InternPro Weekly Progress Update

Name	Email	Project Name	NDA/ Non-NDA	InternPro Start Date	OPT
Adharsh Prasad Natesan	anatesan@asu.edu	IT-Core Foundation Suriname	Non-NDA	2024-08-05	Yes

Progress

Include an itemized list of the tasks you completed this week.

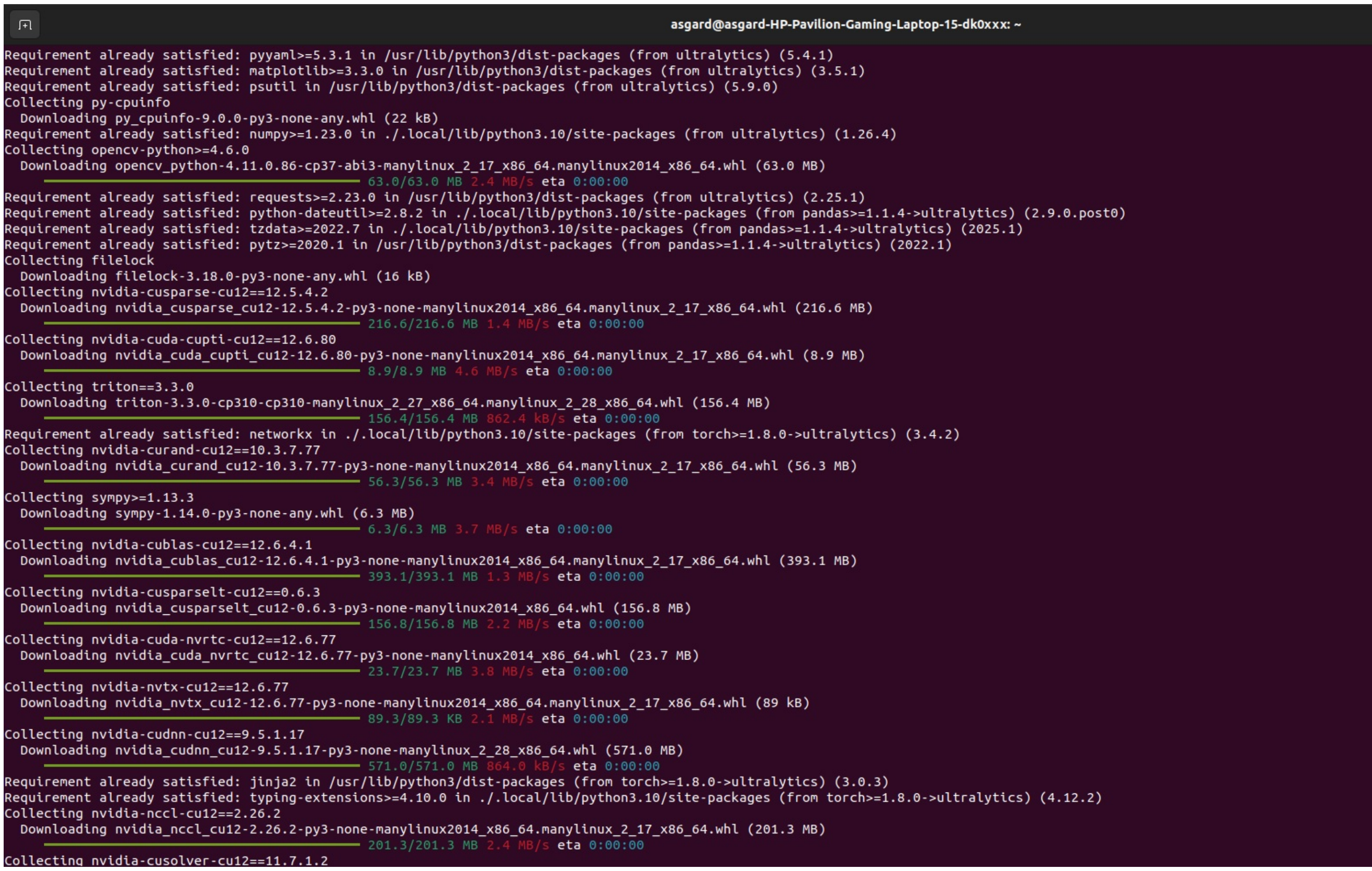
#	Action Item/ Explanation	Total Time This Week (hours)
1	Creating the YOLO Inference Workspace and Installing Required Packages	3
2	Writing the YOLOv8 Inference Node in ROS2	3
3	Run YOLOv8 Node and View Annotated Output in ROS2	1
4	Saving Annotated YOLO Images to a Designated Directory	3
5	Dist-YOLO: Fast Object Detection with Distance Estimation	3
6	Estimating Distance to Detected Crops Using Bounding Box Height	3
7	Review of the field environmental sensing methods based on multi-sensor information fusion technology	3
8	Report Writing	1
	Total hours for the week:	20

Verification Documentation:

Action Item 1: Creating the YOLO Inference Workspace and Installing Required Packages – 3 hour(s).

Project Work Summary

- To kick off object detection for the Farmbot project, I began by preparing the workspace environment specifically for YOLOv8 integration. Since YOLOv8 isn't natively supported by ROS2 yet, I decided to treat the YOLO inference as a Python-side utility that can be wrapped inside a ROS2 node, leveraging the clean API provided by the Ultralytics library.
- Did not create a separate package, since the existing farmbot_autonomous_pkg already contained vision-related nodes. Instead, I chose to integrate the new detection functionality directly into this package, maintaining consistency and reusing the same camera topic subscriptions.
- Before diving into coding, I installed the ultralytics package via pip — a process that took a bit longer than expected due to its dependencies and build size. I used the following command:pip install ultralytics
- This also pulled in required dependencies like PyTorch, which added to the installation time but was necessary for GPU/CPU inference. After the installation, I ran a quick import test in a Python shell to confirm that the YOLO class was recognized, and also verified that the yolo CLI worked on test images.
- Also confirmed that OpenCV and cv_bridge were already working in my ROS2 environment, both were essential for image decoding from the simulation camera. Also installed the same package in the terminal of the VS code to make the VS code loaded with the package.
- With all the relevant packages installed and tested, the workspace was ready for building a detection pipeline. The next step was to start coding the actual ROS2 inference node.



```
asgard@asgard-HP-Pavilion-Gaming-Laptop-15-dk0xxx: ~
Requirement already satisfied: pyyaml>=5.3.1 in /usr/lib/python3/dist-packages (from ultralytics) (5.4.1)
Requirement already satisfied: matplotlib>=3.3.0 in /usr/lib/python3/dist-packages (from ultralytics) (3.5.1)
Requirement already satisfied: psutil in /usr/lib/python3/dist-packages (from ultralytics) (5.9.0)
Collecting py-cpuinfo
  Downloading py_cpuinfo-9.0.0-py3-none-any.whl (22 kB)
Requirement already satisfied: numpy>=1.23.0 in ./.local/lib/python3.10/site-packages (from ultralytics) (1.26.4)
Collecting opencv-python>=4.6.0
  Downloading opencv_python-4.11.0.86-cp37-abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (63.0 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 63.0/63.0 MB 2.4 MB/s eta 0:00:00
Requirement already satisfied: requests>=2.23.0 in /usr/lib/python3/dist-packages (from ultralytics) (2.25.1)
Requirement already satisfied: python-dateutil>=2.8.2 in ./.local/lib/python3.10/site-packages (from pandas>=1.1.4->ultralytics) (2.9.0.post0)
Requirement already satisfied: tzdata>=2022.7 in ./.local/lib/python3.10/site-packages (from pandas>=1.1.4->ultralytics) (2025.1)
Requirement already satisfied: pytz>=2020.1 in /usr/lib/python3/dist-packages (from pandas>=1.1.4->ultralytics) (2022.1)
Collecting filelock
  Downloading filelock-3.18.0-py3-none-any.whl (16 kB)
Collecting nvidia-cusparse-cu12==12.5.4.2
  Downloading nvidia_cusparse_cu12-12.5.4.2-py3-none-manylinux2014_x86_64.manylinux_2_17_x86_64.whl (216.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 216.6/216.6 MB 1.4 MB/s eta 0:00:00
Collecting nvidia-cuda-cupti-cu12==12.6.80
  Downloading nvidia_cuda_cupti_cu12-12.6.80-py3-none-manylinux2014_x86_64.manylinux_2_17_x86_64.whl (8.9 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8.9/8.9 MB 4.6 MB/s eta 0:00:00
Collecting triton==3.3.0
  Downloading triton-3.3.0-cp310-cp310-manylinux_2_27_x86_64.manylinux_2_28_x86_64.whl (156.4 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 156.4/156.4 MB 862.4 kB/s eta 0:00:00
Requirement already satisfied: networkx in ./.local/lib/python3.10/site-packages (from torch>=1.8.0->ultralytics) (3.4.2)
Collecting nvidia-curand-cu12==10.3.7.77
  Downloading nvidia_curand_cu12-10.3.7.77-py3-none-manylinux2014_x86_64.manylinux_2_17_x86_64.whl (56.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 56.3/56.3 MB 3.4 MB/s eta 0:00:00
Collecting sympy>=1.13.3
  Downloading sympy-1.14.0-py3-none-any.whl (6.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6.3/6.3 MB 3.7 MB/s eta 0:00:00
Collecting nvidia-cublas-cu12==12.6.4.1
  Downloading nvidia_cublas_cu12-12.6.4.1-py3-none-manylinux2014_x86_64.manylinux_2_17_x86_64.whl (393.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 393.1/393.1 MB 1.3 MB/s eta 0:00:00
Collecting nvidia-cusparselt-cu12==0.6.3
  Downloading nvidia_cusparselt_cu12-0.6.3-py3-none-manylinux2014_x86_64.whl (156.8 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 156.8/156.8 MB 2.2 MB/s eta 0:00:00
Collecting nvidia-cuda-nvrtc-cu12==12.6.77
  Downloading nvidia_cuda_nvrtc_cu12-12.6.77-py3-none-manylinux2014_x86_64.whl (23.7 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 23.7/23.7 MB 3.8 MB/s eta 0:00:00
Collecting nvidia-nvtx-cu12==12.6.77
  Downloading nvidia_nvtx_cu12-12.6.77-py3-none-manylinux2014_x86_64.manylinux_2_17_x86_64.whl (89 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 89.3/89.3 KB 2.1 MB/s eta 0:00:00
Collecting nvidia-cudnn-cu12==9.5.1.17
  Downloading nvidia_cudnn_cu12-9.5.1.17-py3-none-manylinux_2_28_x86_64.whl (571.0 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 571.0/571.0 MB 864.0 kB/s eta 0:00:00
Requirement already satisfied: jinja2 in /usr/lib/python3/dist-packages (from torch>=1.8.0->ultralytics) (3.0.3)
Requirement already satisfied: typing-extensions>=4.10.0 in ./.local/lib/python3.10/site-packages (from torch>=1.8.0->ultralytics) (4.12.2)
Collecting nvidia-nccl-cu12==2.26.2
  Downloading nvidia_nccl_cu12-2.26.2-py3-none-manylinux2014_x86_64.manylinux_2_17_x86_64.whl (201.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 201.3/201.3 MB 2.4 MB/s eta 0:00:00
Collecting nvidia-cusolver-cu12==11.7.1.2
```

Action Item 2: Writing the YOLOv8 Inference Node in ROS2 – 3 hour(s).

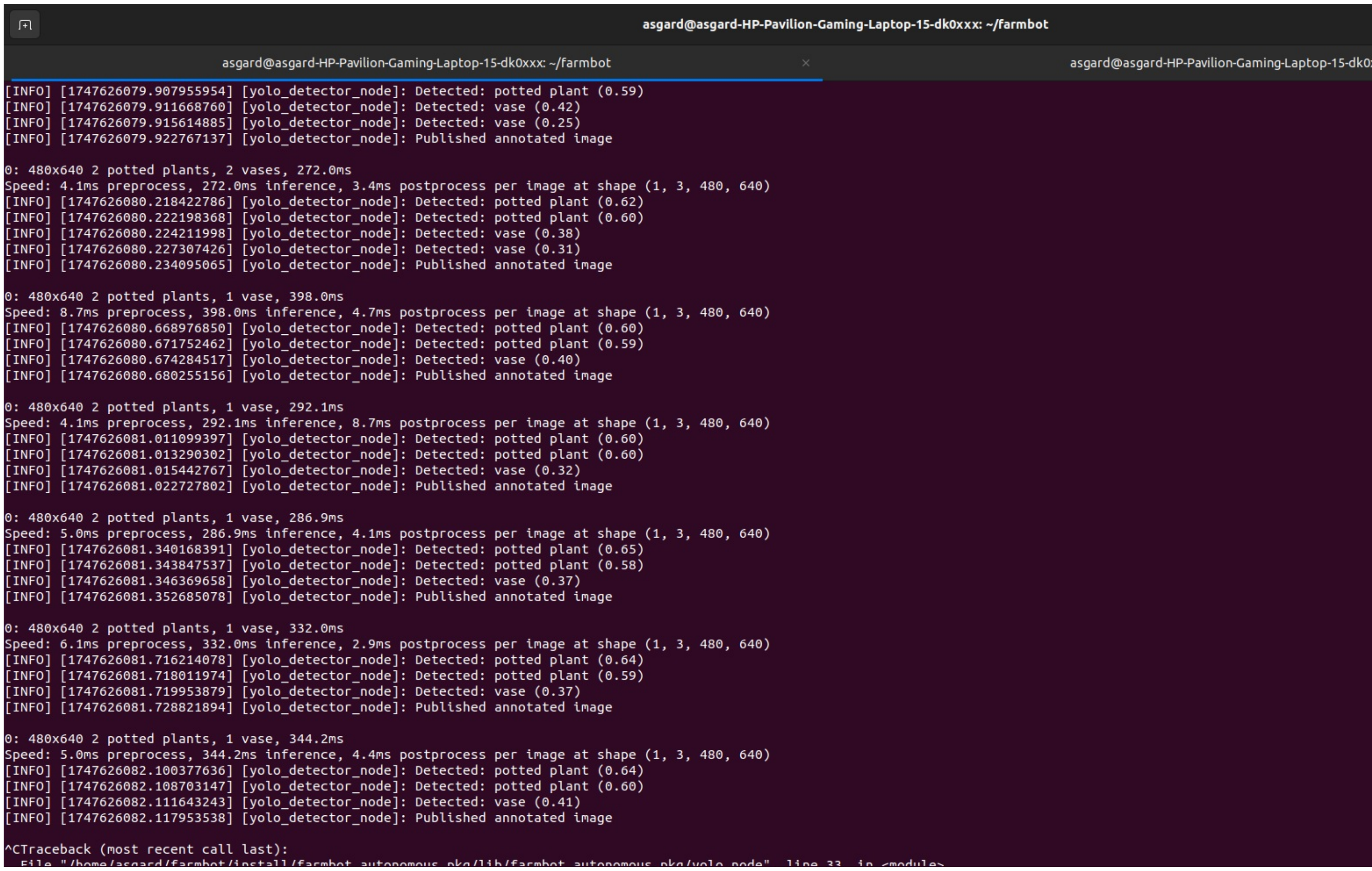
Project Work Summary

- With the environment now set up and YOLO installed, I proceeded to build the actual inference logic. The focus here was to implement a lightweight ROS2 node that could perform real-time object detection on incoming camera feeds and validate whether the YOLOv8 model was functioning as expected within the ROS2 pipeline.
- Inside the farmbot_autonomous_pkg, I created a new file called yolo_node.py under the internal farmbot_autonomous_pkg/ directory, which already houses my other perception nodes. This ensured the node remained logically grouped with existing vision modules.
- The script initialized a ROS2 node that subscribed to /front_camera/image_raw. I used cv_bridge to convert ROS image messages into OpenCV frames, and then passed these frames to the YOLO('yolov8n.pt') model for inference. For each detected object, I printed the class label and confidence score to the console, avoiding visualization or publishing at this stage to keep the implementation clean and focused.
- After coding the node, I completed two essential steps to ensure it could be run from the command line: Made the script executable. Inside the farmbot_autonomous_pkg/farmbot_autonomous_pkg directory, I ran: chmod +x yolo_node.py. This allowed the script to be invoked directly through ROS2's launch system. Added the node to setup.py
- I registered the node in the entry_points section of setup.py so it could be executed via ros2 run. My updated entry looked like this: yolo_node = farmbot_autonomous_pkg.yolo_node:main',
- After making these changes, I rebuilt the workspace with colcon build, sourced it, and tested the node using: ros2 run farmbot_autonomous_pkg yolo_node
- This successful run completed the foundation for YOLO-based object detection. With the core node in place, the next goal is to add annotated image publishing and message structuring for downstream modules to consume.

Action Item 3: Run YOLOv8 Node and View Annotated Output in ROS2 – 1 hour(s).

Project Work Summary

- To verify the functionality of the YOLOv8 inference pipeline, I allocated a short session to test the existing ROS2 node in conjunction with the Gazebo camera feed. The objective was to confirm that recognized objects from the simulation environment were being detected and printed correctly via the YOLO model.
- Launched the YOLO detection node using the command `ros2 run farmbot_autonomous_pkg yolo_node` and ensured that the simulated camera in Gazebo was active and publishing to `/front_camera/image_raw`. Monitored the terminal output from the node to check for class labels and confidence scores being logged in real time, indicating that image frames were successfully received and YOLO inference was operational.
- Let the node run for a few minutes, verifying consistency in detections and ensuring that the node responded to visual changes in the environment, which confirmed the end-to-end functionality of the recognition pipeline.



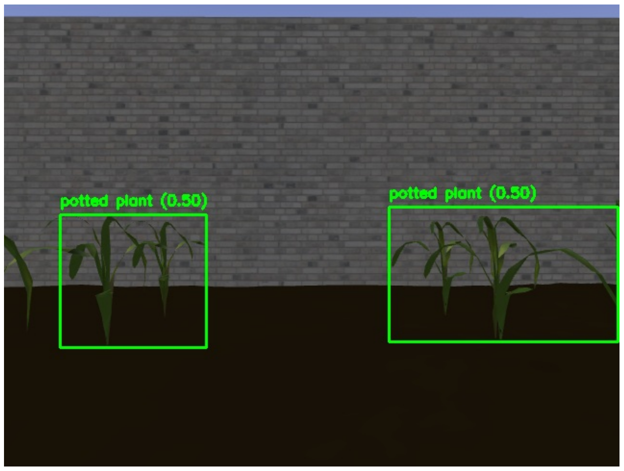
asgard@asgard-HP-Pavilion-Gaming-Laptop-15-dk0xxx: ~/farmbot

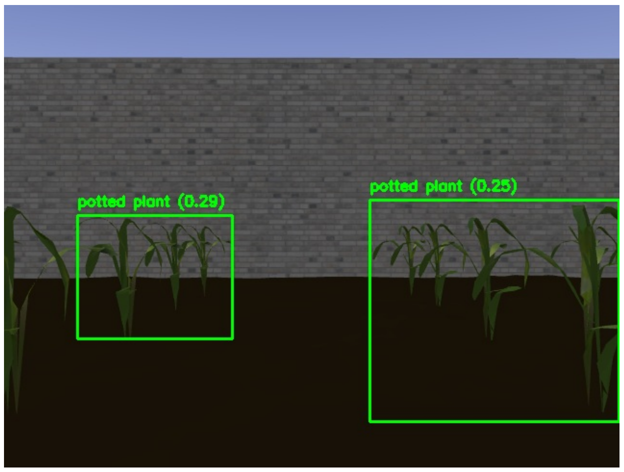
```
[INFO] [1747626079.907955954] [yolo_detector_node]: Detected: potted plant (0.59)
[INFO] [1747626079.911668760] [yolo_detector_node]: Detected: vase (0.42)
[INFO] [1747626079.915614885] [yolo_detector_node]: Detected: vase (0.25)
[INFO] [1747626079.922767137] [yolo_detector_node]: Published annotated image

0: 480x640 2 potted plants, 2 vases, 272.0ms
Speed: 4.1ms preprocess, 272.0ms inference, 3.4ms postprocess per image at shape (1, 3, 480, 640)
[INFO] [1747626080.218422786] [yolo_detector_node]: Detected: potted plant (0.62)
[INFO] [1747626080.222198368] [yolo_detector_node]: Detected: potted plant (0.60)
[INFO] [1747626080.224211998] [yolo_detector_node]: Detected: vase (0.38)
[INFO] [1747626080.227307426] [yolo_detector_node]: Detected: vase (0.31)
[INFO] [1747626080.234095065] [yolo_detector_node]: Published annotated image

0: 480x640 2 potted plants, 1 vase, 398.0ms
Speed: 8.7ms preprocess, 398.0ms inference, 4.7ms postprocess per image at shape (1, 3, 480, 640)
[INFO] [1747626080.668976850] [yolo_detector_node]: Detected: potted plant (0.60)
[INFO] [1747626080.671752462] [yolo_detector_node]: Detected: potted plant (0.59)
[INFO] [1747626080.674284517] [yolo_detector_node]: Detected: vase (0.40)
[INFO] [1747626080.680255156] [yolo_detector_node]: Published annotated image

0: 480x640 2 potted plants, 1 vase, 292.1ms
Speed: 4.1ms preprocess, 292.1ms inference, 8.7ms postprocess per image at shape (1, 3, 480, 640)
[INFO] [1747626081.011099397] [yolo_detector_node]: Detected: potted plant (0.60)
[INFO] [1747626081.013290302] [yolo_detector_node]: Detected: potted plant (0.60)
[INFO] [1747626081.015442767] [yolo_detector_node]: Detected: vase (0.32)
[INFO] [1747626081.022727802] [yolo_detector_node]: Published annotated image

0: 480x640 2 potted plants, 1 vase, 286.9ms
Speed: 5.0ms preprocess, 286.9ms inference, 4.1ms postprocess per image at shape (1, 3, 480, 640)
[INFO] [1747626081.340168391] [yolo_detector_node]: Detected: potted plant (0.65)
[INFO] [1747626081.343847537] [yolo_detector_node]: Detected: potted plant (0.58)
[INFO] [1747626081.346369658] [yolo_detector_node]: Detected: vase (0.37)
[INFO] [1747626081.352685078] [yolo_detector_node]: Published annotated image

0: 480x640 2 potted plants, 1 vase, 332.0ms
Speed: 6.1ms preprocess, 332.0ms inference, 2.9ms postprocess per image at shape (1, 3, 480, 640)
[INFO] [1747626081.716214078] [yolo_detector_node]: Detected: potted plant (0.64)
[INFO] [1747626081.718011974] [yolo_detector_node]: Detected: potted plant (0.59)
[INFO] [1747626081.719953879] [yolo_detector_node]: Detected: vase (0.37)
[INFO] [1747626081.728821894] [yolo_detector_node]: Published annotated image

0: 480x640 2 potted plants, 1 vase, 344.2ms
Speed: 5.0ms preprocess, 344.2ms inference, 4.4ms postprocess per image at shape (1, 3, 480, 640)
[INFO] [1747626082.100377636] [yolo_detector_node]: Detected: potted plant (0.64)
[INFO] [1747626082.108703147] [yolo_detector_node]: Detected: potted plant (0.60)
[INFO] [1747626082.111643243] [yolo_detector_node]: Detected: vase (0.41)
[INFO] [1747626082.117953538] [yolo_detector_node]: Published annotated image

^CTraceback (most recent call last):
  File "/home/asgard/farmbot/install/farmbot_autonomous_pkg/lib/farmbot_autonomous_pkg/yolo_node", line 33, in <module>
```

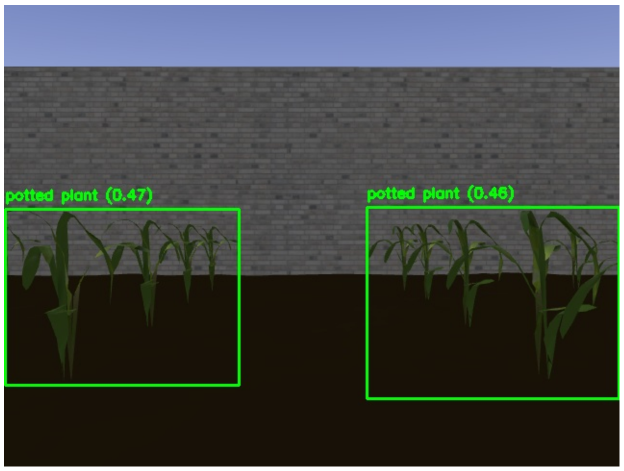
Action Item 4: Saving Annotated YOLO Images to a Designated Directory – 3 hour(s).

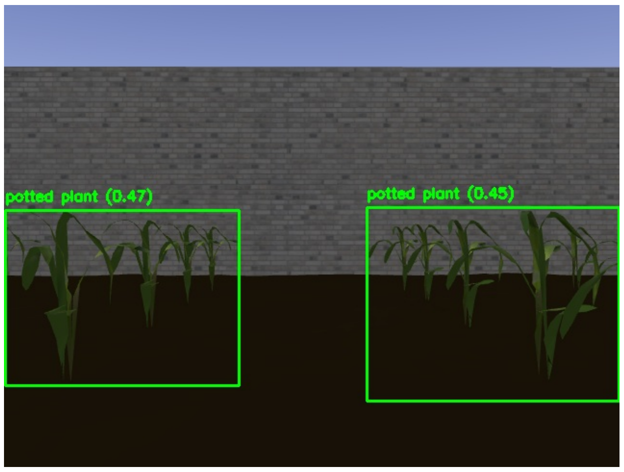
Project Work Summary

- After successfully overlaying YOLO detections on the camera feed and publishing the annotated images in ROS2, I wanted to retain a copy of those images for offline analysis and project documentation. The goal was to store each frame as a .png file in a clean, organized location outside the ROS2 message system, making it easier to review inference results or create visual logs of the simulation.
- To start, I created a dedicated directory named videos under the farmbot/ root folder to hold the annotated frames. This path structure keeps perception outputs centralized while maintaining separation from the package codebase. The full path used was: farmbot/videos/
- I then modified the existing YOLO node to include logic for saving each processed frame. Inside the node's constructor, I added logic to create the videos directory dynamically if it didn't exist, using Python's os.makedirs() with the exist_ok=True flag to avoid errors on re-runs. I also initialized a frame_count variable to uniquely index saved images.
- In the image callback, right after drawing bounding boxes and publishing the annotated image, I added a write step using OpenCV's cv2.imwrite() function. The output files were named using a zero-padded frame count (e.g., frame_0000.png, frame_0001.png) to ensure ordered storage. Each frame was saved to the farmbot/videos folder immediately after annotation, and a log message was printed to confirm successful write.
- This enhancement now allows me to visually archive YOLO detections frame by frame during any run of the simulation. The image files are readily accessible, timestamped through their sequence number, and useful for generating visual results, debugging detection accuracy, or creating future demos of the perception system in action.



- 

- 

- 

-

Action Item 5: Dist-YOLO: Fast Object Detection with Distance Estimation – 3 hour(s).

Research

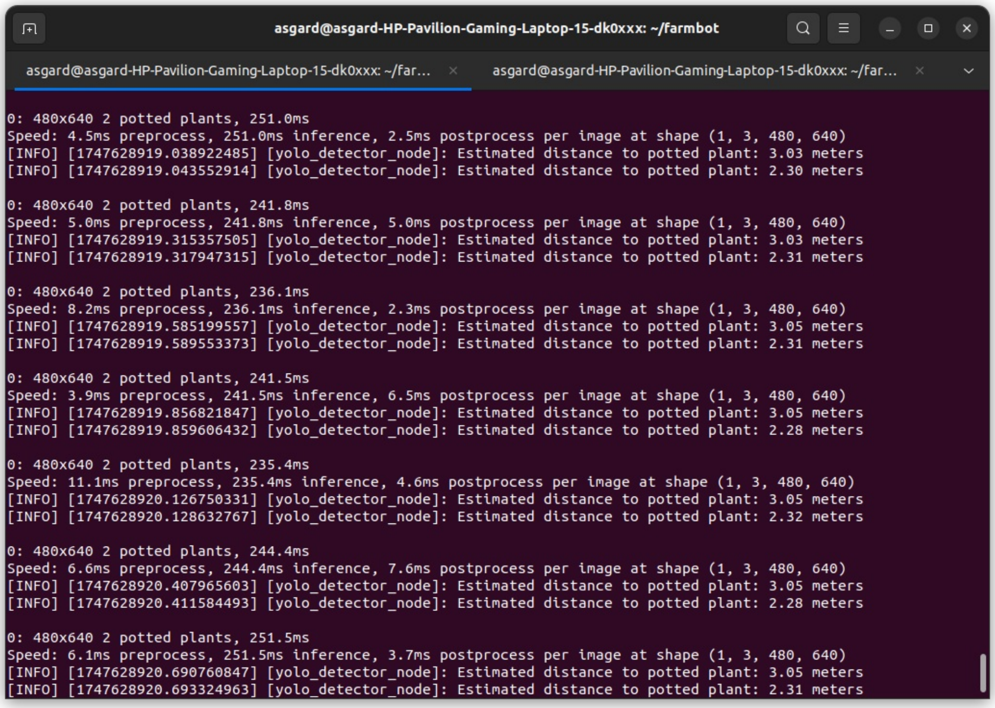
- https://www.mdpi.com/2076-3417/12/3/1354
- Dist-YOLO: Fast Object Detection with Distance Estimation
- Summary of Report
    - This paper introduces Dist-YOLO, an enhanced version of the YOLO object detection framework that integrates distance estimation capabilities using only a monocular camera. The authors propose modifications to the YOLO architecture by extending the prediction vectors to include distance information, sharing the backbone's weights with the bounding box regressor, and updating the loss function to account for distance estimation.
    - They proposed an architecture by which the YOLO network is augmented to predict absolute distances by adding a distance estimation component to the prediction vectors.
    - The model is trained using the KITTI dataset, which provides real-world driving scenarios with annotated distances, enabling the network to learn object detection and distance estimation simultaneously.
    - The proposed Dist-YOLO achieves a mean relative error of 11% across eight object classes within a distance range of 0 to 150 meters. Notably, this enhancement does not compromise the inference speed, maintaining a real-time performance of 45 frames per second.
- Relation to Project
    - In our Farmbot project, accurately determining the distance between the rover and detected crops is crucial for tasks such as navigation, targeted interventions, and yield estimation. The Dist-YOLO framework offers a promising solution by enabling real-time object detection coupled with distance estimation using only a single RGB camera.
    - By knowing the exact distance to each plant, the rover can plan optimal paths, avoiding obstacles and minimizing crop disturbance.
    - Perform Targeted Actions: Precise distance measurements allow for accurate application of treatments (e.g., watering, fertilizing) directly to individual plants. This approach will make the navigation through the fields efficiently..
- Motivation for Research
    - Our current setup utilizes YOLOv8 for object detection, successfully identifying crops within the field. However, the system lacks the capability to estimate the distance to these detected objects, limiting its effectiveness in real-world applications.
    - The Dist-YOLO paper provides insights into enhancing our existing object detection framework to include distance estimation without the need for additional hardware like stereo cameras or LiDAR. This aligns with our project's goals of maintaining a cost-effective and efficient system.
    - By adopting the methodologies proposed in this paper, we can integrate distance estimation into our current YOLOv8-based detection system, achieve real-time performance without

significant computational overhead and provide precise distance measurements to support various agricultural tasks.

Action Item 6: Estimating Distance to Detected Crops Using Bounding Box Height – 3 hour(s).

Project Work Summary

- After successfully detecting crops in the simulation and overlaying annotated bounding boxes, I wanted to enhance the node's perception capabilities by estimating how far each detected object—specifically the maize plants—was from the robot. The goal was to extract a rough, real-time distance in meters for each detection based on its appearance in the image frame, without requiring stereo vision or external localization systems.
- To do this, I implemented a simple pinhole camera-based distance estimation technique using the known physical height of the object (in this case, assumed to be 1 meter) and the height of the object's bounding box in pixels. The formula used was: distance = (real_object_height * focal_length_pixels) / bbox_pixel_height
- Retrieved the focal length directly from the camera's ROS2 /camera_info topic, using the fx value (554.38 pixels) from the intrinsic matrix. This value was hardcoded into the node for now, though it can be made dynamic later. I also added a constant for the known height of the crop (1.0 meter).
- Inside the YOLO node's image_callback, extracted the bounding box coordinates for each detected object, computed the vertical pixel height, and applied the distance formula. I logged the resulting estimated distance to the console for each detection and overlaid it on the image frame, just beneath the bounding box, using OpenCV's putText() function. This visual cue gives immediate spatial context when viewing detections in rqt_image_view.
- Also added a safeguard to skip estimation if the bounding box height was zero, avoiding divide-by-zero errors. The output image continued to be published to the /yolo/image_annotated topic as before, now with both class labels and distance values rendered on the frame.
- This enhancement now allows the Farmbot system to not only detect crops visually but also reason about their spatial proximity using just a monocular camera, a valuable step toward enabling proximity-aware control and decision-making in field operations.



```
0: 480x640 2 potted plants, 251.0ms
Speed: 4.5ms preprocess, 251.0ms inference, 2.5ms postprocess per image at shape (1, 3, 480, 640)
[INFO] [1747628919.038922485] [yolo_detector_node]: Estimated distance to potted plant: 3.03 meters
[INFO] [1747628919.043552914] [yolo_detector_node]: Estimated distance to potted plant: 2.30 meters

0: 480x640 2 potted plants, 241.8ms
Speed: 5.0ms preprocess, 241.8ms inference, 5.0ms postprocess per image at shape (1, 3, 480, 640)
[INFO] [1747628919.315357505] [yolo_detector_node]: Estimated distance to potted plant: 3.03 meters
[INFO] [1747628919.317947315] [yolo_detector_node]: Estimated distance to potted plant: 2.31 meters

0: 480x640 2 potted plants, 236.1ms
Speed: 8.2ms preprocess, 236.1ms inference, 2.3ms postprocess per image at shape (1, 3, 480, 640)
[INFO] [1747628919.585199557] [yolo_detector_node]: Estimated distance to potted plant: 3.05 meters
[INFO] [1747628919.589553373] [yolo_detector_node]: Estimated distance to potted plant: 2.31 meters

0: 480x640 2 potted plants, 241.5ms
Speed: 3.9ms preprocess, 241.5ms inference, 6.5ms postprocess per image at shape (1, 3, 480, 640)
[INFO] [1747628919.856821847] [yolo_detector_node]: Estimated distance to potted plant: 3.05 meters
[INFO] [1747628919.859606432] [yolo_detector_node]: Estimated distance to potted plant: 2.28 meters

0: 480x640 2 potted plants, 235.4ms
Speed: 11.1ms preprocess, 235.4ms inference, 4.6ms postprocess per image at shape (1, 3, 480, 640)
[INFO] [1747628920.126750331] [yolo_detector_node]: Estimated distance to potted plant: 3.05 meters
[INFO] [1747628920.128632767] [yolo_detector_node]: Estimated distance to potted plant: 2.32 meters

0: 480x640 2 potted plants, 244.4ms
Speed: 6.6ms preprocess, 244.4ms inference, 7.6ms postprocess per image at shape (1, 3, 480, 640)
[INFO] [1747628920.407965603] [yolo_detector_node]: Estimated distance to potted plant: 3.05 meters
[INFO] [1747628920.411584493] [yolo_detector_node]: Estimated distance to potted plant: 2.28 meters

0: 480x640 2 potted plants, 251.5ms
Speed: 6.1ms preprocess, 251.5ms inference, 3.7ms postprocess per image at shape (1, 3, 480, 640)
[INFO] [1747628920.690760847] [yolo_detector_node]: Estimated distance to potted plant: 3.05 meters
[INFO] [1747628920.693324963] [yolo_detector_node]: Estimated distance to potted plant: 2.31 meters
```

asgard@asgard-HP-Pavilion-Gaming-Laptop-15-dk0xxx: ~/farmbot
asgard@asgard-HP-Pavilion-Gaming-Laptop-15-dk0xxx: ~/far...
asgard@asgard-HP-Pavilion-Gaming-Laptop-15-dk0xxx: ~/far...

Action Item 7: Review of the field environmental sensing methods based on multi-sensor information fusion technology – 3 hour(s).

Project Work Summary

- Review of the field environmental sensing methods based on multi-sensor information fusion technology
- https://www.ijabe.org/index.php/ijabe/article/view/8596
- Summary of Report
  - This review explores state-of-the-art sensor fusion techniques designed to support robust navigation and positioning in precision agriculture. It categorizes common sensors — GNSS, inertial units (IMUs), vision systems, and wheel odometry — and compares fusion strategies used to overcome environmental limitations like occlusion, GNSS loss, and terrain variability.
  - It discusses both loosely coupled and tightly coupled fusion architectures. Loosely coupled systems typically run independent localization subsystems (like GNSS and VO) and then merge results, while tightly coupled systems perform joint optimization, integrating raw measurements at the filter level (e.g., EKF or factor graphs).
  - The authors highlight the role of Extended Kalman Filters (EKFs) as the most common fusion method due to their simplicity and effectiveness in agricultural vehicles. However, they also mention the advantages of graph-based SLAM techniques when accurate landmark association is possible.
  - the paper addresses real-world agricultural constraints — such as dusty environments, intermittent GNSS signals under canopy cover, and visually repetitive scenes — and how fusion techniques must adapt. Integration of semantic understanding (e.g., detecting plants or furrows) is briefly touched on as an emerging trend.
- Relation to Project
  - This paper helps me with next steps forward, combining GNSS and visual odometry to achieve a robust localization stream for the farmbot rover. It justifies the use of an EKF-based fusion pipeline, which I plan to implement next using either ROS2's robot_localization package or a custom-built filter.
  - The distinction between loosely and tightly coupled systems helped clarify why my current loosely coupled setup (separate VO and GNSS nodes) suffers from drift and jumps. The paper reinforced the need to bring raw measurements together into a single fusion node.
  - Another key insight was that landmark detection (e.g., YOLO-based crop detection) could potentially serve as a semantic constraint or correction mechanism. This gives me a direction to experiment with later using recurring plant rows as pseudo-landmarks to refine heading.
  - Finally, the paper's examples from vineyard and greenhouse navigation scenarios mirrored the challenges I'm seeing in simulation like erratic VO under repeated textures — and underscored the practical value of fusing multiple sensing modalities.
- Motivation for Research
  - I chose this paper to better understand how sensor fusion is applied in real-world agricultural robotics, particularly in cost-effective outdoor systems. The goal was to identify a framework that could integrate GNSS and VO with minimal drift and without requiring expensive sensors like LiDAR.
  - Reading this helped me decide that an EKF is the right starting point, given its balance of complexity and performance in outdoor robotics. It also exposed limitations of loosely coupled methods, which aligns with the limitations I'm observing in simulation.
  - Now have a clear plan to implement GNSS and VO fusion through a central filtering node, followed by potential extensions involving object-level constraints from the YOLO detector.

Action Item 8: Report Writing – 1 hour(s).

Project Work Summary

- Created word document layout to write contents of the weekly progress.
- Created relevant subsections in the epicspro website and documented 20 hours of weekly progress.
- Collected relevant documents research papers, relevant links and company's objective from their portal.