Robotic Football: Vision-Based Control for an Autonomous Humanoid Robot

Bachelor's Project Thesis

Rodrigo Sierra, s5111234, r.sierra@student.rug.nl,
Supervisors: Mauricio Muñoz Arias, PhD & Juan Diego Cárdenas Cartagena

Abstract: This project presents a vision-based system enabling a Bioloid Premium humanoid robot to autonomously detect, approach, and kick a ball in real time. The robot utilizes a Raspberry Pi 5, running a lightweight YOLOv8n model for highly accurate object detection, and applies monocular depth estimation for calculating distances. A rule-based control architecture coordinates the robot's behaviors, ensuring robust and adaptive performance. Experimental results demonstrate reliable, low-latency operation and effective ball interaction, confirming the system's suitability for real-world robotic soccer scenarios. This work advances affordable, modular robotics by integrating state-of-the-art computer vision and flexible communication for autonomous humanoid robots.

1 Introduction

Robotic football, embodied by RoboCup, the international competition dedicated to developing autonomous robotic soccer teams, has been a key catalyst for advances in artificial intelligence, computer vision, and control systems. Visser & Burkhard (2007) details how the competition has promoted breakthroughs in multi-agent collaboration, real-time reasoning, and robust robot design, with impacts extending to fields like disaster response robotics. Similarly, ongoing research in the Standard Platform League (*RoboCup Standard Platform League Publications* (2025)) has produced state-of-the-art methods in vision-based localization, team coordination, and adaptive control, many of which now influence both academic and industrial robotics. However, enabling robots to detect, track, and interact with dynamic objects—such as pursuing and kicking a soccer ball—remains a formidable challenge, particularly under real-time constraints and limited computational resources. These capabilities are not only critical for robotic soccer but also underpin broader applications in service and assistive robotics.

Building upon this foundation of robotic soccer

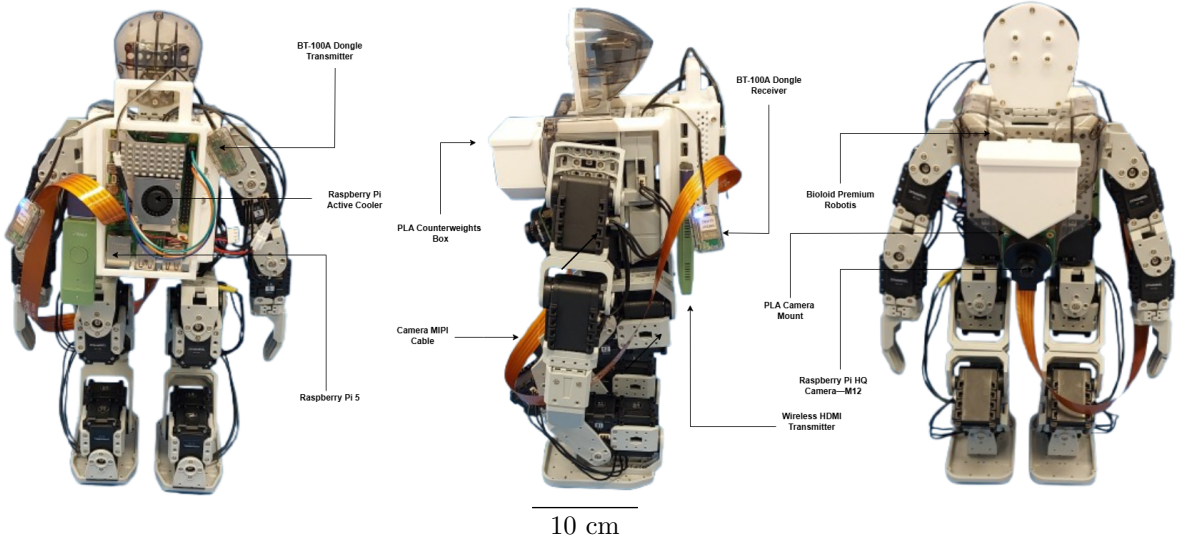


Figure 1.1: The hardware setup of the Bioloid displays the key components: Bioloid Premium Kit (Robotis (2025), Raspberry Pi 5, Raspberry Pi HQ Camera, Camera MIPI Cable, BT-100A Dongles (Receiver and Transmitter), PLA Backpack, Counterweights Box, Camera Mount, Wireless HDMI Transmitter, and Raspberry Pi Active Cooler.

research, humanoid robots have emerged as particularly compelling for tasks requiring social interaction and human-like movement. Research indicates that human-oriented robot appearances enhance social engagement and teamwork, attributes that are especially relevant in collaborative sports scenarios (Kwak (2014),Prescott & Robillard (2021), Wu et al. (2024)). In academic settings, platforms such as the Robotis Bioloid Premium offer a robust, modular, and accessible foundation for prototyping vision-guided control, inverse kinematics, and sensor fusion in interactive tasks (Kalyani (2016), Ali et al. (2020)).

Parallel to advances in robotic platforms, deep learning methods such as YOLO (You Only Look Once) have become the standard for computer vision in robotics, balancing speed and accuracy (Redmon et al. (2016), Bochkovskiy et al. (2020)). Recent studies demonstrate that YOLOv8, in particular, delivers high detection accuracy and efficiency, making it well-suited for real-time object-tracking in robotic soccer, even on platforms with limited computational power Julianda et al. (2024). Furthermore, advances in embedded computing, particularly with devices like the Raspberry Pi 5, have made it feasible to deploy AI algorithms on resource-constrained robotic platforms (Upton & Halfacree (2016)).

This paper presents the design, implementation, and evaluation of a system that enables a humanoid robot, shown in Figure 1.1, to autonomously detect, approach, and kick a ball using real-time vision and control, all powered by an onboard Raspberry Pi 5. To achieve fully autonomous operation, the system integrates (1) a YOLOv8n-based ball detector, (2) a monocular depth estimation framework, and (3) a modular rule-based control architecture.

The remainder of this paper is organized as follows. Section 2 details the methodology, including the hardware setup, YOLO model training, vision system architecture, and control algorithms. Section 3 outlines the evaluation metrics used to assess the performance of the system. Section 4 presents the experimental results, including model performance metrics, behavioral analysis, and trajectory evaluation across multiple trials. Section 5 discusses the implications of the findings, addresses limitations, and outlines directions for future work. Finally, Section 6 concludes with a summary of the contributions and the broader significance of this work for autonomous robotics.

Table 2.1: Shows the hardware components used alongside the Bioloid Kit to provide it with the capabilities of detecting and approaching the ball.

Component	Quantity
Structure	
Bioloid Premium Kit (Robotis)	1
PLA Backpack	1
PLA Camera Mount	1
PLA Counterweights Box	1
Perception/Processing	
Raspberry Pi 5	1
Raspberry Pi HQ Camera - M12	1
M12 Wide Angle Lens	1
500 mm MIPI cable	1
Power	
12V Power Supply	1
Cooling	
Raspberry Pi Active Cooler	1
Connections	
Arduino Cables (Male-To-Female)	4
Visualization (Optional)	
HDMI Wireless Transmitter	1
HDMI Wireless Receiver	1
HDMI to HDMI mini adapter	1

2 Methods

2.1 Materials and Equipment

2.1.1 Robot Platform

All the experiments in this research were conducted using the Bioloid Premium Type-A robot, a commercial humanoid from Robotis shown in Figure 1.1. The robot is powered by 18 Dynamixel AX-12A servo actuators, which provide it with 18 degrees of freedom. The package also comes with a CM-530 controller that manages the motor operations and supports the Dynamixel communication protocol 1, operating at a baud rate of 56700 Mbps via a Transistor-Transistor Logic (TTL) interface. Finally, the kit also comes with an LBS-10 LiPo 11.1V battery, which is used to give power to the CM-530 and all the servos.

2.1.2 Computing Hardware

A Raspberry Pi 5 single-board computer was employed for onboard processing, featuring a quad-core Cortex-A76 processor at 2.4 GHz, 8 GB of RAM, and a Broadcom VideoCore VII GPU. Also, a Raspberry Pi HQ Camera - M12 Mount—alongside an M12 wide-angle lens with 12 megapixels, 2.7 mm of focal length, an aperture size of F2.5, and a field of view of 140 degrees. These components were connected with a 500 mm ribbon cable. In addition to accommodating these components in the robot hardware, a custom 3D-printed PLA backpack mounting solution was designed to

secure the Raspberry Pi to the Bioloid's back, and a camera mount was created to attach the camera to the front hip of the robot. The camera was mounted at a 17-degree angle with respect to the robot's torso.

2.1.3 Other Equipment

Ball: An orange plastic ball with a diameter of 0.135 m was used as the target object for the robot to kick during experiments.

Counterweights: To compensate for the shift in the robot's center of mass caused by the addition of the backpack, a custom 3D-printed PLA counterweight box was securely mounted to the robot's torso. This box was filled with spare tungsten rods to achieve the desired balance.

2.2 Software Development and Implementation

2.2.1 Ball Detection System

The object detection system was built using the YOLOv8n model. YOLO (You Only Look Once) is a real-time object detection algorithm that processes images in a single pass to predict bounding boxes and class probabilities for objects, making it fast and efficient for myriad types of backgrounds such as robotics, healthcare, or agriculture. It uses a convolutional neural network to detect objects, in this case a ball, by dividing the image into a grid and predicting object presence, location, and class within each cell. Since the created model only works with a single class, YOLOv8n was chosen due to its lightweight architecture (3.2 million parameters), optimized for resource-constrained devices like a Raspberry Pi 5, delivering high speed (up to 30 FPS) and sufficient accuracy (mAP@0.5 0.85 for single-class tasks), where mAP (mean Average Precision) is a standard metric that summarizes how well the model detects and localizes objects, indicating reliable performance for the intended application.

The model's creation follows the process described by Figure 2.1. The model was trained on a custom dataset comprising 652 images. The image annotation was done with Roboflow, first with the automatic annotation tool, and then each image was revised one by one manually to ensure all bounding boxes were correct and only the correct class was labeled. After this, the images were auto-oriented and resized to 640x640 pixels and augmented with the parameters that can be seen in table 2.2.

After augmentation, the total number of images amounted to 1433, with an 82%, 9% , 9% split for training, validation, and testing, respectively. Once

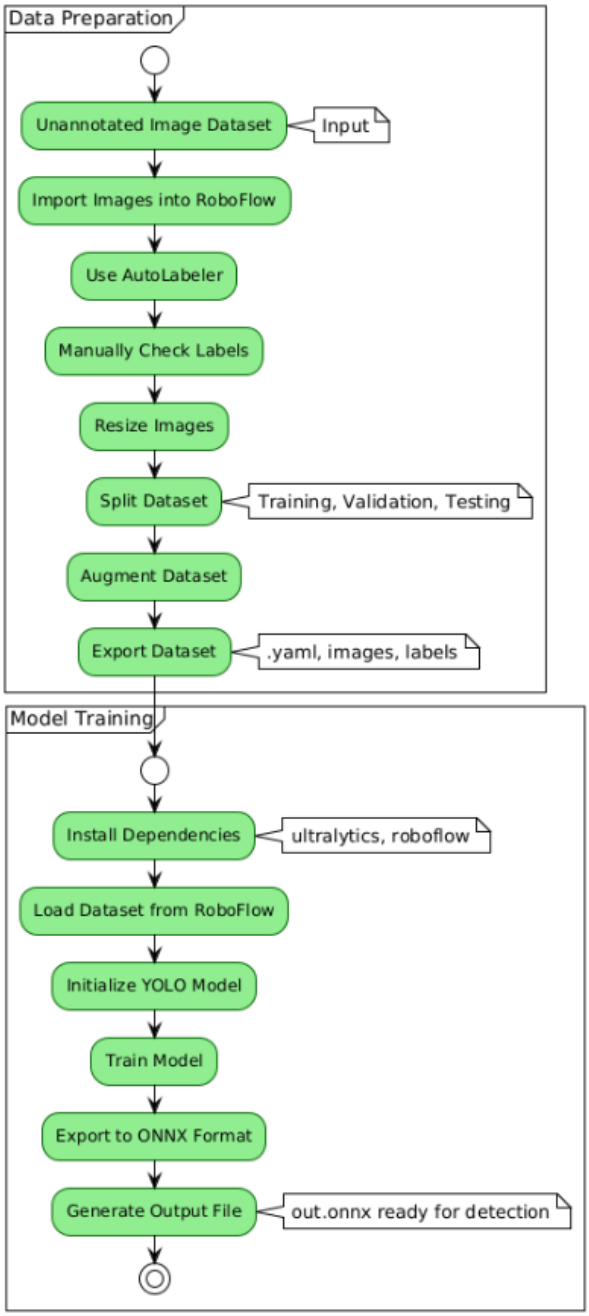


Figure 2.1: Activity Diagram illustrating the Data Preparation and Model Training Workflow.

Table 2.2: Augmentation Parameters

Augmentation	Details
Outputs per training example	3
Flip	Horizontal
90° Rotate	Clockwise, Counter-Clockwise, Upside Down
Shear	±10° Horizontal, ±10° Vertical
Hue	Between −10° and +10°
Brightness	Between −15% and +15%
Blur	Up to 1.1px
Noise	Up to 0.1% of pixels

the dataset was prepared, the model was trained on 70 epochs, a batch size of 16, and only a single class. Finally, once the model was trained, it was converted from PyTorch (.pt) format to Open Neural Network Exchange (.onnx), a format designed to improve the performance of models across different devices (PyTorch Contributors (2024)).

2.2.2 Vision System Architecture

Camera Calibration To accurately map 3D real-world points to 2D image projections and correct for lens distortion in a wide-angle fisheye camera, a calibration process was performed to determine the camera's intrinsic parameters (e.g., focal lengths and principal point), distortion coefficients, and extrinsic parameters (rotation and translation). The fisheye camera calibration method, utilizing a planar checkerboard pattern, was employed due to the wide-angle lens characteristics. This process follows the procedure described in the OpenCV documentation ("OpenCV: Camera Calibration and 3D Reconstruction (Fisheye Model)" (2025)) and detailed in the tutorials by Jiang (Jiang (2019a,b)) and involves the following steps:

Checkerboard Creation A checkerboard pattern consisting of 8x5 squares, with each square measuring 3.5 mm, was printed on an A3 sheet. The pattern was securely affixed to a rigid cardboard backing using adhesive tape to maintain a flat, stable, and consistent surface during calibration. The 8x5 layout was selected to ensure a minimum of 7x4 inner corners, as complete corner detection is essential for accurate calibration.

Image Collection A total of 40 images of the checkerboard were captured at distances ranging from 0.1 m to 0.5 m from the camera. The images were taken to cover a variety of angles and inclinations of the checkerboard relative to the camera, ensuring that all corners of the checkerboard were fully visible in each image.

Image Verification Prior to calibration, all images were inspected to confirm their suitability. This involved verifying that all checkerboard squares were fully visible, no extraneous lines (other than the checkerboard pattern) were detected, and each square had a minimum resolution of 30 pixels to ensure clear corner detection.

Calibration Process The calibration was performed by minimizing the reprojection error between observed 2D image points and projected 3D world points, as described by Equation 2.1. This process estimates the camera's intrinsic matrix K, distortion coefficients D, and extrinsic parameters (rotation $\mathbf{R}_i$ and translation $\mathbf{t}_i$ for each image i). The intrinsic matrix K captures the camera's internal geometry, including focal lengths (f_x, f_y) and principal point coordinates (c_x, c_y). The distortion coefficients D model the fisheye lens distortion, and the extrinsic parameters define the spatial relationship between the camera and the checkerboard in each image. These parameters enable accurate correction of lens distortion and precise mapping of 3D real-world coordinates to 2D image coordinates, which is critical for computer vision applications such as measurement and 3D reconstruction.

The calibration minimizes the following objective function:

$$\underset{K,D}{\operatorname{argmin}} \sum_{i=1}^{N} \sum_{j=1}^{M} \left\| \mathbf{x}_{ij} - \pi(K, D, \mathbf{R}_i, \mathbf{t}_i, \mathbf{X}_j) \right\|^2 \quad (2.1)$$

where the function π projects 3D world points to 2D image points using the estimated camera model.

Inputs:

N Number of calibration images.

M Number of detected checkerboard corners per image.

$\mathbf{x}_{ij}$ 2D image coordinates of corner j in image i.

$\mathbf{X}_j$ 3D real-world coordinates of corner j in the checkerboard reference frame.

Outputs:

K Intrinsic camera matrix:

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

where f_x, f_y are focal lengths (in pixels), and c_x, c_y are principal point coordinates (in pixels).

D Distortion coefficients vector (fisheye model): $D = [k_1, k_2, k_3, k_4]^T$, where:

$\mathbf{R}_i$ Rotation matrix for image i (extrinsic parameter).

$\mathbf{t}_i$ Translation vector for image i (extrinsic parameter).

Depth Perception Given the calibrated camera, the distance (depth) Z from the camera to a detected object whose real-world width is known can be estimated with the pinhole camera model (van den Boomgaard) following Equation 2.2 :

$$Z = \frac{f_x \cdot \phi}{\omega} - \delta \tag{2.2}$$

Where:

Z Estimated depth from the camera to the object (in meters).

f_x Horizontal focal length of the camera (in pixels), obtained from the intrinsic matrix K after calibration.

ϕ Real-world diameter of the object (in meters).

ω Observed width of the object in the image (in pixels).

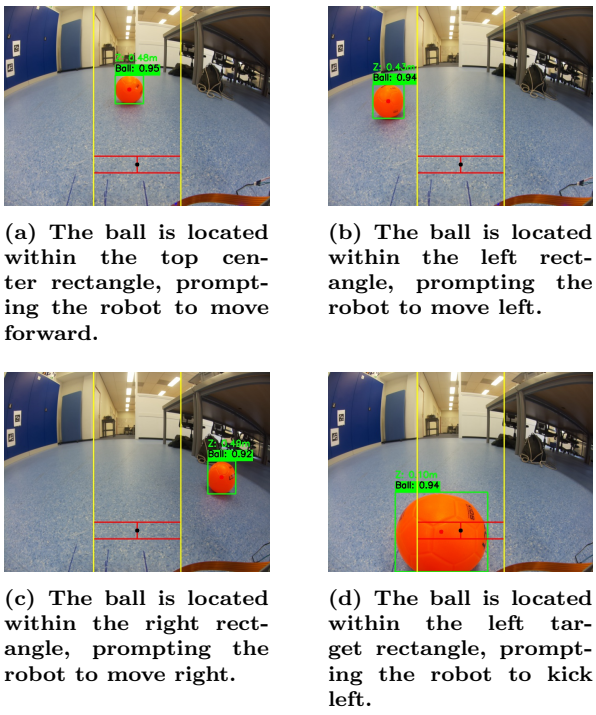
δ Empirical offset (in meters) to account for the distance between the camera and the reference point of the robot.

Note: The offset δ arises because the system measures the ball's maximum width at its midpoint. To calculate the distance to the closest point, half the ball's diameter is subtracted.

This framework enables real-time depth perception using a single camera as long as the full object is in view. This can be crucial for tasks that include object localization, navigation, and mainly any computer vision application.

2.2.3 Controller Architecture

To guide a robot toward a target object, such as a ball in a robotic soccer scenario, a modular, rule-based proportional controller was developed. This system leverages the 2D centroid coordinates (μ, ν) in the pixel space of the detected ball, provided by the YOLO detection model, within the camera's 640×480 image frame. The robot's behavior is now governed by three specialized controllers, each dedicated to a specific task: searching for the ball, approaching the ball, and executing the kick. This separation ensures that action selection is decoupled from the current high-level goal, improving robustness and clarity.



(a) The ball is located within the top center rectangle, prompting the robot to move forward.

(b) The ball is located within the left rectangle, prompting the robot to move left.

(c) The ball is located within the right rectangle, prompting the robot to move right.

(d) The ball is located within the left target rectangle, prompting the robot to kick left.

Figure 2.2: Sample robot camera views showing the ball detected at different positions during approach and alignment for kicking.

Before diving into the architecture, it is important to note that the system architecture was developed around a characteristic observed during experimentation: when the robot is commanded to move forward or backward, its trajectory tends to include a slight turn to the left or right. This directional bias depends on the most recent turning command. If the previous issued action was a left turn or if it is the first forward step after initiation, subsequent forward movements will steer slightly to the left until a right turn is issued, and vice versa. This behavior was consistently encountered during testing and was taken into account throughout the design and evaluation of the experiment.

Search Controller: When the ball is not visible, it rotates the robot in place to scan the full landscape. This controller outputs action "left" until the ball is detected, where control is handed over to the Approach Controller or a full rotation is completed, in which case the program ends.

Algorithm 2.1 Region-Based Controller for Ball Approach and Kick

Input: μ (horizontal centroid coordinate), ν (vertical centroid coordinate)
if ball not detected **then**
    Activate Search Controller (rotate in place)
else if centroid in kicking region **then**
    Activate Kick Controller (kick left or right based on μ)
else
    Activate Approach Controller (move left, right, forward, or backward to center ball)
end if=0

Approach Controller: Once the ball is detected, it uses the centroid coordinates (μ, ν), in the pixel space (640 X 480) to issue a discrete action command—"left," "right," "forward," or "backward"—to align the robot and bring it to the optimal kicking position. Figure 2.2 denotes these boundaries in the frame, where the position of the centroid outside the yellow lines and the red box will indicate which action will be issued. The controller ensures the ball's centroid is steered toward the central region, preparing for a precise kick.

Kick controller: When the ball's centroid enters the central kicking region (horizontally between 215 and 425 pixels, vertically between 360 and 400 pixels), depicted by the red boxes in Figure 2.2 the kick controller determines whether to kick with the left or right leg based on the centroid's horizontal position relative to the central line (μ'). If the ball is not yet within this region, the controller continues to issue navigation commands to fine-tune the robot's alignment.

In overview, the core logic of this module is based on comparing the detected centroid coordinates to predefined region boundaries. Depending on the ball's position, the robot selects from six discrete actions: "kick_left," "kick_right," "left," "right," "forward," or "backward." This process is summarized in Algorithm 2.1,

2.2.4 Serial Communication Protocol

To enable autonomous execution of commands based on visual input, the Raspberry Pi 5 was configured to communicate with the CM-530 controller on the Bioloid robot. This communication was established using the BT-100A wireless dongles included with the robot kit. The receiver dongle was connected to the CM-530, while the master dongle, originally intended for use with the included remote, was connected to the Raspberry Pi's GPIO pins. A UART connection was used, employing four male-to-female Arduino cables to connect both devices as shown in Table 2.3 according to the official Robotis (ROBOTIS (2025)) and Raspberry Pi (Raspberry Pi Foundation (2025)) documentation.

BT100A Pin	Name	Description	GPIO Pin
1	RXD	Receive data	8
2	TXD	Transfer data	10
3	VCC	3.3V DC	1
4	GND	Ground (0V)	6

Table 2.3: BT100A to Raspberry Pi GPIO Pin Mapping

The command packet follows the Remocon protocol and consists of a 6-byte, 8-bit structure, illustrated in Figure 2.3 and described by Equation 2.3. Each packet includes header bytes, data bytes, and their bitwise complements for error checking.

$$\text{Packet} = [\texttt{0xFF}, \texttt{0x55}, \texttt{LB}, \sim \texttt{LB}, \texttt{HB}, \sim \texttt{HB}] \quad (2.3)$$

0xFF	0x55	LB	∼LB	HB	∼HB

Figure 2.3: Remocon packet structure used for UART transmission to the CM-530 (Robotis (n.d.-b)).

Legend:

- `0xFF`, `0x55`: Fixed header bytes.
- `LB`, `HB`: Low and high bytes of the command.
- ∼`LB`, ∼`HB`: Bitwise complements used for error checking.

This structure ensures robust communication between the Raspberry Pi and the CM-530 controller using TTL UART, with basic built-in error detection via complement verification.

The information that is sent within the packet also follows a specific structure. The remote control has a total of ten buttons. As shown in Figure 2.4, each button is represented as a power of 2, starting from 1 (2 to the power of 0) and ending in 512 (2 to the power of 9). For instance, the button U, which executes the walk forward command, has a value of 1. For instructions that require more than two buttons, the value of both buttons is added, and that is the information that is passed. Now the importance of the power of two codes is seen since, in this case it is impossible to create the same sum with different button combinations.

In this way, there are up to 512 executable button combinations Robotis (n.d.-b), although for this research, only six were necessary. The button combinations and the actions they trigger are shown in

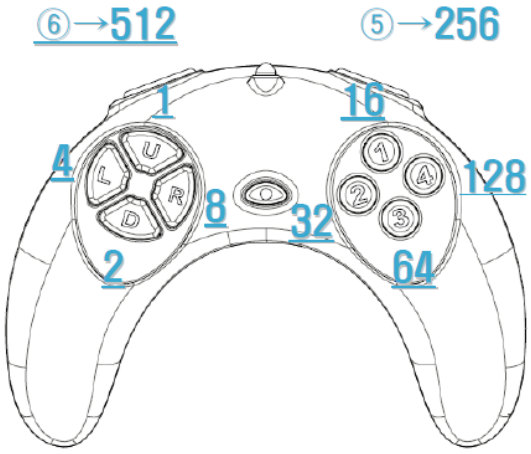


Figure 2.4: Screenshot of the RC-100 communication manual (Robotis (n.d.-b)).

Table 2.4 according to the official Robotis documentation Robotis (n.d.-a)

Table 2.4: Button combinations and their associated codes

Action	Button(s)	Code(s)
Forward	U	1
Backward	D	2
Left	L	4
Right	R	8
Kick Left	L + 3	4, 64
Kick Right	R + 3	8, 64

2.3 Control Loop

The robot navigation system operates within a continuous control loop that is automatically initialized upon program startup. As illustrated in Figure 2.5, the system's physical components interact sequentially to facilitate real-time navigation. The process begins with the camera, which captures raw environmental data and transmits it to the Raspberry Pi via the camera cable. The Raspberry Pi then processes this incoming data to accurately determine the position of the ball and to decide on the most appropriate action for the robot to take. Once a decision is made, the Raspberry Pi communicates the corresponding command to the CM-530 controller using a UART dongle. Upon receiving these instructions, the CM-530 executes the necessary motor commands, causing the Bioloid robot to move accordingly. This movement alters the robot's environment, enabling the camera to capture new data and thereby closing the feedback loop. This cyclical process ensures that the robot continually adapts its actions based on the most recent sensory input, resulting in robust and responsive navigation

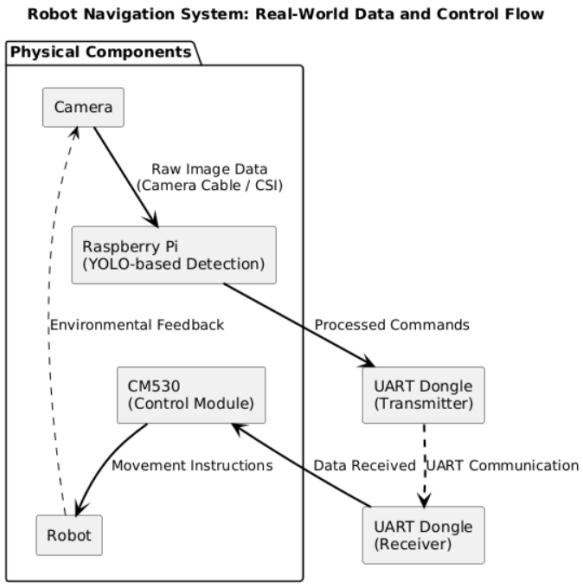


Figure 2.5: Diagram illustrating the data and control flow within the robot navigation system, depicting the interaction between the Camera, Raspberry Pi 5, CM530, and the Bioloid.

performance.

Transitioning to the software architecture, Figure C.1 is a class diagram that provides a detailed view of how the code implements this physical interaction. The process begins with camera initialization, followed by frame resizing from 640×480 to 640×640 to match the YOLO model's input requirements. After inference, the frame is resized back to 640×480 for display. The YOLO model generates bounding boxes with confidence scores, which are refined using Non-Maximum Suppression (NMS) to eliminate overlapping detections, retaining the highest-confidence box per object. This box is then used to compute both the distance (depth) Z to the ball and the centroid (μ, ν). If the ball is not detected, the Search Controller will be activated. On the contrary, if the ball is detected, based on the distance and the centroid position, either the Approach Controller or the Kick Controller will be activated. These controllers will issue one of seven actions: "left," "right," "forward," or "backward" if the ball is not within kicking distance; "kick_left" or "kick_right" if the ball is in the kicking box and finally, "finished" if a full scan is completed without any detection. The selected action is encoded into a packet by a communication class and sent to the CM-530 controller for execution. This iterative process is facilitated by a modular approach, preparing for a more complex decision-making process.

3 Evaluation Metrics

3.1 Object Detection Metrics

3.1.1 Classification and Box Loss

Box Loss: Measures the error in predicting bounding box coordinates for detected objects (e.g., a ball). It combines Intersection over Union (IoU) error, Euclidean distance between box centers, and aspect ratio consistency as shown in Equation 3.1. It averages errors across all bounding boxes, penalizing inaccuracies in position, size, and shape, ensuring a precise localization of the target object.

$$\text{Box Loss} = \frac{1}{N}\sum_{i=1}^{N}\left(1 - \text{IoU}_i + \frac{\rho^2(\hat{b}_i, b_i)}{c_i^2} + \alpha v_i\right) \tag{3.1}$$

where:

- N is the number of bounding boxes.
- $\hat{b}_i$ represents the predicted bounding box coordinates (e.g., (x, y, w, h)).
- b_i represents the ground-truth bounding box coordinates.
- IoU_i is the Intersection over Union for box i.
- $\rho(\hat{b}_i, b_i)$ is the Euclidean distance between the centers of the predicted and ground-truth boxes.
- c_i is the diagonal length of the smallest enclosing box covering both boxes.
- v_i measures the aspect ratio consistency.
- α is a trade-off parameter weighting the aspect ratio term.

Classification Loss: Uses cross-entropy to measure the difference between predicted and true class probabilities across objects, as shown in Equation 3.2. A high classification accuracy indicates that the model identifies the ball correctly, avoiding irrelevant objects.

$$\text{Classification Loss} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} y_{i,c}\log(\hat{y}_{i,c}) \tag{3.2}$$

where:

- N is the number of objects.
- C is the number of classes.
- $y_{i,c}$ is the ground-truth label (1 if class c is correct for object i, 0 otherwise).
- $\hat{y}_{i,c}$ is the predicted probability for class c for object i.

3.1.2 Mean Average Precision (mAP)

Intersection over Union (IoU): Measures overlap between predicted and ground-truth bounding boxes, calculating the ratio of intersection area to union area as shown in Equation 3.3. IoU is the base for evaluating detection accuracy, ensuring the model locates the ball.

$$\text{IoU} = \frac{\text{Area}(\hat{B} \cap B)}{\text{Area}(\hat{B} \cup B)} \tag{3.3}$$

where:

- $\hat{B}$ is the predicted bounding box.
- B is the ground-truth bounding box.
- $\text{Area}(\hat{B} \cap B)$ is the intersection area.
- $\text{Area}(\hat{B} \cup B)$ is the union area.

Average Precision (AP): Summarizes precision at various recall levels for a single class by integrating precision over recall values (0 to 1) using interpolation as shown in Equation 3.4. Provides a single metric for detection performance and assesses the model's ability to detect the ball consistently.

$$\text{AP} = \sum_{r \in \{0, 0.01, \ldots, 1\}} \text{Precision}(r) \cdot \Delta r \tag{3.4}$$

where:

- $\text{Precision}(r)$ is the precision at recall level r.
- Δr is the recall increment (typically 0.01 for 101-point interpolation).

Mean Average Precision (mAP): Averages AP across all classes as shown in Equation 3.5. In this case, since there is only a single class, mAP equals AP.

$$\text{mAP} = \frac{1}{C}\sum_{c=1}^{C}\text{AP}_c \tag{3.5}$$

where:

- C is the number of classes.
- AP_c is the Average Precision for class c.

3.1.3 Precision and Recall

Precision: Measures the proportion of correct detections among all detections. High precision minimizes false detections, ensuring the model is focused on the ball.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{3.6}$$

where:

- TP is the number of true positives (correct detections).
- FP is the number of false positives (incorrect detections).

Recall: Measures the proportion of actual objects detected. A high recall indicates that the model is consistently detecting the ball, avoiding missed opportunities to act.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{3.7}$$

where:

- TP is the number of true positives.
- FN is the number of false negatives (missed detections).

3.2 System Performance Metrics

3.2.1 Iteration Time

Average time to process one frame and act on it. A low interaction time indicates that the system can operate in real-time.

$$\text{Iteration Time} = \frac{T_{\text{total}}}{N_{\text{iterations}}} \tag{3.8}$$

where:

- T_{total} is the total time for all iterations (in seconds).
- $N_{\text{iterations}}$ is the number of iterations (images processed).

3.2.2 Task Success Rate

Proportion of successful task completions (e.g., kicking the ball) compared to the total number of attempts.

$$\text{Task Success Rate} = \frac{N_{\text{success}}}{N_{\text{total}}} \tag{3.9}$$

where:

- N_{success} is the number of successfully completed tasks (e.g., successful ball kicks).
- N_{total} is the total number of tasks attempted.

3.2.3 Trajectory error metrics

Horizontal Error (μ): Mean horizontal distance (in pixels) between the ideal centroid and the ball's centroid. A low horizontal error indicates that the robot is aligned with the ball.

$$\mu = \frac{1}{N} \sum_{i=1}^{N} |\hat{x}_i - x_i| \tag{3.10}$$

where:

- N is the number of trajectory points.
- $\hat{x}_i$ is the horizontal position of the ideal centroid.
- x_i is the actual horizontal position of the ball's centroid.

Vertical Error (ν): Mean vertical distance (in pixels) between the ideal centroid and the ball's centroid. A low vertical error indicates that the robot is close to the ball.

$$\nu = \frac{1}{N} \sum_{i=1}^{N} |\hat{y}_i - y_i| \tag{3.11}$$

where:

- N is the number of trajectory points.
- $\hat{y}_i$ is the vertical position of the ideal centroid.
- y_i is the actual vertical position of the ball's centroid.

3.3 Statistical Analysis

All metrics are reported with mean and standard deviation across multiple trials:

Mean: The mean of the metric across trials is calculated as:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i \tag{3.12}$$

where:

- n is the number of trials.
- x_i is the value of the metric for trial i.

This provides insight into the average system performance.

Standard Deviation: The standard deviation measures the variability across trials:

$$\sigma = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})^2} \tag{3.13}$$

where:

- n is the number of trials.
- x_i is the value of the metric for trial i.
- $\bar{x}$ is the mean value across trials.

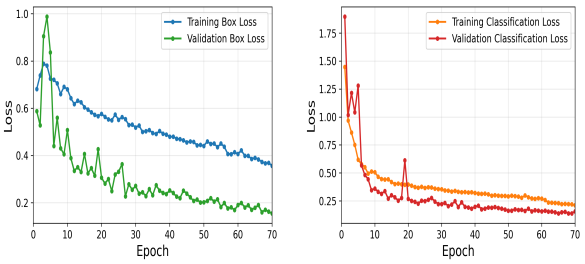
4 Results

The system's performance was evaluated through three complementary approaches: the YOLO detection model's performance, behavioral analysis of the robot's task execution, and quantitative trajectory analysis. All assessments were conducted over five independent trials to ensure statistical reliability, with initial positioning at 40 cm for approach-and-kick tasks and five different out-of-view positions for ball-scanning scenarios.

4.1 YOLO Model Performance

Figure 4.1 presents the evolution of both training and validation losses for bounding box regression and classification over 70 epochs. The training and validation box losses, depicted in Figure 4.1a, start near 1.0 and decrease to approximately 0.2 by the end of training, indicating effective learning and improved localization accuracy without signs of overfitting. The parallel trends in both curves suggest that the model generalizes well to unseen data, as there is no divergence between training and validation losses.

Similarly, the training and validation classification losses shown in Figure 4.1b exhibit a sharp decline within the first 10 epochs, dropping from above 1.2 to below 0.2, and then plateauing. This rapid reduction reflects the model's increasing accuracy in object classification. The absence of significant fluctuations or divergence between training and validation losses further confirms stable learning and minimal overfitting throughout the training process.

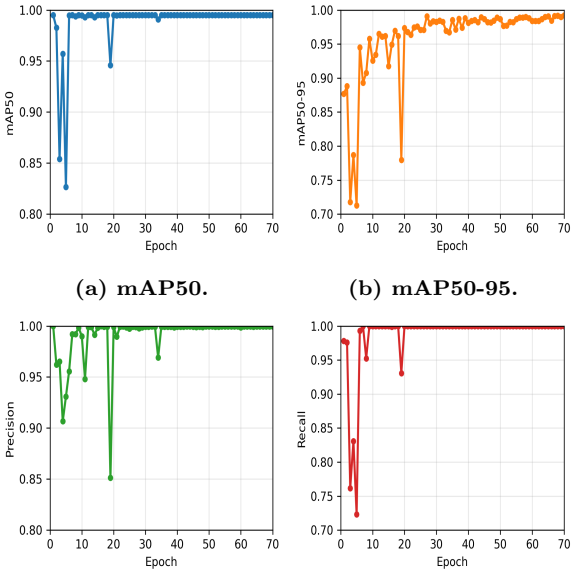


(a) Training and Validation Box Losses. **(b) Training and Validation Classification Losses.**

Figure 4.1: Training and validation box and classification loss curves over 70 epochs.

Continuing with the analysis, Figure 4.2 summarizes the main detection performance metrics: mAP@0.5, mAP@0.5:0.95, precision, and recall.

- mAP@0.5 (Figure 4.2a) measures the mean average precision at an IoU threshold of 0.5. The curve quickly rises and plateaus near 1.0 after about 10 epochs, indicating a solid performance in localization and detection accuracy at this threshold.
- mAP@0.5:0.95 (Figure 4.2b) extends this evaluation to a range of IoU thresholds (from 0.5 to 0.95). This metric is more stringent, yet the model still stabilizes close to 1.0, suggesting robust detection performance across varying levels of localization strictness.
- Precision (Figure 4.2c) and recall (Figure 4.2d) both rapidly approach 1.0 and remain stable, with only minor fluctuations in the early epochs. High precision indicates that most predicted positives are correct, while high recall shows that the model successfully identifies almost all true positives.



(a) mAP50. **(b) mAP50-95.**

(c) Precision. **(d) Recall.**

Figure 4.2: Main performance metrics of the model across all 70 training epochs.

To complete the model evaluation, Figure 4.3 presents a scatter plot illustrating the model's confidence in detecting the ball as a function of depth. In the plot, each point represents a detection done during one of the five runs. The model consistently maintains high confidence levels, typically above 90%, with only a few outliers. This implies that the model can identify the ball consistently throughout the whole distance range.

4.2 Task Performance Evaluation

Figures 4.4 and 4.5 present an example of one of the five-trial datasets, illustrating the robot's performance during approach-and-kick and scanning tasks, respectively. The dual-perspective visualization combines external camera views (left column)

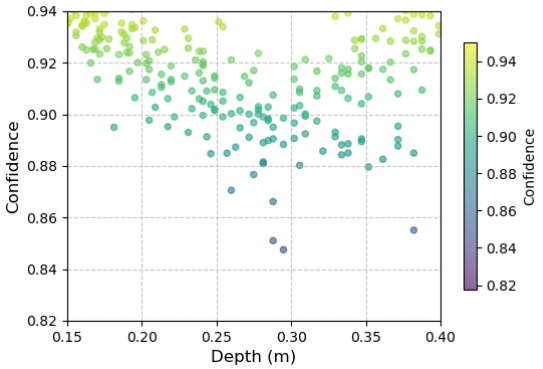


Figure 4.3: Scatterplot of how confident the model is in each detection across all five runs.

with the Bioloid's onboard camera feed (right column), providing insights into both the robot's behaviour and its perception mechanism.

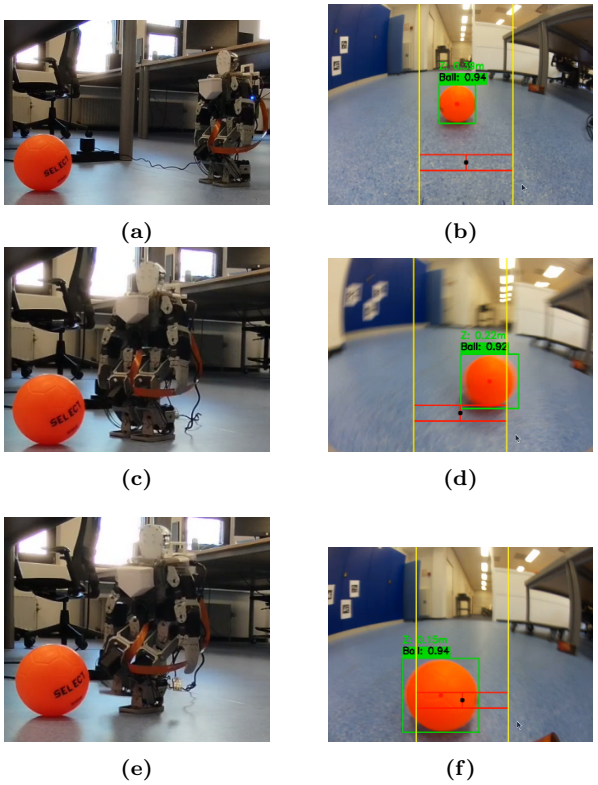
The frames in Figure 4.4 demonstrate the robot's systematic approach behavior, with both sequences of images showing a progressive distance reduction, while onboard views (b, d, f) confirm consistent high-confidence ball detection throughout the trajectory, which correlates with the findings presented in Section 4.1. The consistency across multiple runs confirms the robustness of the integrated perception-control system.

To complement the behavioral analysis, Table 4.1 summarizes the main performance metrics for the approach and kick task across all runs. The results show high consistency, with low standard deviations in both the iteration time (0.500 ± 0.049 s) and the distance traveled (27.14 ± 0.011). The average speed is about 1 cm/s, which is limited by the robot's hardware and the slight deviations in its trajectory, as discussed in Section 2.2.3. Finally, the 100% success rate further demonstrates the robustness of the approach and kick behavior.

The low variability in iteration time and traveled distance across trials highlights the reliability of the control strategy under consistent experimental conditions.

Table 4.1: Behavioral Performance Metrics (Mean $\pm$ SD, $n = 5$ runs, d =0.4 m)

Metric	Approach & Kick
Avg. iteration time (s)	0.500 ± 0.049
Avg. time per run (s)	26.17 ± 1.38
Avg. steps per run	20.2 ± 1.32
Avg. traveled distance (m)	27.14 ± 0.011
Success rate	100% (5/5)



(a) (b) (c) (d) (e) (f)

Figure 4.4: The Bioloid at different steps of the way towards the ball from two different perspectives. Subfigures a, c, and e are from an outside view, and b, d, and f are from the Bioloid's view.

4.3 Trajectory Analysis

Figure 4.6 presents the mean depth in each timestep across all five runs. This figure confirms the findings of Table 4.1, where smooth and consistent convergence toward the kicking region can be appreciated, with notably low variability, especially in the final steps. This indicates that the robot reliably approaches the target area with increasing precision as the task progresses.

Finally, analysis of the error metrics reveals interesting behavioral trends. Figure 4.7a shows the mean horizontal error (μ), which quantifies how far the detected ball position deviates to the left (negative) or right (positive) from the expected position. The plot reveals high variability and a relatively stable mean error throughout the run. By the end of the run, the mean error approaches zero, indicating realignment with the ball.

In contrast, Figure 4.7b depicts the mean vertical error (ν), representing the robot's distance from the ball. A negative decrease represents an approach in the forward direction. This figure shows a smooth and pronounced decrease in error over depth, with a far lower variability compared to the horizontal error.

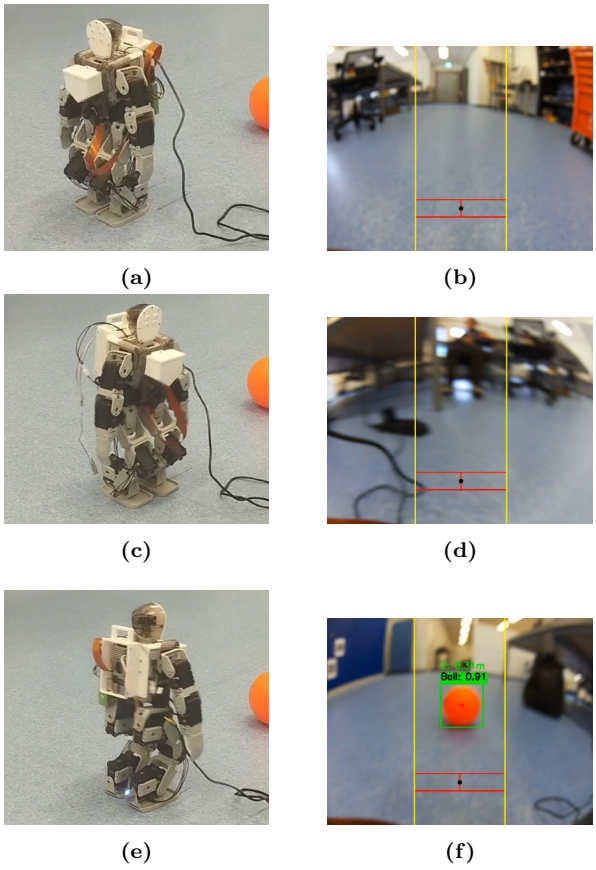


Figure 4.5: The Bioloid at different steps of the process of scanning for the ball from two different perspectives. Subfigures a, c, and e are from an outside view, and b, d, and f are from the Bioloid's view.

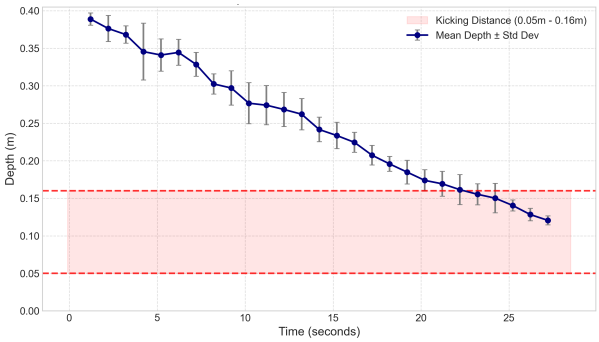
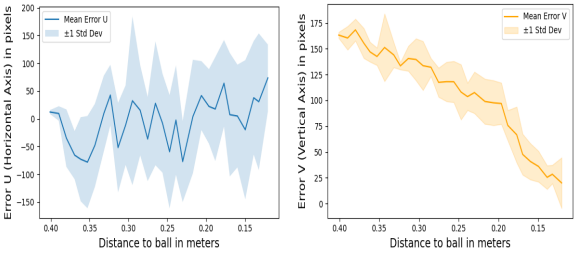


Figure 4.6: Mean depth with standard deviation over time. The shaded region indicates the "kicking distance" range (0.05 m - 0.16 m).



(a) Mean Error μ **(b) Mean Error ν**

Figure 4.7: Comparison between mean error μ and mean error ν across all five runs.

5 Discussion

5.1 Key Insights

YOLO Model Performance: The results for the YOLO model are highly encouraging. As shown in Figure 4.1, the parallel decrease in both training and validation losses, along with their close alignment, indicates that the model is learning effectively without overfitting. This observation is further supported by the consistently high and stable values of all detection metrics (Figure 4.2), including mean Average Precision (mAP), precision, and recall. Together, these results demonstrate that the YOLO model achieves strong generalization and high detection accuracy, making it well-suited for reliable object detection tasks in this context.

Additionally, the slight dip in performance observed in Figure 4.3 can be explained by the blurriness induced by the robot's motion. For instance, both subfigures 4.5d and 4.5d, which correspond to frames captured near the middle of the process, exhibit significantly more blur compared to those at the beginning or end of the sequence. This is because at the beginning and in the end, the robot is momentarily still, resulting in a boost in confidence and explaining the U-shaped trend in the confidence plot, where confidence is highest when the robot is still and dips during phases of rapid motion.

Behavioral Analysis: Table 4.1 reports a 100% success rate for the evaluated runs. However, it is important to note that this figure excludes trials in which the robot lost balance and fell. These incidents were primarily due to hardware constraints: the robot's original center of mass, as configured in the CM-530 controller, was significantly altered by the addition of the backpack. Although the 3D-printed counterweight box filled with tungsten rods was attached to the torso to restore balance, its effectiveness was limited. First, the weight of the backpack's components (including the Raspberry Pi, fan, BT-100A dongles, and wireless HDMI dongle) was not evenly distributed. Second, the tungsten rods could shift within the box during motion, further destabilizing the robot. Finally, even though the counterweights nearly offset the backpack's weight, the robot was not designed to carry such a load in the first place, inevitably causing it to lose balance. As a result, some runs ended due to the robot losing balance and falling. These trials were excluded from the primary analysis, as they

reflect hardware limitations rather than a failure in algorithmic performance. If all runs were to be included, the success rate would drop to 83.33% (5 out of 6), which remains acceptable given the experimental challenges.

Beyond balance issues, the algorithm's reliability is further demonstrated by the low standard deviations in key metrics—particularly the average iteration time (±0.049) and traveled distance (±0.011)—indicating highly consistent performance across trials. The iteration time's efficiency (2 frames processed per second) ensures computational latency never becomes the bottleneck, as the time that it takes the robot to perform a movement is greater than the time it takes to process a frame. The short average kicking distance (mean 0.13 m, max 0.16 m) suggests the robot acts decisively once the ball enters its target range.

A final consideration is the robot's speed, currently averaging just one centimeter per second (Table 4.1). This limitation arises from the CM-530 controller, which cannot modulate speed; it can only issue movement commands such as "forward" or "left," with fixed parameters regardless of the situation. As a result, the robot is forced always to have the same pace regardless of the circumstances.

Error Analysis and Trajectory Behavior: The error analysis provides further insight into the robot's behavior during task execution. As described in Section 2.2.3, the robot's characteristic tendency to introduce a slight left or right turn during forward movement results in a zigzagging trajectory. This behavior is reflected in the horizontal error (μ), which shows high variability as the robot alternates between corrective maneuvers. Initially, the horizontal error is low because the robot starts almost perfectly aligned with the ball. However, as the robot advances, the directional bias leads to high variability depicted in the plot. Additionally, the sudden changes between positive and negative error values are a direct consequence of the frequent left and right turns the robot must execute along its trajectory. Despite these deviations, the robot is generally able to realign with the ball by the end of each run, resulting in a mean horizontal error that approaches zero in the final steps.

In contrast, the vertical error (ν) demonstrates a pronounced and steady decrease throughout the trials. This trend highlights the robot's consistent ability to reduce its distance to the ball, underscoring reliable performance in the forward (depth) direction even in the presence of lateral trajectory deviations.

5.2 Limitations and Future work

The most significant direction for future work is to completely remove the CM-530 microcontroller and transfer all computation and decision-making power to the Raspberry Pi 5. This change would allow upcoming investigations to address many of the current system's limitations and open the door to substantial improvements in both flexibility and performance.

One key limitation was the already discussed robot's characteristic tendency to veer left or right when moving forward. This issue is not solvable with the current configuration due to the restricted action set of the CM-530, which only allows for basic instructions such as "move right," "move left," "move forward," or "kick right." These instructions are executed the same every time they are issued, which does not support fine-grained control over the degree of turning, the speed of movement, or the kicking power. By shifting control to the Raspberry Pi, it would be possible to implement more nuanced behaviors, such as adjusting speed based on distance to the target or modulating the strength of a kick depending on the objective's location. This would also allow for software-based correction of the turning bias, leading to more accurate and reliable navigation.

Another limitation was imposed by the lack of mobility in the robot's neck. Because the robot's head is fixed, attaching the camera to the head would have resulted in a loss of visual contact with the ball, especially when the ball is close to the robot, which happens to be the most critical area. For this reason, the camera is currently mounted on the robot's hip. This is functional but can be problematic in more complex setups. A promising path for future work would be to add servos to the head to allow pan and tilt movements, or at a minimum, to enable tilting. This would expand the robot's field of view and improve object tracking.

Balance also posed a significant challenge. The addition of the backpack shifted the robot's center of mass, requiring a counterweight solution that proved to be only partially effective. Future work could involve recalculating and adjusting the center of mass directly or redesigning the robot's structure to achieve proper balance without relying on counterweights. The extra load from the backpack also led to increased strain on the servos, which ended up damaging the battery supplied with the Bioloid Kit.

Finally, the current setup requires the Raspberry Pi to be powered with an external cable, limiting the robot's autonomy. Identifying a suitable power bank that can power both the Raspberry Pi and all the servos would enable fully autonomous operation and further enhance the system's practical usability.

In summary, addressing these hardware and control limitations will not only improve the robot's robustness and autonomy but also create opportunities for more advanced behaviors and applications in future research.

6 Conclusion

This research successfully demonstrates the feasibility of implementing an autonomous humanoid robot capable of real-time detection, tracking, approaching, and kicking a ball on a resource-constrained framework. By integrating a YOLOv8n object detection model, monocular depth estimation, and a modular rule-based control system, we developed a complete autonomous system that enables a Bioloid Premium robot to detect, approach, and kick a ball using only a Raspberry Pi 5 for processing.

Among the key contributions of this work are (1) the development of a YOLOv8n detection model achieving high performance metrics (mPA@0.5 = 1.0); (2) the implementation of a monocular depth estimation framework using a wide-angle lens;and (3) the design of a modular rule-based controller capable of coordinating three different behaviours: searching, approaching, and kicking.

The experimental results confirm these claims, with the system achieving a 100% convergence rate in completed trials, demonstrating consistent performance and low variability in execution time.

This work demonstrates that autonomous behaviours can be achieved on affordable and accessible robotic platforms, contributing to the broader goal of making advanced robotics more accessible for educational and research applications. The modular architecture provides a solid foundation for future enhancements and more complex autonomous behaviors in humanoid robotics.

7 Acknowledgments

I'd like to express my gratitude to Robin Keekstra for his invaluable assistance in this research. He was responsible for designing and printing all the 3D components used in this study.

References

Ali, B., Ahmed, M., Md Zan, M. M., Hashim, H., et al. (2020). An alternative open architecture controller design for the bioloid humanoid robot/'aqilah zainuddin... [et al.]. *Journal of Electrical and Electronic Systems Research (JEESR)*, *16*, 1–9.

Bochkovskiy, A., Wang, C.-Y., & Liao, H.-Y. M. (2020). YOLOv4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*. Retrieved from `https://arxiv.org/abs/2004.10934`

Jiang, K. (2019a). *Calibrate fisheye lens using opencv.* Retrieved from `https://medium.com/@kennethjiang/calibrate-fisheye-lens-using-opencv-333b05afa0b0` (Accessed: 2025-06-11)

Jiang, K. (2019b). *Calibrate fisheye lens using opencv (part 2).* Retrieved from `https://medium.com/@kennethjiang/calibrate-fisheye-lens-using-opencv-part-2-13990f1b157f` (Accessed: 2025-06-11)

Julianda, R. R., Puriyanto, R. D., et al. (2024). Tracking ball using yolov8 method on wheeled soccer robot with omnidirectional camera. *Buletin Ilmiah Sarjana Teknik Elektro*, *6*(2), 203–213.

Kalyani, G. (2016). *A robot operating system (ros) based humanoid robot control* (Unpublished doctoral dissertation). Middlesex University.

Kwak, S. S. (2014). The impact of the robot appearance types on social interaction with a robot and service evaluation of a robot. *Archives of Design Research*, *27*(2), 81–93.

Opencv: Camera calibration and 3d reconstruction (fisheye model) [Computer software manual]. (2025). Retrieved from `https://docs.opencv.org/4.x/db/d58/group__calib3d__fisheye.html` (Accessed: 2025-06-11)

Prescott, T. J., & Robillard, J. M. (2021). Are friends electric? the benefits and risks of human-robot relationships. *Iscience*, *24*(1).

PyTorch Contributors. (2024). *Exporting a model from pytorch to onnx and running it using onnx runtime.* `https://docs.pytorch.org/tutorials/advanced/super_resolution_with_onnxruntime.html`. (Accessed: 2025-06-08)

Raspberry Pi Foundation. (2025). Raspberry pi documentation: Gpio [Computer software manual]. Retrieved from `https://www.raspberrypi.com/documentation/computers/raspberry-pi.html#gpio` (Accessed: 2025-06-05)

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the ieee conference on computer vision and pattern*

recognition (cvpr) (pp. 779–788). Retrieved from https://arxiv.org/abs/1506.02640

Robocup standard platform league publications. (2025). Retrieved from https://spl.robocup.org/publications/ (Accessed: 2025-05-18)

Robotis. (n.d.-a). *Bioloid premium.* https://emanual.robotis.com/docs/en/edu/bioloid/premium/. (Accessed: 2025-06-05)

Robotis. (n.d.-b). *Rc-100 user manual.* https://emanual.robotis.com/docs/en/parts/communication/rc-100/. (Accessed: 2025-06-05)

Robotis. (2025). Bioloid premium kit manual [Computer software manual]. Retrieved from https://emanual.robotis.com/docs/en/edu/bioloid/premium/

ROBOTIS. (2025). Bt-100/110a [Computer software manual]. Retrieved from https://emanual.robotis.com/docs/en/parts/communication/bt-110/ (Accessed: 2025-06-05)

Roques, A., & contributors. (2025). *Plantuml: Open-source tool to draw uml diagrams.* https://plantuml.com. (Version 1.2025.3, GPL license)

Upton, E., & Halfacree, G. (2016). *Raspberry pi user guide* (4th ed.). Wiley.

van den Boomgaard, R. (2017). *The pinhole camera matrix.* Retrieved from https://staff.fnwi.uva.nl/r.vandenboomgaard/IPCV20162017/LectureNotes/CV/PinholeCamera/PinholeCamera.html (Accessed: 2025-06-11)

Visser, U., & Burkhard, H.-D. (2007). Robocup: 10 years of achievements and future challenges. *AI magazine*, *28*(2), 115–115.

Wu, T., Zheng, H., Zheng, G., Huo, T., & Han, S. (2024). Do we empathize humanoid robots and humans in the same way? behavioral and multimodal brain imaging investigations. *Cerebral Cortex*, *34*(6), bhae248.

A Appendix A: Technology Stack

The robotic system was primarily developed using Python, with select components, such as the backpack and camera mount, implemented using alternative tools. The technology stack comprises a range of Python libraries for computer vision, serial communication, and data analysis, as well as specialized software for hardware design. The following outlines the key tools and libraries employed in the system's development.

A.1 Python Libraries

- **Computer Vision and Machine Learning**:
  - `ultralytics` and `roboflow`: Facilitated training of YOLO models for object detection.
  - `onnxruntime`, `opencv-python`, and `numpy`: Enabled model inference, image resizing, and non-maximum suppression for efficient object detection.
  - `opencv-python` and `picamera2`: Managed camera input for real-time image capture on the Raspberry Pi.
- **Serial Communication**:
  - `pyserial` and `time`: Handled serial communication between the Raspberry Pi and the CM-530 microcontroller.
- **Decision Visualization**:
  - `opencv-python` and `numpy`: Supported real-time visualization of the robot's perception (e.g., detected objects and centroids).
- **Data Analysis and Visualization**:
  - `pandas`, `seaborn`, and `matplotlib`: Utilized for analyzing simulation data and visualizing results, such as centroid trajectories and control actions.

A.2 Other Tools

- **SolidWorks**: Used for designing 3D models of the backpack, camera mount, and counterweight box.
- **PlantUML** (version 1.2025.3, GPL license): Used for generating UML diagrams to visually document and communicate the system architecture and data flow in the project. PlantUML enables customized diagram creation using a simple textual description, supporting reproducible and version-controlled documentation (Roques & contributors (2025)).

B Appendix B: Image-Based Visual Servoing (IBVS)

Although not integrated into the final control loop, an Image-Based Visual Servoing (IBVS) algorithm was developed for potential future applications. IBVS generates a two-component velocity command(v_x, ω_z), which instructs the robot to move forward/backward or rotate left/right, specifying the magnitude of these motions. The framework operates in the 2D image plane, utilizing the centroid (u, v) of the bounding box produced by the YOLO detection model and an ideal centroid(u', v'), representing the optimal ball position for kicking. The error vector is computed as:

$$\mathbf{e} = \begin{bmatrix} u' - u \\ v' - v \end{bmatrix} \tag{B.1}$$

Here, $\mathbf{e}$ represents the displacement between the current and desired centroid positions in the image plane, with components:

- $u' - u$: Horizontal error (pixels) in the image x-axis.
- $v' - v$: Vertical error (pixels) in the image y-axis.

The error vector is related to camera motion through the interaction matrix (Jacobian), which maps image feature velocities to camera velocities. Due to the fixed camera orientation and the robot's planar motion constraints on flat ground, most components of the standard 6x2 interaction matrix are zero or cancel out, simplifying it to a 2x2 form:

$$\mathbf{L}_p = \begin{bmatrix} -\frac{f_x}{Z} & 0 \\ v - c_y & -(u - c_x) \end{bmatrix} \tag{B.2}$$

The components of $\mathbf{L}_p$ are:

- f_x: Focal length in pixels along the x-axis, derived from the camera intrinsic matrix.
- Z: Depth (distance) to the ball, assumed constant or estimated.
- $v - c_y$: Vertical offset of the centroid from the principal pointc_y.
- $u - c_x$: Horizontal offset of the centroid from the principal pointc_x.

The camera velocity is then computed using the pseudoinverse of the interaction matrix and a control gain λ:

$$\mathbf{v}_c = -\lambda \mathbf{L}_p^{+} \mathbf{e} \tag{B.3}$$

where:

- $\mathbf{v}_c = [v_x, \omega_z]^T$: Camera velocity vector, with v_x linear velocity (forward/backward) and ω_z as angular velocity (rotation).
- $\mathbf{L}_p^+$: Moore-Penrose pseudoinverse of $\mathbf{L}_p$, ensuring stable inversion.
- λ: Control gain, tuning the responsiveness of the velocity command.
- $\mathbf{e}$: Error vector defined above.

This formulation enables the robot to adjust its motion to minimize the error $\mathbf{e}$, aligning the ball's centroid with the ideal position.

C Appendix C: Architectural Design

Figure C.1: Diagram showing the main components of the architecture and how they interact in the main loop.