately 25 Hz, executing live camera capture, real-time YOLO detection with ID tracking, and per-object 3D regression to estimate relative distances. An onboard IBVR controller integrates a Kalman filter for measurement smoothing, a gradient-descent-based formation controller, and a centering proportional controller, with a fail-safe halting on lost visual contact. Experiments demonstrate reliable single-agent distance regulation, expose generalization limits in two-agent formation, and reveal angle-dependent perception errors in circular tracking tests. Results confirm the feasibility of embedded IBVR control and identify paths for enhancing perception robustness and scaling to heterogeneous multi-robot teams.**

I. INTRODUCTION

Multi-agent robotics systems are a subset of autonomous distributed systems in which multiple robotic agents act in unison to perform a collaborative task [1]. Multi-agent systems find applications in contexts such as surveillance, exploration, inspection, search and rescue, transportation, and logistics. These systems excel in applications characterized by complexity, dynamism, or spatial distribution, where the deployment of a single robot becomes impractically intuitive [2].

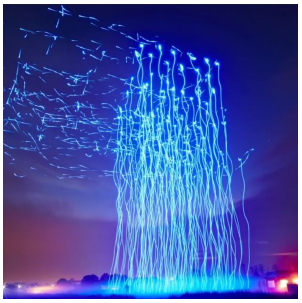
Multi-agent robotic systems require robust formation control methods to effectively coordinate and complete collective tasks in dynamic environments [3] [4]. Formation control approaches are categorized into centralized and distributed methods.

Centralized methods, as observed in Figure 2a, rely on a central server or leader agent to coordinate control throughout a system. Although these systems permit precise coordination, they experience limited scalability, high communication overheads, and vulnerability to single point failures. In contrast, distributed methods, as noted in Figure 2b, enable local decision-making among agents. This decentralized approach enhances system resilience while increasing real-time adaptability and fault tolerance, thus presenting a more scalable and flexible solution [3].

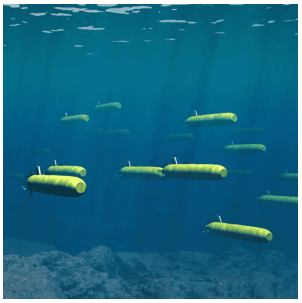
Despite the advantage of distributed control, existing methods are restricted by dependence on high-fidelity perception sensors, such as LiDAR, which are economically expensive to implement and computationally demanding in op-

(a) AMR manufacturing

(b) Drone inspection

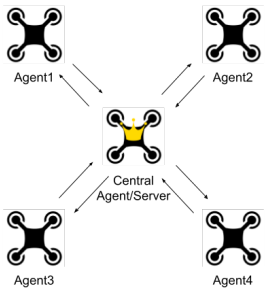


(c) Drone show coordination

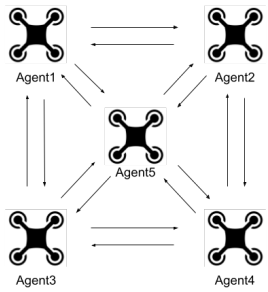


(d) Underwater robot swarm

Fig. 1: Applications of formation control: (a) AMR manufacturing, (b) drone inspection, (c) drone show, (d) underwater robot swarm.



(a)



(b)

Fig. 2: Visualization of (a) centralized formation control schema, and (b) decentralized formation control schema. [3]

eration [5]. Moreover, reliance on satellite-based positioning technologies limits operational flexibility. Many operational environments of interest, such as underwater or underground, are GPS restricted or denied [6]. These limitations hinder the scalability of real-world multi-robotic systems, highlighting the need for more adaptable control strategies.

IBVR offers a promising alternative by introducing an onboard vision-based approach that depends solely on a low-cost monocular 2D camera. This method estimates the relative position and orientation of neighboring agents based on their visible area and image-plane coordinates. Moreover, IBVR provides a rigid formation framework enabling agents to reach and maintain formations in 2D and 3D space, subsection II-C. IBVR represents a significant opportunity in the enhancement of multi-robotic system scalability, affordability, and feasibility [7].

IBVR is currently in the early stages of development and lacks real-world validation. IBVR presents an alternative method that could enable monocular control, representing a promising prospect to empower more scalable multi-agent systems. However, the current lack of an embedded system prevents agents from detecting and inferring the necessary information required to actuate real-world IBVR control. The absence of such a system results in a method that remains purely conceptual, functioning only in idealized simulation environments. [7]

Multiple options for an embedded system are being explored. Due to the nature of vision-based control, a real-time ML-CV object detection-based solution is proposed by the researchers behind IBVR. Although numerous applicable systems and models exist for similar tasks [8] [6], a model specifically tailored and directly implementable for the detection and positioning of neighboring agents for IBVR is required.

To bridge the gap between conceptual IBVR controllers and deployable multi-robot teams, an embedded perception–actuation pipeline is developed. This pipeline detects and localizes neighboring agents using a low-cost monocular camera, operates in real time, and closes the loop through an IBVR formation controller.

The main contributions of this paper are:

- Model Selection. Through the identification of the necessary inputs needs to actuate IBVR formation control, a machine learning-based computer vision perception pipepline is established to be implemented.
- Data-set Creation Pipeline: A end-to-end system for curating and creating data sets for model training is established and explained to be reproduced.
- Embedded System Implementation: A end-to-end deployment system of the selected models is designed, initial feasibility results and recommendations for research are given.

Section section II introduces the essential theory of the project, covering graph theory and rigidity, formation control, IBVR Distributed Formation Control, machine learning, and computer vision. Detailed descriptions of the selected models for system implementation are provided. The context of the testing environment, testing equipment, and software is outlined in section III. The system implementation is explored in section IV, starting with a breakdown of the Dataset Creation Pipeline end-to-end, followed by an establishment of model training methods, and concluding with an exploration of the embedded machine learning-computer vision IBVR control system. Initial feasibility experiments of the implemented system is then presented, with a complete discussion of results and an acknowledgement of the limitations of the project in section V. Finally, in section VI concluding remarks are offered and recommendations for future work and extensions are provided.

II. LITERATURE REVIEW

A. Rigidity Theory and Graph Formulations

This section reviews the fundamentals of rigidity theory and its graph-based formulation, providing the mathematical framework for ensuring and analyzing stable formations in multi-agent systems.

Consider a team of n robotic agents and a single agent i modeled as a kinematic point mass in d-dimensional Euclidean space. In practical applications, agents are modeled in either 2-dimensional (2D) or 3-dimensional spaces (3D).

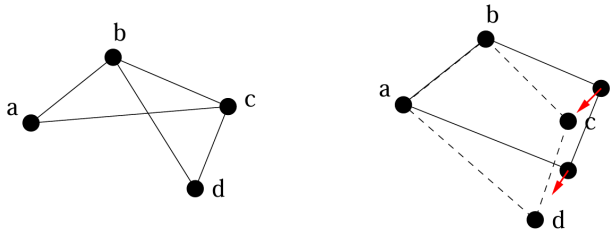
The spatial *configuration* $\mathbf{q}$ of all agents in a system is denoted by a nd-dimensional vector. Configurations considered in multi-agent systems are generic, implying that no special algebraic relationships exist among the coordinate positions $\mathbf{q}_i$ of the agents. This assumption prevents degenerate cases, such as agents being collinear in 2D or coplanar in 3D, which can affect actuation and stability of formation control [9] [10].

The sensing and communication information network of a multi-agent system is encoded into an undirected *graph* $G = (V,E)$, consisting of a set of vertices V and a set of edges E,where $V| = n$ and $|E| = m$. In the context of multi-agent and autonomous systems, the graph is referred to as the *interaction graph*, defined by associating an agent i with a vertex $v_i \in V$, and establishing that the agent i has access to relative information of an arbitrary agent j solely through an edge connection $(v_i, v_j) \in E$ [10].

A *framework* is defined as the pair $(G,\mathbf{q})$ that combines graph topology with the spatial layout of vertices. A graph can induce multiple frameworks, differentiated by specific spatial configurations and associated edge lengths [9].

A framework is said to be *rigid* if the only vertex motions that preserve all edge lengths correspond to a trivial rigid-body transformation: global translation or rotation. A *flexible* framework, by contrast, preserves edge lengths under continuous non-trivial deformations by adapting the overall shape of the framework. In the context of formation control, rigidity guarantees the enforcement of prescribed inter-agent distances in multi-agent systems [9] [10].

To refine the basic notion at the local or first-order level, infinitesimal rigidity is introduced. Infinitesimal rigidity is defined locally, or at the "micro-scale," if and only if no non-trivial infinitesimal motions exist that preserve inter-vertex distances. That is, no velocity vector $\mathbf{u} = (u_1, \ldots, u_n) \in \mathbb{R}^{dn}$



(a) Rigid framework (b) Flexible framework

Fig. 3: A rigid framework versus a flexible framework. [11]

exists such that the first-order change in edge lengths is zero, except for those motions corresponding to rigid body translations and rotations [11].

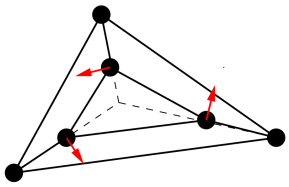


Fig. 4: A rigid, but not infinitesimally rigid framework. The arrows show the non-trivial infinitesimal motion. [11]

This definition is further formalized through the *rigidity matrix* $\mathbf{R} \in \mathbb{R}^{n\times m}$, representing how infinitesimal motions at each vertex affect edge lengths. The structure of the rigidity matrix is derived from the *incidence matrix* $\mathbf{B} \in \mathbb{R}^{n\times m}$, encoding the graph topology through the assignment of $+1$ and -1 to each endpoint. Extending to d-dimensional space as $\tilde{\mathbf{B}} = \mathbf{B} \otimes \mathbf{I}_d$, the incidence matrix expresses the relative displacement vector $\mathbf{z} = \tilde{\mathbf{B}}^\top \mathbf{q}$, from which the rigidity matrix can be constructed [12]. As described in [10], for an edge $\{i, j\} \in E$, the corresponding row of $R(\mathbf{q})$ contains non-zero entries only in the columns corresponding to vertices i and j, represented as:

$$\text{Row}_{\{i,j\}} = \begin{pmatrix} 0 \cdots 0 & (q_i - q_j)^T & 0 \cdots 0 & (q_j - q_i)^T & 0 \cdots 0 \end{pmatrix}$$

In other words, $R(\mathbf{q})$ is the Jacobian[1] of the function that maps the configuration q to its set of edge lengths. A framework is infinitesimally rigid if the only solutions to

$$R(\mathbf{q})\ \mathbf{u} = 0,$$

are trivial motions [11]. These trivial motions form the null space of the rigidity matrix and correspond exactly to rigid-body translations and rotations. In $\mathbb{R}^d$, translations contribute d motions, one along each axis, while rotations contribute $d(d-1)/2$, one for each plane of rotation. Altogether,

[1]For a vector-valued map $f : \mathbb{R}^{dn} \to \mathbb{R}^m$, the Jacobian is the $m \times dn$ matrix whose (k, ℓ) entry is $\partial f_k / \partial q_\ell$, encoding how small changes in q affect the outputs $f(q)$ [12].

$$d\ (translations) + \frac{d(d+1)}{2}\ (rotations)$$

dimensions are considered. Equivalently, the framework is infinitesimally rigid if

$$rank(R(p)) = dn - [d + \frac{d(d+1)}{2}] = dn - \frac{d(d+1)}{2}.$$

In this case, no non-trivial infinitesimal motions exist; therefore, the framework is classified as infinitesimally rigid. In the context of formation control, infinitesimal rigidity provides a practical and mathematical condition that guarantees locally stable and controllable formations against small perturbations [12] [10].

While infinitesimal rigidity guarantees that no small deformations of the formation are possible, it does not eliminate the possibility of non-congruent realizations that meet the same edge-length constraints. To address this ambiguity, the concept of generic global rigidity is introduced. Related to the global rigidity of a framework, generic global rigidity applies to the graph itself, independent of the exact coordinates, except for the aforementioned non-generic configurations. A framework is generically globally rigid if every other framework with the same edge lengths is congruent to it, meaning all configurations are identical up to rigid motions. The implication of generic global rigidity is that the shape of the framework is uniquely determined by the prescribed edge lengths [10]. For a graph to be generically globally rigid in $\mathbb{R}^d$, it must satisfy Hendrickson's necessary conditions for all d:

- $(d+1)$-vertex connectivity, the graph G can only be broken if $d+1$ vertices are removed from the graph,
- Redundant rigidity, the graph G remains infinitesimally rigid even if an edge is removed [10].

In the context of formation control generic global rigidity guarantees unique, and unambiguous convergence to a desired shape ensuring a controllable system [9].

B. Formation Control Problem

In the following section, key concepts for formulating and analyzing distributed formation control algorithms, as described in [10] and [12], are introduced.

1) Measurement Function: A *measurement function*

$$F : \mathbb{R}^{nd} \longrightarrow \mathbb{R}^m$$

maps the configuration $\mathbf{q}$ to a vector of relative quantities on the edges E of the interaction graph. The measurement function defines the sensed variables in the multi-agent system. Two common instances are:

- *Distance–based:*
  $d_{ij}(\mathbf{q}) = |q_i - q_j| \longrightarrow \mathbf{F_{dist}}(\mathbf{q}) = (\ldots, d_{ij}(p), \ldots)_{(i,j)} \in E$
- *Bearing–based:*
  $g_{ij}(\mathbf{q}) = \frac{q_j - q_i}{|q_j - q_i|} \longrightarrow \mathbf{F_{bear}}(\mathbf{q}) = (\ldots, g_{ij}(\mathbf{q}), \ldots)_{(i,j)} \in E$

The combination of distance, bearing, and other relative measurement concepts can be achieved through the design of a measurement function $F(\mathbf{q})$.

2) Converging to the Nominal Configuration: Error and Potential Functions: The desired configuration is specified by the *nominal configuration* $\mathbf{q}^*$, with the goal of formation being to drive the agents toward the nominal or desired measurement $F(\mathbf{q}^*)$. The *error function* is defined.

$$e(\mathbf{q}) = F(\mathbf{q}) - F(\mathbf{q}^*) \in \mathbb{R}^m,$$

which quantifies, through the measurement map, the extent of deviation of the current configuration from the nominal. The *potential function* is then introduced

$$U(\mathbf{q}) = \tfrac{1}{2}\|e(\mathbf{q})\|^2,$$

a nonnegative scalar 'energy' whose gradient with respect to each agent position produces an explicit control law aimed at driving the error to zero. In particular,

$$u_i = -\nabla_{q_i} U(\mathbf{q})$$

drives $e(\mathbf{q}) \to 0$, and hence $F(\mathbf{q}) \to F(\mathbf{q}^*)$.

3) Problem Formulation: With the above setup, the formal distributed formation control problem is defined in precise terms as follows:

Problem 1: Distributed Formation Control

Consider a team of n agents, each modeled as kinematic point masses in $\mathbb{R}^d$, with generic positions encoded in $\mathbf{q} \in \mathbb{R}^{nd}$. The agents interact according to the defined undirected interaction graph $G = (V,E)$, where $|V| = n$ and $|E| = m$. Given a nominal configuration $\mathbf{q}^*$, design for each agent i a velocity control law

$$\mathbf{u}_i = \mathbf{u}_i(\{F_{ij}(\mathbf{p})\}_{j:(i,j)\in E}),$$

satisfying:

1) The control $\mathbf{u}_i$ is distributed, dependent only on relative measurements from and agent i's neighbors in the graph $G(V,E)$.
2) The control is stable, small perturbations in the measurement function F induce small changes in the control output $\mathbf{u}_i$.
3) The control converges, that is, for any generic initial configuration $\mathbf{q}$ the control $\mathbf{u}_i$ will in time drive agents the the nominal configuration $\mathbf{q}^*$.

C. IBVR Distributed Formation Control

In this section, an Image Based Visual Relative (IBVR) approach to distributed formation control is presented as described in [7]. IBVR proposes a fully vision-based approach for distributed formation control [3]. Within IBVR, each agent, equipped with an onboard 2D-RGB camera, is modeled as a single integrator system with a corresponding relative reference frame. To reconcile the distance and positioning of neighboring robotic agents, IBVR utilizes known agent information and the perceived visible area of an assumed spherically shaped neighbor. This assumption is satisfied by several recent spherical designs of robots [13] [14] [15].

1) IBVR Sensing and Distance Inference: IBVR defines a rigid formation framework by comparing each agent's known spherical geometry to the perceived image area. The resulting discrepancy is utilized as a proxy to infer the inter-agent distance. Referencing Figure 5 the geometric relation between the spherical agent and its image in the image coordinate plane of the 2D camera is illustrated:

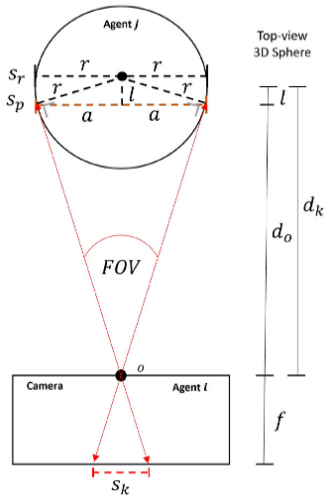


Fig. 5: Top-view plot of a pinhole camera system, projecting spherical object on to 2D-camera image-coordinate plane [7].

Here o represents the optical center (pinhole), f is the focal length measured in pixels, d_o is the distance from the pinhole to the surface of the object, l is the distance between the visible part of the sphere and its center, s_r is the area of the circle with true radius r, s_p is the area of the perceived circle with radius a, and s_k is the circular area of the neighboring agent "j" in the image-coordinate plane of the onboard camera of agent "i". Within the pinhole system, the ratio of the sphere's visible area to the true surface area satisfies:

$$s_{ratio} = \frac{s_p}{s_r} = \frac{d_k - r}{2d_k}.$$

The relationship between the distance from a neighboring agent and the measured visible area is expressed as follows:

$$\frac{f^2}{d_o^2} = \frac{s_k}{s_p} \tag{1}$$

Assuming a sufficiently large inter-agent distance $d_k >> r$, the visible area converges to the true area of a neighboring agent; $a \approx r$, and $s_p \approx s_r$. In this case, the visible area of the agent and its projection on the camera plane satisfy the following:

$$\frac{f^2}{d_o^2} \approx \frac{f^2}{d_k^2}, \tag{2}$$

$$\frac{s_k}{s_p} \approx \frac{s_k}{s_r}. \tag{3}$$

This approximation enables the inference of the distance d_k using image-based visual relative information (IBVR) of the visible area s_k, or following (1) (2) (3)

$$s_k \approx \frac{s_r f^2}{d_k^2}. \tag{4}$$

The relation between the IBVR variable s_k and the relative position z_k in 3D space is illustrated in Figure 6. Furthermore, the original assumption of $d_k >> r$ implies that spherical agents remain outside of collision range.

The following establishes the relation between the IBVR variable s_k and the relative position $\mathbf{z_k}$, referencing Figure 6 the pinhole camera model.

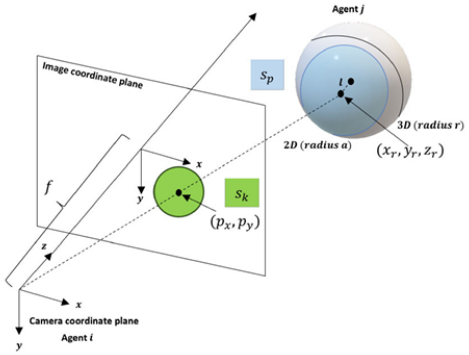


Fig. 6: An illustration of a pinhole camera model depicting the relative position of agent j compared to the camera coordinate plane of agent i [7].

The relative position of agent j from agent i is inferred by projecting the camera coordinate position (p_x, p_y)

$$z_k = \begin{bmatrix} \frac{z_r}{f} p_x & \frac{z_r}{f} p_x & z_r \end{bmatrix}^\top, \tag{5}$$

where $z_r = \sqrt{\frac{s_r}{s_k}} f$.

2) IBVR Rigid Framework: Based on image-based distance inference in (1) - (4), IBVR measurements are utilized to define rigid formation shapes $\mathbf{q}$. Utilizing the IBVR variable s_k, a measurement map, as discussed in subsubsection II-B.1, is defined by

$$F_{\text{area}}(\mathbf{q}) = [\dots s_k \dots]^\top, \forall k \in \{1, \dots, m\}.$$

Based on (4), the expression above can also be expressed as

$$F_{\text{area}}(\mathbf{q}) = [\dots \frac{s_r f^2}{d_k^2} \dots]^\top. \tag{6}$$

To describe the associated rigidity matrix $\mathbf{R_{area}}(\mathbf{q})$, as described in subsection II-A, the Jacobian of (6) is taken as follows:

$$\mathbf{R}_{\text{area}}(\mathbf{q}) = \frac{\partial F_{\text{area}}(\mathbf{q})}{\partial \mathbf{q}} \tag{7}$$

Then, through an application of chain rule factorization to (7)

$$\mathbf{R_{area}}(\mathbf{q}) = \frac{\partial \mathbf{f}_{\text{area}}(\mathbf{d_k})}{\partial \mathbf{Q}} \frac{\partial \mathbf{Q}}{\partial \mathbf{q}} = -\mathbf{T}(\mathbf{d})\ \mathbf{R_{dist}}(\mathbf{q})$$

where $\mathbf{Q} = [\ d_1^2, \dots, d_m^2]^\top \in \mathbb{R}^m$ represents the vector of squared inter-agent distances, and $\mathbf{T}(\mathbf{d}) = diag\{s_r f^2 d_k^{-4}\}_{k \in m} \in \mathbb{R}^{m \times m}$ denotes the diagonal scaling that arises when differentiating the area measurement from (4). As $d_k \gg 0$, the matrix $\mathbf{T_d}$ is positive definite, which implies $rank\ (\mathbf{R}_{area}) = rank\ (\mathbf{R}_{dist})$. This demonstrates the one-to-one relationship between infinitesimally rigid formation shapes defined using a distance-based measurement function $F_{\text{dist}}(\mathbf{q})$ and an area-based measurement function $F_{\text{area}}(\mathbf{q})$.

Now consider a nominal target visible-area vector

$$s_k = [s_k^*]_{k \in m},$$

where s_k^* represents the desired visible area for each edge k. A framework $(G, \mathbf{q})$ achieves the nominal configuration $\mathbf{q}^*$ if

$$F_{\text{area}}(\mathbf{q}) = \mathbf{s}^*.$$

Referencing subsection II-A, it is important to recall that the kernel of the rigidity matrix

$$ker\{\mathbf{R}_{\text{area}}(\mathbf{q})\} = \{\mathbf{u} \in \mathbb{R}^{3n} \mid \mathbf{R}_{\text{area}}(\mathbf{q})\mathbf{u} = \mathbf{0}\} \tag{8}$$

always contains the six trivial rigid-body motions: three translations and three rotations. The framework is infinitely rigid only when trivial motions are the only infinitesimal motions $\mathbf{u}$ in the kernel.

By the rank-nullity theorem for a $m \times nd$ matrix,

$$dim[ker\{\mathbf{R}_{\text{area}}(\mathbf{q})\}] + rank\{\mathbf{R}_{\text{area}}(\mathbf{q})\} = nd,$$

where the first term in the sum counts the number of unique infinitesimal motions, and the latter term counts the independent measurement constraints for d dimensions. For the 3-dimensional system considered in IBVR, this implies $dim[ker\{\mathbf{R}_{\text{area}}(\mathbf{q})\}] = 6$ and

$$rank\{\mathbf{R}_{\text{area}}(\mathbf{q})\} = 3n - 6. \tag{9}$$

Conversely, if (9) holds, then $dim[ker\{\mathbf{R}_{\text{area}}(\mathbf{q})\}] = 6$, indicating that the null space can contain nothing beyond trivial motions, which is analogous to stating that the framework is infinitesimally rigid.

The rigid IBVR framework is finalized by defining the set of all configurations that realize the target perceived visible areas $\mathbf{s}^*$ as

$$\mathcal{D} = \{\mathbf{q} \mid F_{\text{area}}(\mathbf{q}) \ = \ \mathbf{s}^*\}.$$

Throughout this report, a set of desired visible areas $\mathbf{s}^*$ is considered, such that the corresponding rigidity matrix of the visible area $\mathbf{R_{area}}$ has rank $3n-6$, and the formation shape defined by the framework $(\mathbf{G}, \mathbf{q}^*)$ is infinitesimally rigid.

[1]Here, the Jacobian is taken of the measurement map $F : \mathbb{R}^{3d} \longrightarrow \mathbb{R}^{\times}$ and linearly approximates how small changes in the configuration $\mathbf{q}$ affect the measurement $F(\mathbf{q})$.

3) IBVR Distributed Formation Control: To discuss the actuation of distributed formation control using IBVR information, consider single-integrator agents

$$\dot{\mathbf{q}} = \mathbf{u_i}, \quad \forall \mathbf{i} = 1, ..., N, \tag{10}$$

where $\mathbf{q_i} \in \mathbb{R}^3$ denotes the position and $\mathbf{u_i} \in \mathbb{R}^3$ represents the input velocity. Next, a gradient-based controller is formulated utilizing an error based on the visible area.

$$e_k = s_k^* - s_k, \tag{11}$$

where s_k^* represents the desired visible area of the neighbor j in the image coordinate plane of agent i. Consider the related potential function

$$V(\mathbf{e}) = \frac{1}{2} \sum_{k=1}^{|m|} \kappa_k (s_k^* - s_k)^2, \tag{12}$$

where $\mathbf{e} = col_{k \in \mathcal{E}}$, κ is a positive constant for every $k \in 1, ..., m$. Referencing the standard gradient-based control law

$$\mathbf{u_i} = -\sum_{k=1}^{|\mathcal{E}|} \frac{\partial}{\partial q_k} V(\mathbf{e}),$$

the distributed control law for each agent i is obtained through direct substitution of (12) into the previously mentioned gradient-based control law. Thus,

$$\mathbf{u_i} = -\sum_{j \in \mathcal{N}} \frac{\mathbf{z_{ij}}}{||\mathbf{z_{ij}}||s_r f^2} s_{ij}^2 \kappa_{ij} e_{ij}, \tag{13}$$

where $e_{ij} = e_k$, $\kappa_{ij} = \kappa_k$, $\mathbf{z_{ij}} = \mathbf{z_k}$, and $s_{ij} = s_k$ with $(i, j) = E_k$. The above distributed control law utilizes only visible-area (IBVR) information, obtaining the unit vector direction of the centroid of a neighboring agent j and through $\frac{z_{ij}}{s_{ij}}$, to achieve the desired formation shapes.

4) Vision Requirements for IBVR: Building on the theoretical IBVR framework of [7], a practical, end-to-end perception system must extract four core outputs from a single 2D-RGB camera:

1) Localisation. The perception system must infer the image-plane coordinate of neighbouring robotic agents. This is essential for the formulation of z_k from (5), which by reference to Figure 6 maps the image coordinate positions to the 3D camera coordinate plane. Accurate mapping also depends on the intrinsics of a properly calibrated camera, as shown in Figure 5.
2) Classification. To continuously track relevant robotic agents and ignore irrelevant objects in camera frames, the perception stack must assign each detection in the camera frame a semantic label relating its relevance to IBVR control.
3) Identification. To continuously assign a tracking id to relevant detected objects, such that the controller does not switch targets mid-operation and that relevant metrics for control and perception can be recorded.
4) Size Estimation. IBVR uses s_k, reference (4), as a proxy for distance

$$s_k \approx \frac{s_r f^2}{d_k^2} \implies d_k \approx f\sqrt{\frac{s_r}{s_k}}.$$

Thus, to actuate formation control, the perception pipeline must measure the apparent spherical cap area s_k. In nonspherical cases a consistent method must be established to infer a reliable spherical equivalent of the agent geometry, and be robust in non-symmetrical agent cases where view-angle changes means a direct measurement of the surface area is inconsistent.

To transform raw image frames into four abstract outputs, a perception pipeline is required that first identifies candidate regions of interest in the image coordinate pixel domain and subsequently infers the high-level attributes necessary for the IBVR control law within the continuous 3D camera coordinate domain. Modern machine learning-based computer vision algorithms deliver these capabilities effectively.

D. Machine Learning

Below is a brief introduction to Machine Learning (ML), fundamental in developing both the theory and motivation for a ML-based perception stack implemented for IBVR formation control.

Machine learning is a subset of Artificial Intelligence (AI); it is a field that studies algorithms capable of learning patterns from data rather than from hard-coded rules. ML covers a wide variety of methods and approaches, with the intention of creating adaptive and generalizable algorithms that can interpret and operate in a large variety of contexts. A *model* is a formal mathematical expression relating input information to output predictions. The expression is governed by a set of *parameters*. For example, in the case of linear regression, this could refer to the slope of the fit line, which is adjusted during the *training* process that optimizes model performance. Model training is the process in which algorithms iterate through data to learn the latent patterns within; it is the process of generating the model's mathematical expression [16].

Many traditional ML models consist of the same basic components and developmental steps. These models heavily rely on feature engineering, where human-designed input data or *features* are fed into relatively simple models characterized by a low order of parameters, such as decision trees, support vector machines, and linear regression. Optimization schemas are then employed to train the model, evaluating performance against explicit metrics. This overall modular approach results in a workflow wherein data preprocessing, feature engineering, model training, and evaluation are perceived as distinct and isolated steps [16].

Deep-learning ML models, such as Neural Networks (NN), diverge from traditional ML approaches. These models learn hierarchical representations of data directly from raw inputs, reducing or eliminating the need for feature engineering. NNs utilize multiple layers of computation to automatically infer relevant features during training. In the context of

low-level image processing, the model learns edges, corners, and color gradients related to an object, using these as building blocks to progress toward higher-level features. This process encompasses more meaningful patterns, including textures, shapes, and eventually entire objects and concepts. NNs often require large amounts of data and significant computational power, relying on mechanisms and large-scale optimization to tune a relatively high order of parameters [17].

E. Computer Vision and Object Detection

To further elaborate on the motivation for a machine learning-based computer vision system embedded into IBVR, Computer Vision (CV) models are introduced. Subsequently, a survey is provided that examines modern CV technologies spanning 2D detection, tracking, 6D pose, and depth estimation, which deliver the four sensor outputs discussed in subsubsection II-C.4.

1) Convolution al Neural Networks: CV is a subset of ML that leverages concepts of neural networks to enable computers and systems to derive information from visual inputs.

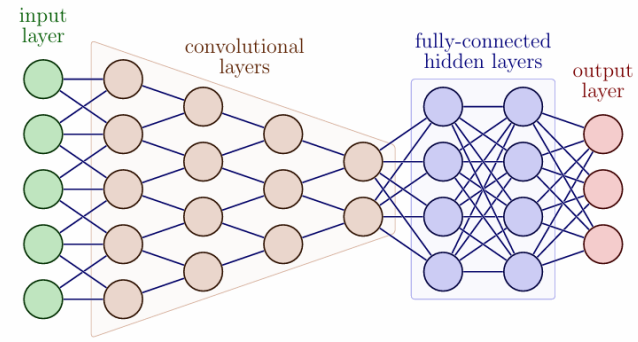


Fig. 7: Topology of a Convolutional Neural Network [17].

Figure 7 presents the topological view of a *Convolutional Neural Network* (CNN), the most prevalent NN class for vision-based machine learning. The trapezoidal section is crucial in representing the conversion of higher-dimensional input data, such as high-definition images or frames of a video, into the lower-dimensional outputs required for IBVR distributed formation control [17].

Fig. 8: Object detection running on sample artwork and natural images from the internet [18].

2) Object Detection: Localization, Classification and Identification: Figure 8 illustrates the extension of ML-CV models to 2D object detection models. These models possess the capability to classify objects from an input image or real-time video stream and localize positions in the image plane using a 2D bounding box. Additionally, these models can be extended to track specific objects, either through explicit training or alternative methods.

Most contemporary object detectors, particularly those designed for real-time applications, operate under a *closed-set* assumption: only a certain set of objects is relevant for the application. Consequently, these detectors can only detect the classes explicitly provided in the dataset. Large labeled datasets are required for training these models; each image has a corresponding text-based annotation containing the relevant data fields for detection. Like other NNs feature extraction is automatic; the associated spatial and semantic patterns related to the bounding box and object class are learned directly from raw data **arani2019** .

Open-set object detectors extend traditional object detection by recognizing objects from a fixed, known set of classes while also detecting objects outside established categories. Recently, *Open Vocabulary Object Detectors* (OVOD), which utilize a joint image-text embedding space [2], have demonstrated the capability to make inferences based on provided text prompts. This emergent technology has implications for the creation of more flexible detection systems and assistance with dataset creation [19][20].

.

3) Depth and Size Estimation Methods: While 2D object detectors provide image-plane localization and classification, actuating IBVR formation control also requires metric-scale information of the perceived agent, namely s_k. Alternatively, a direct prediction of depth z_k would circumvent the need for size estimation methods, allowing for the creation of a direct depth-based camera-actuated formation control method. However, a reliable direct z_k approximation in parallel to a s_k inference could also support IBVR information inference, creating a more robust depth inference from s_k.

6D pose inference and monocular depth inference are introduced as methods for depth and size estimation. The former enables direct size measurement, though not implicitly s_k, while both methods provide depth z_k relative to the camera-coordinate frame.

a) 6D pose inference & 3D Bounding Boxes : Referencing Figure 9, the concept of a 3D bounding box is introduced. A single image frame can be processed through a dedicated neural network (NN) that predicts the eight corners of an enclosing 3D bounding box, along with the associated object-centric coordinate frame. This output preserves complete 6D-pose information:

- Position (X,Y,Z) of the object's centroid i.e 3D *Localization* in camera-coordinate plane,

[2]A joint high-dimensional vector space where both image regions and text prompts are mapped to embeddings such that semantically related visuals and words lie close together and can be correlated [19][20]

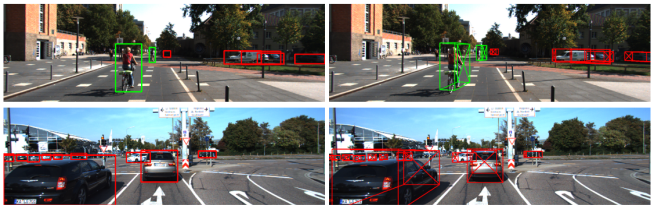


Fig. 9: Example of 3D bounding boxes with direct comparison to 2D bounding boxes [21].

Fig. 10: Sample images from the 3D Movies dataset with their inverse depth maps [23].

- Orientation (α, β, γ) expressed as Euler angles or a rotation matrix.

Furthermore, the predicted 3D box directly yields the predicted dimensions associated with the object in the camera coordinate frame. This satisfies the IBVR *Size-Estimation* requirements.

b) Monocular Depth Inference: In parallel or as an alternative sensing approach, monocular depth estimation networks generate a dense per-pixel depth map. Depth values may be averaged within each 2D bounding box or recovered at the center of the box, enabling the prediction of a robust z_k value for each detected agent [22].

F. Model Selection

With the theoretical foundation established for camera-based perception and key-related computer vision architectures capable of delivering the required four-sensor outputs, as defined in subsubsection II-C.4, surveyed. The following section addresses the selection of specific models for the IBVR perception pipeline.

1) YOLO for 2D Localization, Classification and Identification: YOLO ("You Only Look Once"), presented by [18], approaches 2D object detection as a single end-to-end regression. A single CNN predicts bounding box coordinates, object class labels, and associated class probabilities directly from a raw image frame in a single forward pass. By simplifying the previous state-of-the-art two-network pipeline [24] and collapsing region proposal and classification into one network, YOLO eliminates costly multi-stage pipelines,

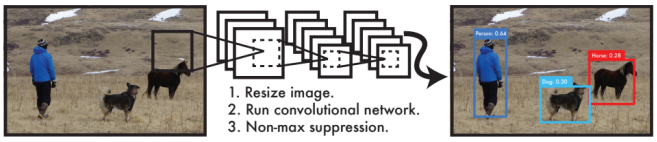


Fig. 11: The YOLO Detection System. (1) Resize input image, (2) run single CNN on image, and (3) thereshold the resulting detection by model confidence [18].

thereby enabling efficient real-time object detection inference.

Since debuting in 2015, YOLO has undergone rapid iterations that have improved speed, accuracy, and usability while significantly evolving the surrounding software ecosystem. Original YOLO implementations (v1-v4) operated on Darknet, utilizing a custom C and CUDA framework[3]. Modern YOLO implementations (v5-v11), considered in this project, were redesigned for PyTorch[4] by Ultralytics, resulting in a unified Python Ultralytics package and Command Line Interface (CLI) for seamless model implementation workflows[25].

More specifically, Ultralytics YOLOv8 and YOLOv11 have been selected for this project. These models decompose the inference network into three components:

1) Backbone. High-level CNN based feature extractor that encodes raw images into hierarchical feature maps.
2) Neck. A dynamic feature fusion model that refines and combines feature maps so both small and large agents can be detected reliably.
3) Head. A light weight prediction layer, that at each spatial location, outputs bounding box coordinates, object class labels, and associated class probabilities.

To explicitly link IBVR vision requirements to Ultralytics YOLOv8 and YOLOv11, the direct outputs of the models are defined as follows:

- Class ID. Number ID associated to a class or label,
- X-Center. Central x position of detected object in image coordinate plane,
- Y-Center. Central x position of detected object in image coordinate plane,
- Width. Width of predicted bounding box around central x coordinate.
- Height. Height of predicted bounding box around central y coordinate.

Alignment with the IBVR vision requirements is evident: the box coordinates provide pixel-level localization of each agent; the predicted labels fulfill classification. When integrated with a tracking-by-detection module [26], YOLO's

[3]C is a general-purpose, compiled programming language known for its efficiency, low-level memory control, and portability across platforms. CUDA is NVIDIA's extension to C/C++ that provides a parallel computing platform and API, enabling the development of code that runs directly on NVIDIA GPUs for massively parallel, high-performance computations.

[4]PyTorch is an open-source machine learning library for Python that provides tensor computations with GPU acceleration and a dynamic computation graph, making it intuitive for building and training deep neural networks.

per-frame detections can be linked over time to ensure consistent identification.

a) YOLO variants: Within the considered Ultralytics YOLO models, sub-variants include nano (n), small (s), medium (m), large (l), and extra large (x). The selected variant dictates model parameter capacity, with n possessing the smallest number of parameters and x possessing the largest. Model size directly influences inference speed and accuracy of the applied YOLO model, as indicated in the figure below.

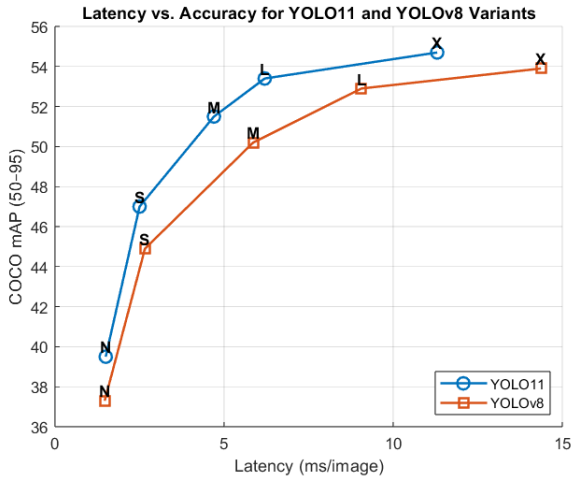


Fig. 12: YOLOv8 and YOLOv11, performance comparison accuracy versus processing time by Ultralytics [25].

Only the n and s variants of the YOLOv8 and YOLOv11 models are considered in this project. This selection is based on the controlled laboratory deployment context, wherein the robustness of larger models is unnecessary and may hinder IBVR control due to limitations in inference speed. Referencing Figure 12, the performance trade-off when moving from S to M size models yields diminishing returns relative to the increased latency.

2) 3D Bounding-Box Regression for Size-Estimation: Building on [21], the complete 3D bounding box of an object is regressed, encompassing the metric size and distance of the detected object within local camera coordinates. Henceforth, this report refers to the described method as "3D Bounding-Box Regression Network" or "3DBB Regression Net" for short.

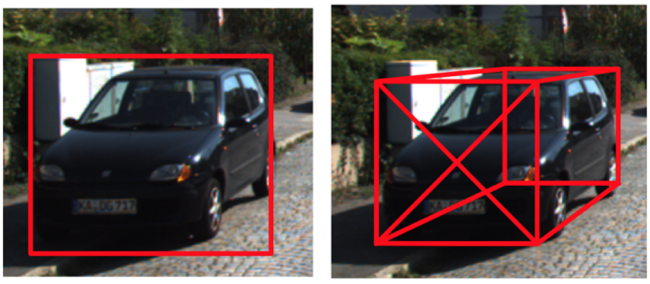


Fig. 13: Image crop to 3D Bounding Box [21].

3D bounding-box regression is achieved using a single image crop of the object, as shown in Figure 13. The model directly outputs the following:

- Orientation (α, β, γ). Using a MultiBin formulation, the heading angle is first "binned" into a discrete sector of the full rotational range and then the residual angle is resolved within each bin. This discrete to continuous approach reduces ambiguity and improves the orientation inference accuracy.
- Dimensions (h, w, l). Using known measurements of detected object classes, the model learns how much each trained instance differs to the average. Then it is able to output a predicted based on training that minimizes the absolute error of those differences.

With the orientation R and dimensions D, the eight corners of the 3D bounding box are computed in the object-relative coordinate frame. The final translation $T = (X, Y, Z)$ is determined by minimizing the projection error of the 3D box corners against the edges of the image crop, under known camera intrinsics. This constraint outputs an estimate of the object's centroid depth Z and lateral offsets (X, Y) in the camera-coordinate frame.

Together (R, T, D) provides a full 6D-pose output. From this output, two quantities necessary for formation control actuation are extrapolated. First, size estimation is considered; the box dimension D can be utilized to compute the apparent surface area of a sphere corresponding to the object. Furthermore, the dimensions D remain consistent at any object viewing angle, and by extension, the inferred surface area is valid even for non-spherical, non-symmetric objects. Second, the depth Z of the object's centroid is determined based on the direct depth to the object from the camera's optical center.

The original formulation of the 3DBB Regression Net was intended for single image inference, not for real-time applications utilizing a constant video stream. To establish a complete real-time pipeline, YOLO can be utilized to continuously feed single image 2D crops (bounding boxes) into the 3D Bounding Box Regression Network, facilitating real-time 6D pose inference.

a) 3D Bounding-box Regression Network Variants: From a theoretical standpoint, the 3D regression task is inherently more computationally expensive than YOLO's lightweight approach for 2D object detection. Consequently, the 3D BB Regression Net is anticipated to be the bottleneck of the perception pipeline. To optimize the network for real-time IBVR information inference,considerations include NN backbone selection, orientation bin count, and input crop size.

Various established neural network (NN) backbones are available for adaptation and deployment within regression networks. Currently, VGG-19 serves as an effective feature extractor for the 3D Bounding-box Regression Network. However, a major drawback of VGG-19 is its weight and associated computational expense, which may render it unsuitable for real-time inference. Consequently, various relatively lightweight backbones are considered for deployment

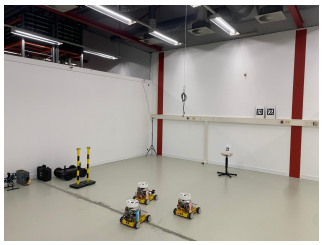
[27].

The orientation bin count refers to the number of discrete sectors into which the rotation range of the object is divided for the two-step orientation inference process. Fewer bins simplify residual regression and reduce overall model weight; however, this approach can negatively impact orientation inference accuracy. Conversely, an increased number of bins enhance angle estimates but also increase inference time.

The input crop size directly influences the fidelity of the feed-cropped images to the regression network. Similar to the orientation bin count, a higher crop fidelity (448×448 pixels) leads to increased accuracy but results in inflated inference times, while lower crop fidelity (224×244 pixels). Unlike the orientation bin count, the input image does not increase the number of layers in the model; rather, it increases the amount of data that requires processing. The selection of the crop size must be compatible with the fed input image.

III. SYSTEM CONTEXT: TESTING ENVIRONMENT, HARDWARE & SOFTWARE

A. Testing Environment: DTPA Laboratory



(a)

(b)

(c)

Fig. 14: DTPA Laboratory Testing Environment.

Testing of IBVR will be undertaken in the DTPA laboratory, reference Figure 14 for related images. Figure 14a and Figure 14b present the closed testing environment. Figure 14c presents the observation workspace.

B. System Hardware

a) Nexus Mobile Robot: The Nexus is a 2D-planar mobile robot equipped with four mecanum wheels, enabling omnidirectional motion without the need for steering linkages. This capability simplifies motion control and allows for precise alignment with visual targets as dictated by the IBVR controller.

Low-level control operates on the Nexus's local computer, which runs Ubuntu 20.04. The underlying code is built using a ROS Noetic network. For further insight and direct code access for the Nexus base, refer to the `nexus_base_ros` package [28]. The Nexus onboard computer facilitates remote access via a wireless connection.

The Nexus dimensions, including all onboard components and sensors, are:

- Width: 35*cm*
- Height: 36*cm*
- Length: 40*cm*

b) NVIDIA Jetson Orin Nano: To meet the real-time inference demands of the perception stack, the computational power of the NVIDIA Jetson Orin Nano is leveraged. The Orin Nano features a dedicated Ampere GPU with full CUDA support. This additional GPU compute headroom—far beyond the capabilities of a standard laptop's CPU or onboard NexusAMR computer—enables the execution of both YOLOv8/v11 detection and the 3DBB Regression Net with enhanced inference speed.

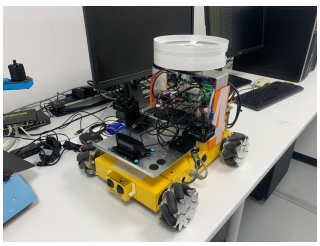
Moreover, the Jetson Orin Nano's compact, lightweight form factor renders it ideal for deployment onboard the Nexus AMR. Tight dimensions and low power draw facilitate seamless integration into the robot's chassis, providing onboard, end-to-end perception without the bulk or cooling requirements of desktop-class GPUs.

c) Logitech C920 Webcam: The Logitech C920 is a low-cost commercially available desktop camera that provides HD images (1080×1920 pixels) at 30 frames per second. The camera is mounted onto the NexusAMR using a passive camera-stabilizing unit.

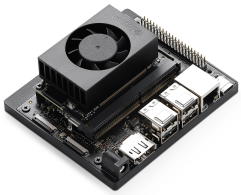
d) Marvelmind Nav Local GPS: An indoor acoustic-based local GPS system leverages MarvelMind's proprietary beacons to precisely measure distances and bearings between the beacons. Four beacons are positioned around the DTPA laboratory for general localization, while two beacons are mounted on the front of the Nexus. This setup provides a global position and measures global rotation.

e) Logitech F710 Controller: Compatible with Martin's `nexus_base_ros` code for Nexus operation, the Logitech F710 functions as a commercially available remote control device utilized for teleoperation of the Nexus mobile robot.

f) Harbok Computation Cluster: The Harbrok functions as the high-performance computation cluster of the University of Groningen, serving as the backbone for model training in this project. Operated through SLURM[29], multiple users gain access to high compute nodes for model training, simulation, and parallel data processing.



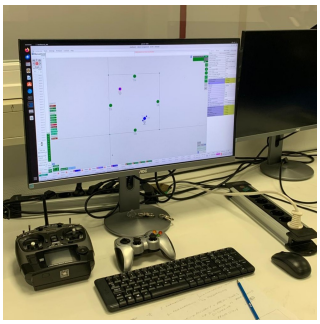
(a) Nexus Mobile Robot



(b) NVIDIA Jetson Orin Nano

(c) Logitech C920 Webcam

(d) Marvelmind Beacon Sensor



(e) Marvelmind System UI

(f) Logitech F710 Remote Controller

Fig. 15: Hardware components of the IBVR system: (a) Nexus mobile robot; (b) NVIDIA Jetson Orin Nano; (c) Logitech C920 Webcam; (d) Marvelmind Beacon Sensor; (e) Marvelmind System UI; (f) Logitech F710 Controller.

IV. SYSTEM IMPLEMENTATION

With the relevant models and testing context established, the report shifts focus to the direct implementation steps of the perception stack for practical IBVR formation control.

A. Dataset Collection & Annotation

To deploy a capable perception system for the experimentation context established in section III, a custom dataset must be curated and collected for the selected Ultralytics YOLOv8, Ultralytics YOLOv11, and 3D Bounding Box Regression Network models.

1) Dataset Requirements: The following section explores the explicit dataset requirements for the selected models to determine the data necessary for training a relevant model for the deployment context.

All considered models share the same overarching data structure for training: an image and associated text-based annotations. Annotation formats vary from model to model; they relate essential numerical information to the corresponding image. An annotation comprises multiple fields, with each field representing a distinct piece of relevant information.

a) Ultralytics YOLO: Both Ultralytics YOLOv8 and YOLOv11 utilize the same image and annotation format. The input annotations required for YOLO training match the outputs established in subsubsection II-F.1: Class Id, X-Center, Y-Center, Width, and Height. The latter four labels related to the 2D bounding boxes are normalized to the image frame.

Fig. 16: YOLO formatted normalized bounding box example for 2 persons (class 0) and a tie (class 27) [25].

TABLE I: Corresponding label file for 2 persons (class 0) and a tie (class 27) [25].

class_id	center x	center y	width	height
0	0.481719	0.634028	0.690625	0.713278
0	0.741094	0.524306	0.314750	0.933389
27	0.364844	0.795833	0.078125	0.400000

Referencing Figure 16 and Table I above, an example of the 2D bounding box label for YOLO, along with a visualization of the corresponding 2D bounding box, is presented.

b) 3D Bounding Box Regression Network: The 3D BB Regression Net adheres to annotations formatted according to the KITTI dataset, a widely used ML-CV benchmark dataset for autonomous driving contexts. Developed by the Karlsruhe Institute of Technology, primary applications include tasks such as 3D object detection and 3D tracking [30].

Table II lists the 15 KITTI annotation fields. In the custom dataset, both Truncated and Occluded are hard coded to 0 under controlled deployment conditions. KITTI's 2D bounding box format (left, top, right, bottom) differs from YOLO's (normalized center x, center y, width, height), necessitating a conversion step.

2) ROS-Based Data Collection System: To satisfy the data set requirements defined in subsubsection IV-A.1, synchronized images and corresponding positional measurements are necessary for the creation of KITTI style labels. Figure 17

TABLE II: KITTI-style 3D Bounding Box Label Fields [30].

Field	Description
Type	Type of object (e.g. Car, Pedestrian), not an ID number.
Truncated	Float in [0,1], fraction of object outside the image.
Occluded	Integer {0,1,2,3}, level of occlusion (0 = fully visible, 3 = largely occluded).
Alpha	Observation angle of object relative to camera axis (radians).
Bounding Box Left	2D left (x) coordinate of the image-plane bounding box (pixels).
Bounding Box Top	2D top (y) coordinate of the image-plane bounding box (pixels).
Bounding Box Right	2D right (x) coordinate of the image-plane bounding box (pixels).
Bounding Box Bottom	2D bottom (y) coordinate of the image-plane bounding box (pixels).
Height	3D object height (meters).
Width	3D object width (meters).
Length	3D object length (meters).
X	Horizontal distance from camera to object center (meters).
Y	Vertical distance from camera to object center (meters).
Z	Depth (forward) distance from camera to object center (meters).
RotationY	Rotation around the Y-axis (yaw) in camera coordinates (radians).

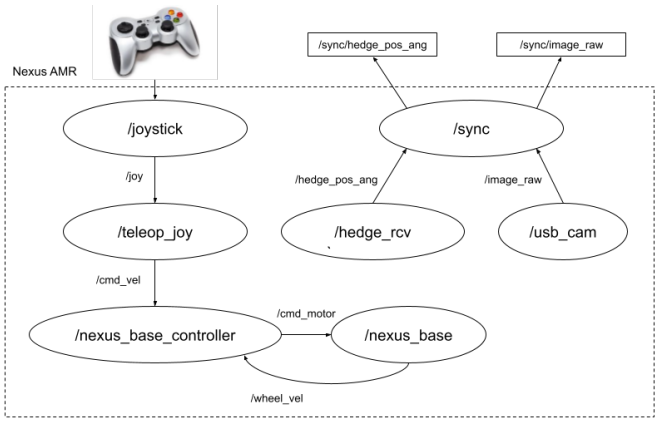


Fig. 17: RQT graph of implemented synchronized image and pose collection pipeline.

provides an overview of the RQT graph of the ROS data collection system utilized for synchronized data collection.

The ROS system functions as follows:

1) Teleoperation. Using the Logitech F710 wireless controller, the Nexus is teleoperated to navigate the DTPA laboratory environment while pointing the onboard camera toward a static Nexus agent. The Logitech F710 operates on the `/joystick` node and outputs messages to the `/teleop_joy` node. Here, raw joystick inputs are converted to velocity command messages `/cmd_vel` that the base Nexus controller nodes, `/nexus_base_controller` and `/nexus_base`, use to drive the motors.
2) Image Capture. Using ROS's integrated USB camera package (`usb_cam`), the Logitech C920 is implemented in the `/usb_cam` node. This node publishes raw images `/image_raw` at configurable frame rates and resolutions. Compressed steams are available in formats such as `png` or `jpg` to reduce bandwidth.
3) Global Pose Capture. Using Marvelmind's ROS noetic package, the `/hedge_rcv` node records the pose of the Nexus in a global reference frame. It publishes the position and heading data on `/hedge_pos_ang`, while the static recorded agent's pose is recorded once at initialization and taken as constant. The pose of static agents is recorded as the relative distances and bearings between the camera and the agent pictured are needed to fill the fields defined in Table II.
4) Data Sync and Recording. To synchronize the recorded data so that the recorded image and pose are representative of each other, the `/sync` node is introduced. Applying ROS's message filters package, the node subscribes to the image and pose topics, and republishes the messages if they are received within a selected time frame. Messages are dropped unless their timestamps fall within the synchronization window. The synchronized messages, `/sync/hedge_pos_ang` and `/sync/image_raw`, are then recorded using ROS bags, ROS's in built message recording format. Uncompressed image topics can cause dropout over the wireless link; using local SSD recording is recommended.

3) Automatic 2D Bounding-Box Annotation Pipeline: subsubsection IV-A.1 establishes the need for precisely correlated images and corresponding bounding-box annotations for selected models; traditional manual labeling scales poorly for large datasets. To accelerate the generation of high-quality 2D bounding boxes for the Nexus agent, a semi-automated pipeline based on Grounding-DINO was integrated via Autodistills API [19][31].[5]

First, each raw frame is passed through the Grounding-DINO inference endpoint with a related text prompt. The model returns a set of candidate region proposals ranked by grounding confidence. These proposals are then:

1) Filtered & Thresholded. Proposals with confidence below a chosen threshold are discarded to reduce false positives.
2) Normalized to YOLO Format. Each remaining box $(x_{\min}, y_{\min}, x_{\max}, y_{\max})$ is converted to YOLO $(x_{\text{center}}, y_{\text{center}}, w, h)$, where w is the width of the box and h the height of the box.
3) Exported. Finalized initial annotations are saved as text files with the corresponding title to associate them with image frames.

The Autodistill Grounding-Dino system undergoes fine-tuning through prompt engineering, which includes the addition of adjectives, context phrases, or multi-prompt ensembling. This approach facilitates the tuning of recall versus

[5]Grounding-DINO internally uses a vision backbone plus feature-pyramid neck to convert image patches into "tokens" that attend jointly with text embeddings in a cross-modal transformer head. This detail can be treated as a black box when invoking via the API [19].

precision trade-offs without requiring model retraining. The text prompts utilized for detecting the Nexus mobile robot with Grounding-DINO are presented below:

TABLE III: Text prompts used for Grounding-DINO inference

Prompt
"a yellow mobile robot with white and blue components"
"a yellow autonomous robot with mecanum wheels"
"a yellow logistics robot on the floor in the foreground"
"a small yellow delivery robot with sensors on top"
"a robot with a yellow base and white upper structure"

The Grounding-DINO outputs from the aforementioned text prompts, reference Table III, and the test data set of images are presented below:

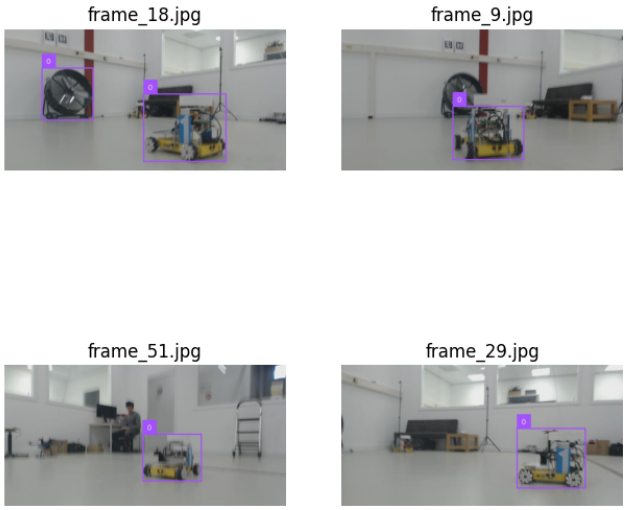


Fig. 18: Grounding-DINO output for Nexus in DTPA lab

Referencing Figure 18, Nexus is consistently detected; however, noise and additional false detections appear in certain frames. This observation establishes the necessity for manual processing. Roboflow, a widely used online application for managing computer vision datasets, was utilized to enhance dataset quality and consistency. The annotation workspace enables visualization of bounding boxes over image frames, facilitating efficient visualization and editing [32].

Leveraging Grounding-DINO in this manner enables the annotation pipeline to scale efficiently to thousands of frames, ensuring consistency in complete YOLO formatted 2D bounding box annotations.

4) Text Annotation Generation: At this stage, synchronized poses, images, and finalized YOLO 2D boxes have been collected. To prepare inputs for the 3D bounding-box regression network, post-processing of these data is required to extract and reformat the KITTI-style fields listed in Table II. The following workflow outlines the transformation of the collected data into the respective 3D bounding box regression field formats.

a) 2D Bounding Box from YOLO to KITTI format: To convert YOLO formatted 2D bounding boxes (x_c, y_c, w, h) to KITTI formatted 2D bounding boxes $(x_{\min}, y_{\min}, x_{\max}, y_{\max})$ for the corresponding image size (W, H) in pixels, apply:

$$x_{\min} = (x_c - w/2) \times W, \quad y_{\min} = (y_c - h/2) \times H,$$
$$x_{\max} = (x_c + w/2) \times W, \quad y_{\max} = (y_c + h/2) \times H.$$

b) Positional Parameter Extraction for 3D Bounding Box Regression Network: $(X, Y, Z, \alpha, \text{rotationY})$*:* The local position information between the camera and the observed agent facilitates the determination of the distance to the relative center (X, Y, Z), along with the orientation angles *alpha* and rotationY. Refer to Figure 19 and the following list for definitions of these variables:

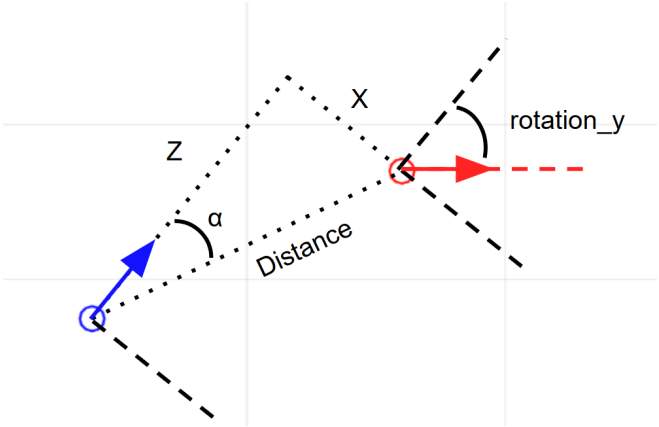


Fig. 19: Geometry for 3D annotation: distance from camera to object center, depth Z, horizontal offset X, the viewing angle α, and the object yaw (rotation_y).

- Z: forward distance along the camera's optical axis.
- X: horizontal offset rightwards in the camera frame.
- Y: vertical offset downwards, kept as constant planar motion result in marginal difference in camera height.
- α: the observation angle, i.e. angle between the camera's forward direction and the object center ray.
- rotationY: the object's yaw in camera coordinates (difference between the camera's heading and the object's heading).

Starting from the synchronized Marvelmind camera pose $(x_{\text{cam}}, y_{\text{cam}}, z_{\text{cam}}, \theta_{\text{cam}})$ and the fixed object's global position $(X_{\text{obj}}, Y_{\text{obj}}, Z_{\text{obj}}, \theta_{\text{obj}})$, the calculation of KITTI-style positional fields proceeds as follows.

First, the global offset vector is calculated,

$$\Delta x = X_{\text{obj}} - x_{\text{cam}},$$
$$\Delta y = Y_{\text{obj}} - y_{\text{cam}},$$
$$\Delta z = Z_{\text{obj}} - z_{\text{cam}}.$$

Then (X, Y, Z) is calculated by rotating the global offset vector $(\Delta x, \Delta y, \Delta z)$ into the camera coordinate plane. Define

$\phi = \frac{\pi}{2} - \theta_{\text{cam}}$ as the rotation from the local heading to the camera's relative frame, and,

$$X = \Delta x \cos\phi \; - \; \Delta y \sin\phi,$$
$$Z = \Delta x \sin\phi \; + \; \Delta y \cos\phi,$$
$$Y = -\Delta z.$$

The object Yaw in the camera-coordinate frame, referencing Figure 19, is

$$\text{rotationY} = \theta_{\text{obj}} - \phi,$$

Then finally, the observation angle α is

$$\alpha = y_{rotation} - arctan2(X, Z).$$

At each instance of data recording, the established mathematical computation is utilized to calculate $(X, Y, Z, \text{rotationY}, \alpha)$ through a Python script accessible via the related project GitHub [33]. The initial script saves the positional data in a `.csv` format, facilitating visualization compared to individual text files. A secondary Python script parses the output `.csv` and the corresponding YOLO formatted data from Roboflow to create KITTI-style annotations in text-file format.

5) Camera Intrinsics Calibration: Although camera intrinsics are not direct labels in KITTI, they are essential for back-projecting 2D image points into metric 3D space. Accurate camera intrinsics are required to ensure the 3D bounding-box regression network produces reliable position and orientation estimates. The camera intrinsic parameters are compactly represented by the matrix

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}, \tag{14}$$

where f_x, f_y represents the focal lengths (in pixels) along the image axes and (c_x, c_y) denotes the principal point.

Calibration employed an 8×6 inner-corner checkerboard, captured at the deployment resolution under varied orientations using the ROS USB-cam node discussed in subsubsection IV-A.2. Checkerboard frames provided the reference grid for camera intrinsic estimation. OpenCV's[6][34] built-in routines processed these images to compute K. OpenCV calibration produced stable estimates of f_x, f_y, and (c_x, c_y) with a mean re-projection error below 0.2px.

The corresponding camera intrinsics for the Logitech C920 are represented in the matrix below:

$$K = \begin{bmatrix} 4.88 \times 10^2 & 0 & 3.31 \times 10^2 \\ 0 & 4.86 \times 10^2 & 1.52 \times 10^2 \\ 0 & 0 & 1 \end{bmatrix} \tag{15}$$

[6]OpenCV is an open-source computer vision library providing real-time image and video processing capabilities through a comprehensive set of optimized algorithms, was used for camera calibration and image preprocessing.

6) End-to-End Data-Set Creation Pipeline: Figure 20 illustrates the parallel annotation pipelines that transform raw sensor data into final annotations formatted for KITTI and YOLO, necessary for model training.

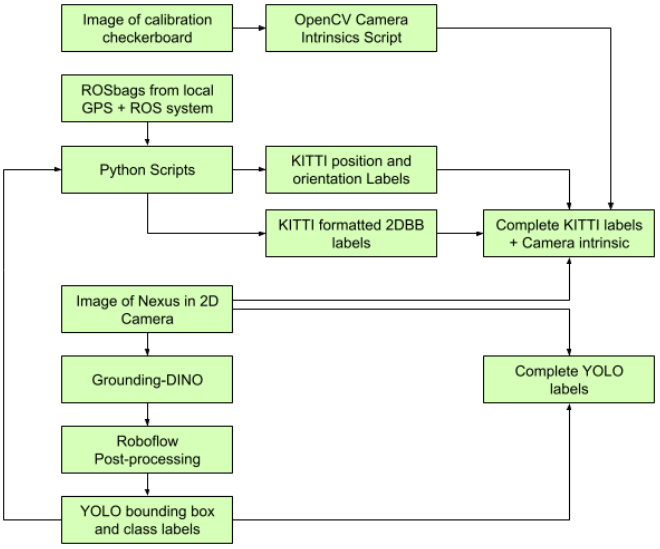


Fig. 20: End-to-end data-flow pipeline: from checkerboard images and ROSbags, through 2D/3D label generation, to complete datasets for YOLO and 3D-BB regression models.

B. Model Training

With all labels and images collected, model training can now be undertaken. As discussed in section III model training leverages RUG's High-Performance Computing Cluster, the Hábrók. This section explores the training process for the model, including final data preparation and formatting, training scripts, and initial training output results.

1) Data Preparation & Formatting:

a) Dataset Split: The final step in dataset preparation involves the selected dataset training split, which determines the allocation of data for model training, model validation, and model testing. In a highly controlled deployment context, where lighting, background, and object appearance exhibit minimal variation, a large validation set yields diminishing returns. Generally, allocating more frames to the training set enhances model performance, while ensuring a sufficient number of frames for unbiased validation is essential [16].

Only Ultralytics YOLO utilizes an explicit train/validation split for training, while the 3D Bounding Box Regression Network employs the entirety of the data directly for training. Instead, the 3D Bounding Box Regression Network reserves a test set for manual evaluation of model performance, rather than relying on an intermediate validation score obtained during training [18] [21].

Roboflow's online application includes built-in tools for customizing the data split. Multiple versions of the dataset can be generated rapidly with various training, validation, and test configurations [32].

b) Directory Formatting: To ensure compatibility with the specific training process, adherence to strict directory formatting is essential.

For Ultralytics YOLO (both v8 and v11), the specific directory format is as follows:

```
YOLOdataset
├── data.yaml
├── images
│   ├── train
│   └── val
└── labels
    ├── train
    └── val
```

Fig. 21: Ultralytics YOLOv8 and YOLOv11 dataset directory structure

Referencing Figure 21, images and annotations are organized in parent directories, with the training and validation split arranged in sub-directories. For successful Ultralytics model training, the inclusion of `data.yaml` is essential; this file serves as a reference for the training process and contains file paths pointing to the relevant data [25].

For the 3D Bounding Box Regression Network, the specific directory format is as follows:

```
Kitti/
└── training
    ├── image_2
    └── label_2
```

Fig. 22: 3D Bounding Box Regression training directory structure

It is important to note that, for compatibility with the training script, the parent directory must be called `Kitti` [21].

2) Training Scripts: Multiple training scripts were created and deployed on the Harbrok. These scripts are accessible through the related project GitHub **gitME**.

Four separate scripts were created for training Ultralytics YOLO models, with each script corresponding to a specific model: YOLOv8n, YOLOv8s, YOLOv11n, and YOLOv11s.

Two separate scripts were created for training the 3D Bounding Box Regression Network: one script utilized the heavier VGG19 backbone, while the other script employed the lighter ResNet-18 backbone.

C. Embedded IBVR System Implementation

With the Ultralytics YOLO and the 3DBB Regression Net trained, the development of the complete IBVR formation control system can commence. The following section discusses the perception pipeline built on the Jetson Orin Nano, and the IBVR-based control pipeline implemented onboard the Nexus Mobile Robot.

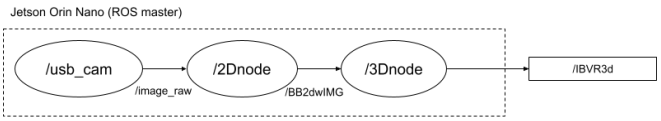


Fig. 23: RQT graph of implemented perception pipeline on Jetson Orin Nano

1) Perception Pipeline: Extracting IBVR Information: Below, the RQT graph view of the perception pipeline is presented, isolated from the rest of the deployed system:
Three separate nodes are utilized onboard the Jetson Orin to achieve the four necessary sensor outputs discussed in subsubsection II-C.4: Localization, Classification, Identification, and Size Estimation. Each node in the perception pipeline functions as follows:

1) `/usb_cam`. Accesses Logitech C920 via ROS's USB-cam driver (also deployed in **??** for the ROS data collection system) and publishes raw frames on `/image_raw`.
2) `/2Dnode`. Subscribes to `/image_raw` and executes selected the Ultralytics YOLO model for object detection. Each frame yields one or more bounding boxes plus class and confidence scores. To maintain persistent IDs for Nexus agents across frame the `/2Dnode` applies a simple IOU(intersection-over-union)-based tracking algorithm. Detections in the current and previous frames are linked through an ID if there exists sufficient overlap between the bounding boxes in the image-coordinate plane. Unmatched detection spawn new tracking IDs, and a tracking ID without matches for three consecutive frames are deleted. The node publishes the `/BB2dwIMG` topic, containing the full image detection was run on, the detected bounding boxes (x_c, y_c, w, h), with the associated classes and IDs.
3) `/3Dnode`. Subscribes to `/BB2dwIMG` and performs 3D Bounding Box inference using the 3D bounding box regression net model. As a proxy for the perceived spherical-cap area s_k, the longest diagonal of the 3DBB is calculated and use as the source of size estimation in the perception pipeline. The node publishes the `/IBVR3D` topic, extending from the `/2Dnode` the class name, ID, 2D bounding box x-center and y center and then the diagonal length.

2) IBVR Controller: Actuating Formation Control: Below, the RQT graph view of the control pipeline is presented, isolated from the rest of the deployed system:

Referencing Figure 24, similarity exists with the data collection system observed in Figure 17. Identical nodes such as `nexus_base_controller`, `/nexus_base` and `/hedge_rcv` function as described in subsubsection IV-A.2. To distinguish the function of `/hedge_rcv` from the data collection system, the marvelmind node is utilized to establish a ground truth for the comparison of perception outputs. This approach allows for proper evaluation of system performance. The `IBVR_controller` has four specific

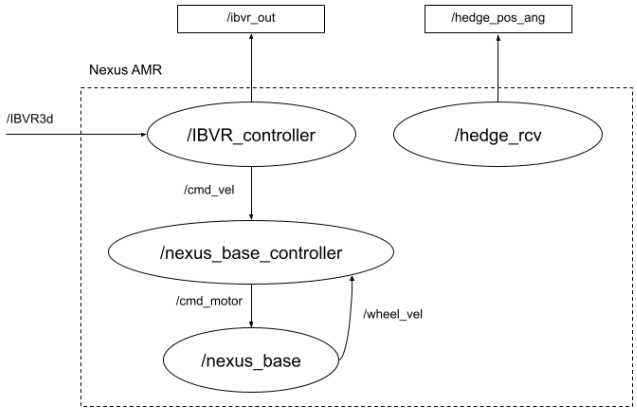


Fig. 24: RQT graph of implemented control pipeline on Jetson Orin Nano

functions explored in the following:

1) Kalman Filter. A Kalman filter is introduced at the head of the `IBVR_controller` node to smooth the diagonal-length measurements arriving on `/IBVR3D`. Real-time 3DBB outputs can jitter or jump due to sensor noise or inference errors. The Kalman filter mitigates this by recursively blending each new measurement with its own prior estimate, yielding a steadier size signal for the controller. Two tunable parameters govern the Kalman behavior: the process noise covariance Q and the measurement noise covariance R. By adjusting these values, the IBVR controller can balance responsiveness against stability, ensuring accurate and smooth size-estimates are used to actuate IBVR control [35].
2) IBVR Controller. Recieves the filtered diagonal length from the Kalman filter, then converts it to spherical-cap s_k for feeding into IBVR's error function. As described in subsubsection II-C.3, the gradient-descent based control law then produces a control output $\mathbf{u}$ that is processed and sent as a `/cmd_vel` to the `nexus_base_controller` node.
3) Center Steering Proportional Controller. Utilizing the center-x values for each detected agent, a simple proportional controller is introduced that rotates the NexusAMR towards the center of the two agents. As established in the perception pipeline consistent tracking IDs through IOU requires that bounding boxes are continuously in view, this ensures that the camera is orientated at equal viewing angles from each agent. Maximizing the likely hood that IDs remain consistent, enabling accurate tracking of individual agents.
4) ID Loss Fail Sage. In the event that localization of one of the agents is loss resulting in a change of or a loss of ID tracking number, the system cuts the publishing of `/cmd_vel` messages stopping any further movement.

3) Complete ML-CV Embedded IBVR Formation System: To unify the previously discussed perception pipeline and IBVR controller into a complete system, a wireless ROS network connects the Jetson Orin Nano to the Nexus AMR. The completed system can be visualized using the RQT graph below:

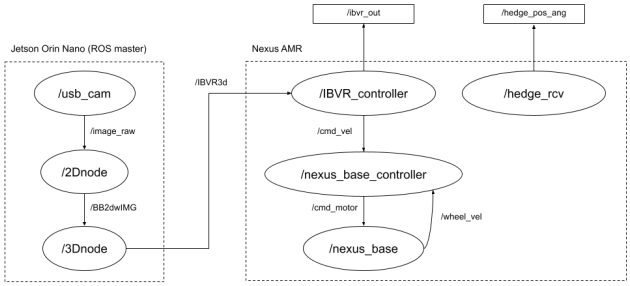


Fig. 25: RQT graph of implemented completed embedded system for IBVR formation control

V. SYSTEM EXPERIMENTATION & RESULTS

A. Experiment Description

Due to project time constraints, a limited set of feasibility experiments were performed on the embedded ML-CV IBVR system established in subsection IV-C. The three test scenarios included:

1) Two-Agent (Triangular) Formation Control. The IBVR controller onboard the Nexus is instructed to keep equal distance from to two stationary agents. The test is initiated from a random initial positions, and is stopped by manual intervention in the case of a potential collision or if the behavior of the Nexus stabilizes. Stabile in this context implies, no movement or repetitive oscillatory movement.
2) Single-Agent Distance Regulation. The IBVR controller onboard the Nexus is instructed to maintain a set distance from a single stationary agent. The test is initiated from random initial positions, with the same stopping condition as the previous test.
3) Circular Teleoperation Test. The Nexus is teleoperated and driven in a complete circle around the static agent, the perception output is recorded and compared to the ground-truth established by the local GPS. The test is complete after one-complete circle navigation at low-speed.

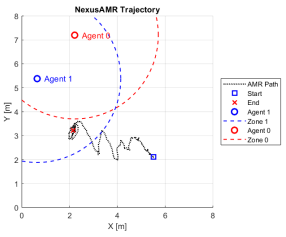
B. Evaluation Metrics

For the autonomous IBVR deployment tests, identical metrics were recorded to measure performance. First, the trajectory path was analyzed to determine the start and end points of navigation to a stable position. The controller-based error e_k for each agent was plotted over time to assess whether the controller drives the system to a zero error state and to visualize the general stability of the perception error value. Finally, two plots are presented related to the true perception error; using (4) to convert the perceived area s_k to the perceived distance d_k, a plot is presented showing both d_k and the global offset distance Δd. An alternative plot is presented displaying the discrepancy between the two values.

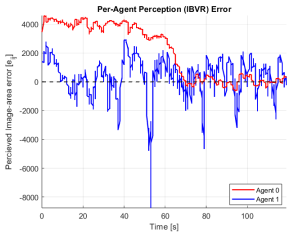
For the teleoperated circle test, the trajectory path is presented alongside a scatterplot representation of the distance discrepancy at each recorded yaw angle of the perceived object.

C. Results

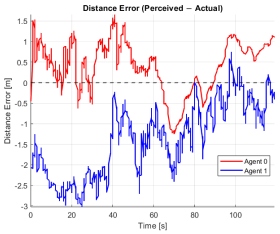
1) Two-Agent (Triangular) Formation Control: Four plots are presented for the two-agent formation control: one for the active agents' trajectory Figure 26a, one for the controller error Figure 26b, one for the perception error Figure 26c, and one for the perceived distance versus measured distance Figure 26d.



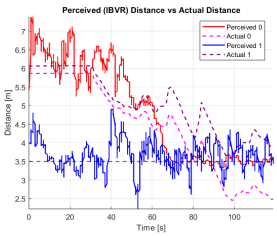
(a) Nexus trajectory with desired distances.



(b) Measured controller-based error e_k over time



(c) Distance error between perceived distance (d_k) and measured distance (Δd) over time.



(d) Perceived (d_k) distance vs. GPS (Δd) distance

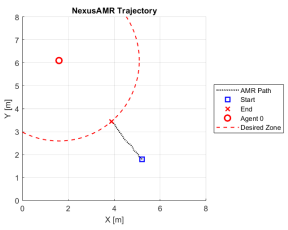
Fig. 26: Two-agent formation control results (1).

2) Single-Agent Distance Regulation Control: Four plots are presented for the single-agent distance regulation control: one for the active agents' trajectory Figure 27a, one for the controller error Figure 27b, one for the perception error Figure 27c, and one for the perceived distance versus measured distance Figure 27d.

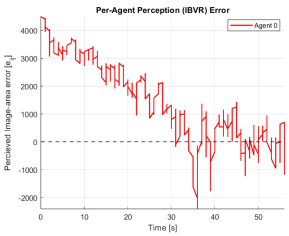
3) Circular Teleoperation Test: Two plots are presented for the circle teleoperation test: the first displays the trajectory of the recorded path Figure 28a, and the second illustrates the perceived distance error with respect to the observation angle Figure 28b.

D. Discussion

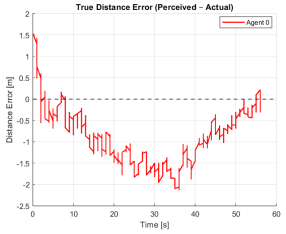
In the two-agent triangular formation control test, referencing Figure 26a, the IBVR-controlled Nexus did not achieve the desired position. In other conducted two-agent tests, the Nexus consistently returned to the same position, as visualized Figure 26a. This repeat behavior implies effective actuation by the controller, driving the agent to the same position repeatedly. Examining Figure 26b, which displays the controller error e_k, provides insight into this behavior.



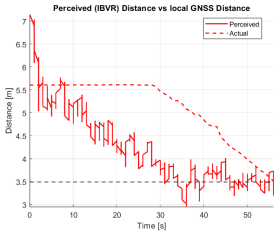
(a) Nexus trajectory with desired distances.



(b) Measured controller-based error e_k over time.

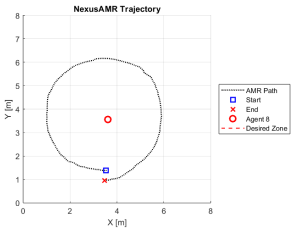


(c) Distance error between perceived distance (d_k) and measured distance (Δd) over time.

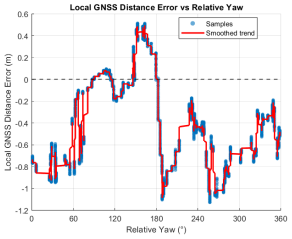


(d) Perceived (d_k) distance vs. GPS (Δd) distance

Fig. 27: Single-agent distance regulations results.



(a) Teleoperated circular trajectory.



(b) Yaw vs. distance error (with fit).

Fig. 28: Sensitivity of perceived distance to relative yaw angle.

While the error for Agent 1 converges to 0 over time, the embedded IBVR system indicates effective actuation for this agent. The error for Agent 0 fluctuates wildly, indicating an unstable perception for that agent. The training method utilized involved training both the YOLO model and 3DBB Regression Net solely on images of Agent 0 (Nexus1). It is likely that both models failed to generalize beyond Agent 0. The models did not adapt to minor visual variations; consequently, the IBVR control could not converge the Nexus agent to the desired formation position. This is further evidenced by referencing appendix A, where no correlation is observed between the measured GPS distance of the agent and the perceived distance of the agent, interpreted from the perceived surface area s_k. Referencing appendix A, the same behavior can be observed in other tests conducted.

In single-agent distance regulation, introduced as a control to test IBVR performance with the dataset subject Nexus1, it was found that Figure 27a the IBVR controller correctly drives the Nexus to the desired distance. Furthermore, Figure 27b the error converges towards zero, although increased

instability is observed compared to previous measurements for two-agent formation control. This instability is attributed to the shortened time-frame of the test, which amplifies the appearance of oscillation, as well as issues with the post-processing of data. As `/hedge_rcv` publishes messages in its own time frame, not one relative to ROS's standards, data points were post-processed through interpolation to synchronize local GPS data with perception data, introducing the displayed choppiness. The issues caused by the attempted synchronization are highlighted referencing Figure 27d, where the perception distance converges to zero, nearly 30 seconds before the local GPS responds. This is caused by two main factors: the natural GPS latency and the method of data time synchronization. Data post-processing steps assumed that a shared time reference frame could be established by setting the initial publishing time for both types of messages as a shared zero time. However, since `/hedge_rcv` and `IBVR_controller` begin publishing at different instances, this reference zero is not shared between them. The true difference between the two reference points is the sum of the latency and the discrepancy in publishing time, with the latter accounting for the significant majority.

In the Teleoperation circle test, substantial fluctuations were observed due to variations in the direct viewing angle of the neighboring agent, as referenced in Figure 28b. The diagnosis for this issue relates to the data-collection method, specifically the ROS system established in subsubsection IV-A.2. During the dataset recording process, the speed of the Nexus capturing images while being teleoperated around the target agent inadvertently introduced noise into the dataset. Although local GPS systems exhibit high accuracy in localizing static objects, inefficiency arises in real-time tracking of objects in the testing environment due to latency. It was discovered that for multiple image frames, the same global position was recorded in the dataset. In the context of 3DBB Regression Net, this indicates that during training, multiple different reference images for the same position and same viewing angle were provided. It is important to note that the test was conducted on a Nexus not included in the model; therefore, fluctuations may be inflated due to previously discussed poor model generalization. Due to data corruption, data for the teleoperated circle test of the Nexus1 is unrecoverable and due to time constraints, a new test could not be undertaken.

E. Limitations

The main limitations in the deployment of the real-world ML-CV enabled IBVR system are as follows:

- Local GPS latency. Despite centimeter accuracy when statically position, dynamic motion induced non-trivial latency and accuracy issues. This injected noisy ground-truth annotations into the 3DBB Regression Network crippling model training, and subsequently model performance effecting the deployed ML-CV based IBVR formation control system. Moreover, lack of a consistent ground-truth from the local GPS limited the ability to properly evaluate model performance in some cases.
- Limited Training Set. Models were exclusively trained on Nexus 1, leading to poor generalization to all other Nexus platforms as seen in the two-agent formation control tests.
- Temporal Constraints. The main limitation of this project was the amount of time allocated versus the ambitions. The majority of the project time was spent building the end-to-end system, leaving insufficient time for testing and optimizing the deployed systems its fullest extent.

VI. CONCLUSIONS & FUTURE WORK

A. Conclusion

This work commenced with an exploration of Image-Based Visual Relative (IBVR) formation control, revealing the necessity for an onboard system to extract localization, classification, identification, and size-estimation information from the selected sensor, a 2D camera. After surveying several approaches, a perception pipeline combining YOLO for 2D object detection with a single-image 3D bounding box regression network inspired [21] was selected to provide all required outputs. A comprehensive end-to-end framework was subsequently implemented, encompassing dataset collection, model training, and integration into an embedded real-world IBVR system.

Closed-loop experiments on Nexus ground robots demonstrated that the vision-only formation control pipeline can reliably drive a multi-agent team to converge into the desired configuration. Despite these promising results, performance was limited by the 3D-box regression model; noisy training labels arising from GPS latency and a narrowly constrained dataset hindered full generalization.

B. Future Work

Future work and extensions of this project include:

- Robust Ground Truth. Replace or augment current local GPS system with low-latency sensor alternative that enables for accurate dynamic measurements for dataset creation and system performance evaluation.
- Model Generalization. Expand training set across the multiple Nexus platforms available in the DTPA laboratory, introduce data augmentation for a more robust deployment system.
- Robust 2D deployment: Extend current approach with Nexus or other planar robots to real-world deployment context and measure system effectiveness.
- 3D Extension: Adapt current framework with improvements to UAV swarms, exploring true 6-DoF coordination in 3-dimensional space.

REFERENCES

[1] J. Ota, "Multi-agent robot systems as distributed autonomous systems," *Advanced Engineering Informatics*, vol. 20, no. 1, pp. 59–70, 2006.

[2] A. L. De Sousa, A. S. De Oliveira, and M. A. S. Teixeira, "Multi-agent robot systems: Analysis, classification, applications, challenges and directions," in *2024 IEEE International Conference on Industrial Technology (ICIT)*, 2024, pp. 1–8.

[3] Y. Liu, J. Liu, Z. He, Z. Li, Q. Zhang, and Z. Ding, "A survey of multi-agent systems on distributed formation control," *Unmanned Systems*, vol. 12, no. 05, pp. 913–926, 2024. eprint: https://doi.org/10.1142/S2301385024500274.

[4] L. Ma, D. Meng, X. Huang, and S. Zhao, "Vision-based formation control for an outdoor uav swarm with hierarchical architecture," *IEEE Access*, vol. 11, pp. 75 134–75 151, 2023.

[5] M. Y. Arafat, M. M. Alam, and S. Moh, "Vision-based navigation techniques for unmanned aerial vehicles: Review and challenges," *Drones*, vol. 7, no. 2, 2023.

[6] Y. Chang, Y. Cheng, U. Manzoor, and J. Murray, "A review of uav autonomous navigation in gps-denied environments," *Robotics and Autonomous Systems*, vol. 170, p. 104 533, 2023.

[7] M. R. Rosa, A. Berkel, and B. Jayawardhana, "Image-based visual relative information for distributed rigid formation control in 3d space," *IEEE Control Systems Letters*, vol. 8, pp. 658–663, 2024.

[8] J. Kaur and W. Singh, "A systematic review of object detection from images using deep learning," *Multimedia Tools and Applications*, vol. 83, pp. 1–86, Jun. 2023.

[9] M.-C. Park, H.-K. Kim, and H.-S. Ahn, "Rigidity of distance-based formations with additional subtended-angle constraints," in *Proceedings of the 17th International Conference on Control, Automation and Systems (ICCAS)*, IEEE, Jeju, Korea, Oct. 2017.

[10] D. Zelazo and S. Zhao, *Formation control and rigidity theory*, Snapshots of Modern Mathematics from Oberwolfach, No. 17/2019, Creative Commons BY-SA 4.0, 2019.

[11] J. Gao, *Rigidity theory*, Lecture 3 of CSE590: Information Processing in Sensor Networks, 2023.

[12] H. J. Garcia de Marina Peinado, "Distributed formation control for autonomous robots," Supervisors: Prof. M. Cao, Prof. B. Jayawardhana, Prof. J.M.A. Scherpen, PhD thesis, University of Groningen, Groningen, The Netherlands, 2016.

[13] M. Bualat, T. Smith, E. Smith, T. Fong, and D. Wheeler, "Astrobee: A new tool for iss operations," May 2018.

[14] K. Watanabe, "Magnetic docking mechanism for free-flying space robots with spherical surfaces," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 5994–5999.

[15] A. Diouf, B. Belzile, M. Saad, and D. St-Onge, "Spherical rolling robots—design, modeling, and control: A systematic literature review," *Robotics and Autonomous Systems*, vol. 175, p. 104 657, 2024.

[16] M. Valdenegro, "Introduction to machine learning (for ai): Introduction to machine learning," University Lecture, 2025.

[17] M. Valdenegro, "Introduction to machine learning (for ai): Neural networks," University Lecture, 2025.

[18] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *CoRR*, vol. abs/1506.02640, 2015. arXiv: 1506.02640.

[19] S. Liu, Z. Zeng, T. Ren, *et al.*, *Grounding dino: Marrying dino with grounded pre-training for open-set object detection*, 2024. arXiv: 2303.05499 [cs.CV].

[20] Y. Li, Y. Wang, W. Wang, D. Lin, B. Li, and K.-H. Yap, "Open world object detection: A survey," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 35, no. 2, pp. 988–1008, 2025.

[21] A. Mousavian, D. Anguelov, J. Flynn, and J. Kosecka, *3d bounding box estimation using deep learning and geometry*, 2017. arXiv: 1612.00496 [cs.CV].

[22] J. Zhang, *Survey on monocular metric depth estimation*, 2025. arXiv: 2501.11841 [cs.CV].

[23] K. Lasinger, R. Ranftl, K. Schindler, and V. Koltun, "Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer," *CoRR*, vol. abs/1907.01341, 2019. arXiv: 1907.01341.

[24] E. Arani, S. Gowda, R. Mukherjee, O. Magdy, S. Kathiresan, and B. Zonooz, *A comprehensive study of real-time object detection networks across multiple domains: A survey*, 2023. arXiv: 2208.10895 [cs.CV].

[25] Ultralytics LLC, *Ultralytics YOLO: Real-Time Object Detection for Commercial and Research Applications*, https://github.com/ultralytics/ultralytics, [Online; accessed 2025-07-04], 2024.

[26] Z. Soleimanitaleb and M. A. Keyvanrad, *Single object tracking: A survey of methods, datasets, and evaluation metrics*, 2022. arXiv: 2201.13066 [cs.CV].

[27] M. Goldblum, H. Souri, R. Ni, *et al.*, "Battle of the backbones: A large-scale comparison of pretrained models across computer vision tasks," in *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., vol. 36, Curran Associates, Inc., 2023, pp. 29 343–29 371.

[28] M. Stokroos, *Nexus_base_ros: Ros wheelbase controller for the nexus omni 4-wheeled mecanum robot*, GitHub repository, [Online; accessed 2025-07-04], 2024.

[29] A. B. Yoo, M. A. Jette, and M. Grondona, "Slurm: Simple linux utility for resource management," in *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*, Springer, 2003, pp. 44–60.

[30] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision bench-

mark suite," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, [Online; accessed 2025-07-04], 2012, pp. 3354–3361.

[31] Autodistill Team, *Autodistill api: Grounding-dino region proposal*, `https://github.com/autodistills/autodistill-grounding-dino`, [Online; accessed 2025-07-04], 2024.

[32] Roboflow, Inc., *Roboflow: Computer vision dataset management and annotation platform*, Online; accessed 2025-07-04, `https://roboflow.com`, 2025.

[33] Ian Michel, *IM-BaIP-IBVR: Embedded Computer Vision System for Practical Image-Based Visual Relative Formation Control*, GitHub repository, [Online; accessed 2025-07-04], 2025.

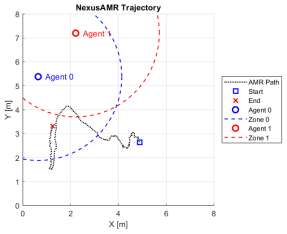
[34] G. Bradski, "The opencv library," *Dr. Dobb's Journal of Software Tools*, 2000, [Online; accessed 2025-07-04].

[35] Y. Pei, S. Biswas, D. S. Fussell, and K. Pingali, *An elementary introduction to kalman filtering*, 2019. arXiv: `1710.04055 [eess.SY]`.

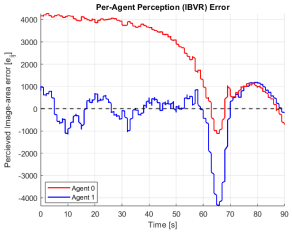
APPENDIX

A: TWO-AGENT (TRIANGULAR) FORMATION CONTROL EXPERIMENT RESULTS

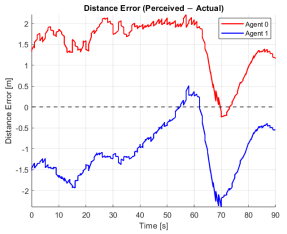
The following figures depict the behavior of the IBVR-enabled nexus AMR from a frontal head-on starting position.



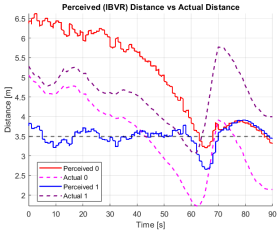
(a) Nexus trajectory with desired distances.



(b) Measured controller-based error e_k over time



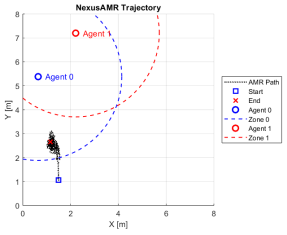
(c) Distance error between perceived distance (d_k) and measured distance (Δd) over time.



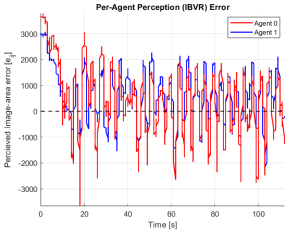
(d) Perceived (d_k) distance vs. GPS (Δd) distance

Fig. 29: Two-agent formation control results (2)

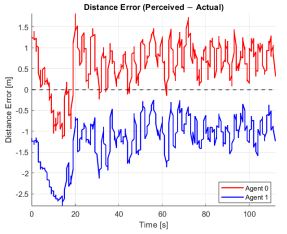
The following figures depict the behavior of the IBVR-enabled nexus AMR from a leftward starting position.



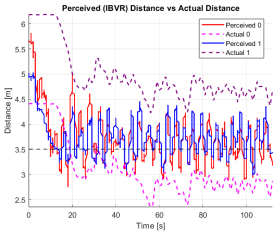
(a) Nexus trajectory with desired distances.



(b) Measured controller-based error e_k over time



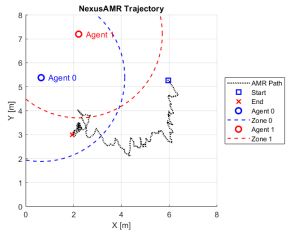
(c) Distance error between perceived distance (d_k) and measured distance (Δd) over time.



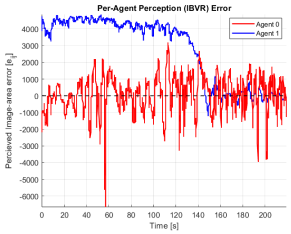
(d) Perceived (d_k) distance vs. GPS (Δd) distance

Fig. 30: Two-agent formation control results (3)

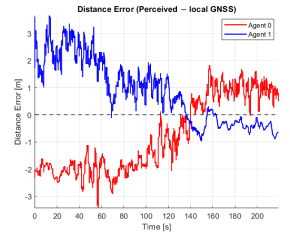
The following figures depict the behavior of the IBVR-enabled nexus AMR from a rightward starting position.



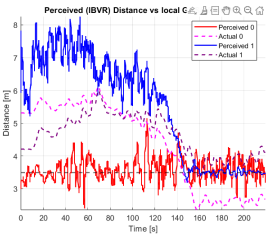
(a) Nexus trajectory with desired distances.



(b) Measured controller-based error e_k over time



(c) Distance error between perceived distance (d_k) and measured distance (Δd) over time.

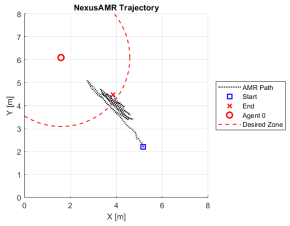


(d) Perceived (d_k) distance vs. GPS (Δd) distance

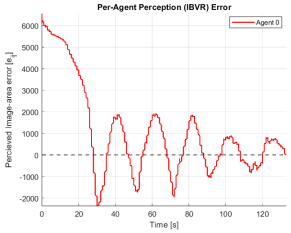
Fig. 31: Two-agent formation control results (4)

B: Single-Agent Distance Regulation Control Experiment Results

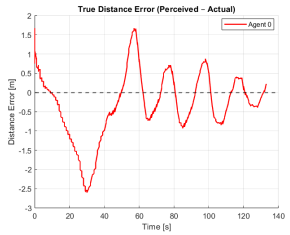
The following figures depict the behavior of the IBVR-enabled nexus AMR for single agent control for a desired distance of $2m$.



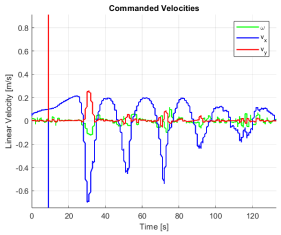
(a) Nexus trajectory with desired distances.



(b) Measured controller-based error e_k over time.



(c) Distance error between perceived distance (d_k) and measured distance (Δd) over time.



(d) Perceived (d_k) distance vs. GPS (Δd) distance

Fig. 32: Single-agent distance regulations results.

C: Model Performance

TABLE IV: Frame rate performance of 3D-bounding-box regressors

Backbone	Output Frame Rate (fps)
VGG-19	≈5
ResNet-18	≈25

TABLE V: Comparison of YOLO model sizes, mAP@0.5, recall, latency and training status

Model	Parameters	$mAP_{0.5}$ (%)	Recall (%)	Latency (ms)	Trained?
YOLOv8n	3,005,843	98.90	96.80	18.2	Yes
YOLOv8s	11,125,971	98.83	96.45	26.0	Yes
YOLOv11n	2,582,346	0.00	0.00	52.7	No
YOLOv11s	9,413,187	86.35	1	112.3	Yes

D: IBVR Perception Visualization

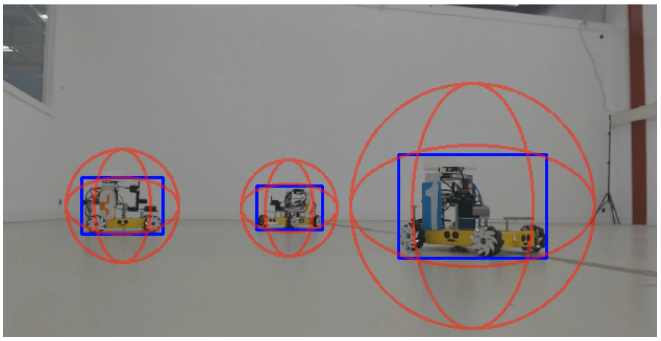


Fig. 33: Visualization of detected 2D-Bounding Box and projected sphere from the perception pipeline.

E: Acknowledgments

AI tools were utilized throughout the project. Large Language Models (LLM) served as essential resources for the practical aspect, orienting research with the necessary tools to create the embedded computer vision system. Furthermore, specific code segments were generated by LLMs to expedite the coding process due to the time constraints of the project.

In writing this report, LLMs were utilized to assist in generating an outline and formulating ideas regarding the structure of the report. The core content and research are authored by the writer of this report, with transitions between sections and footnotes composed with the assistance of AI. The entire report was corrected and spell checked using Writeful, Overleaf's embedded AI writing assistance tool, leveragin their own model and a prompted ChatGPT model. Finally, the Abstract is completely written by AI.