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.

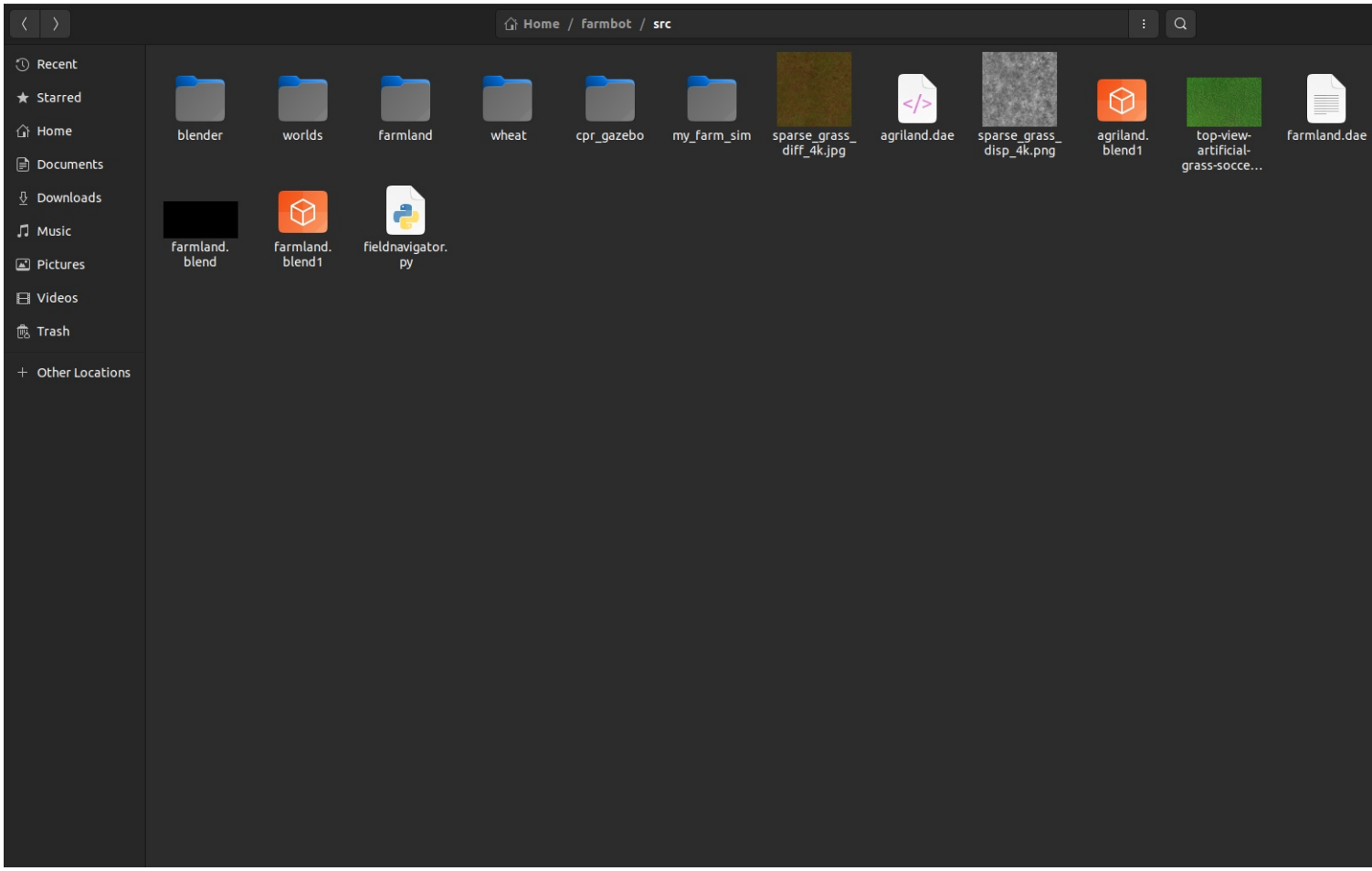
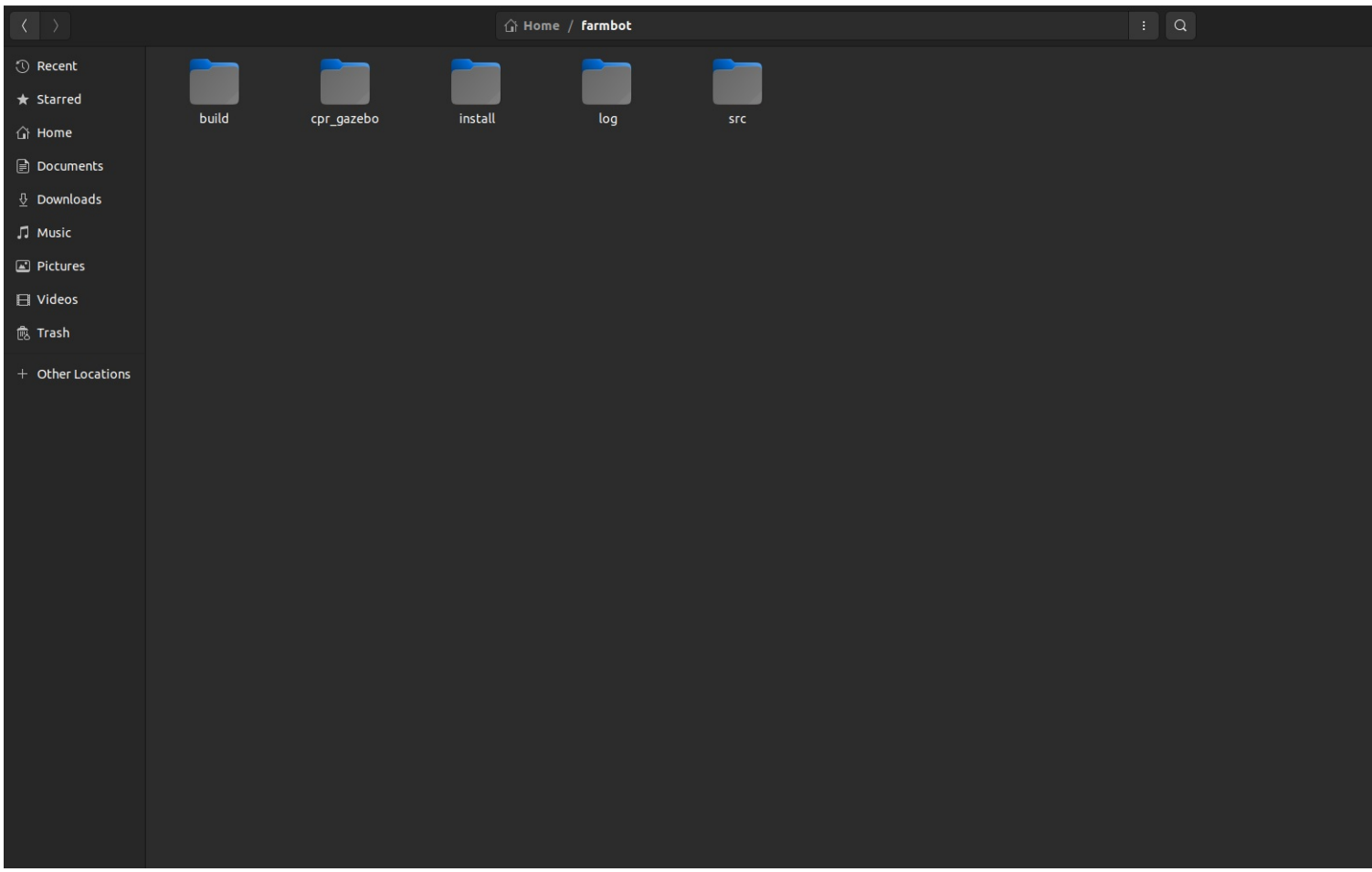
#	Action Item/ Explanation	Total Time This Week (hours)
1	ROS2 Package creation and Workspace Cleanup	3
2	Debugging Motion Plugin Failures and First successful Teleoperation Control for the Husky Rover	3
3	Developing a Vision Node for Camera Feed Processing and Local Video Recording	3
4	Integrate and Deploy Visual Odometry Pipeline (Phase 1 – Feature Matching)	3
5	Pose Estimation from Monocular Visual Odometry	3
6	ORB-SLAM: A Versatile and Accurate Monocular SLAM System	3
7	Weekly Task Plan: Advancing VO Pipeline Toward Sensor Fusion	1
8	Report Writing	1
	Total hours for the week:	20

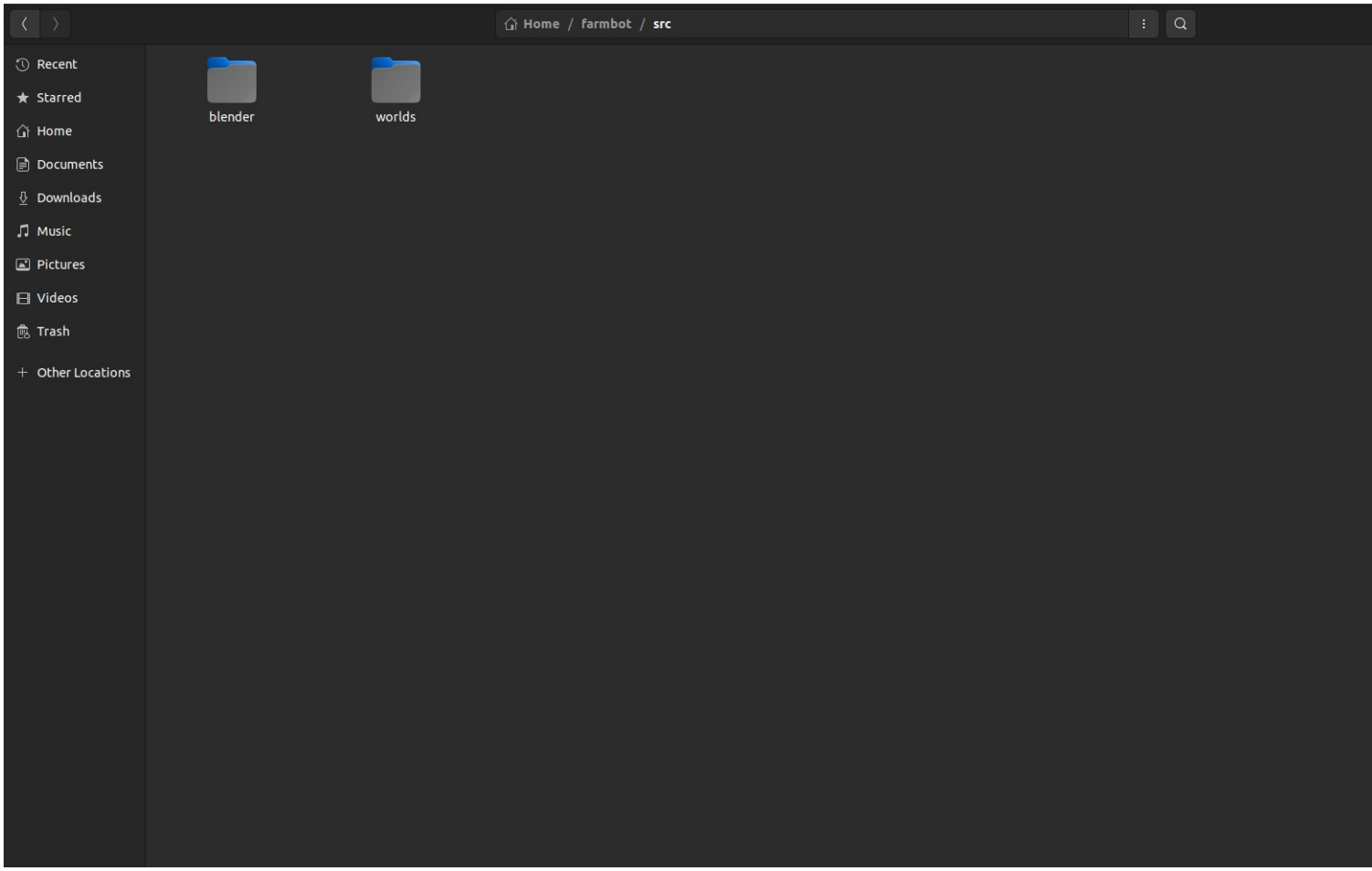
Verification Documentation:

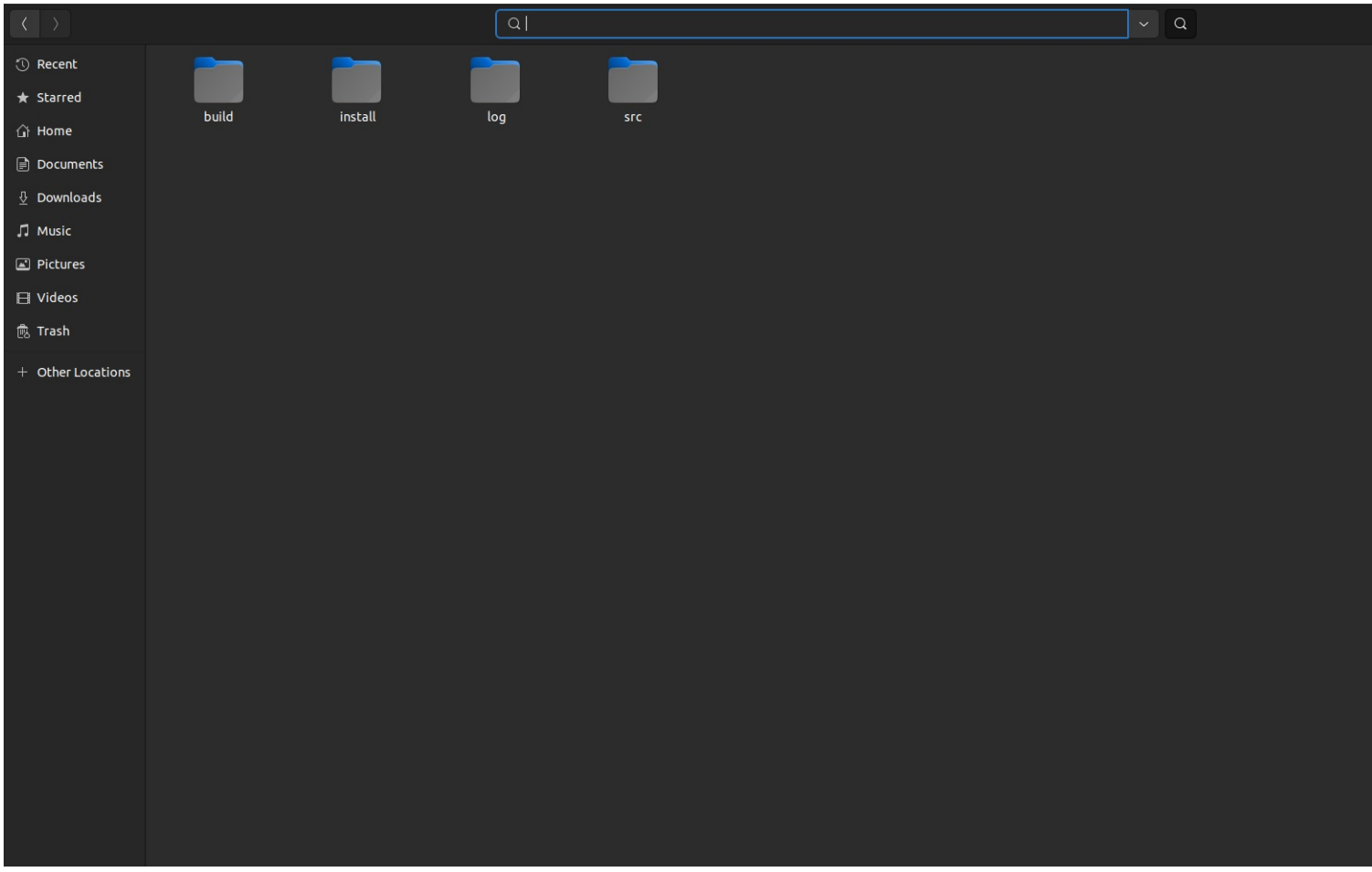
Action Item 1: ROS2 Package creation and Workspace Cleanup – 3 hour(s).

Project Work Summary

- After completing several stages of development and testing within my FarmBot directory, I noticed the workspace had grown cluttered with redundant files, temporary data, and miscellaneous folders. To ensure long-term maintainability and clarity, I decided to perform a structured cleanup and reorganization of the entire workspace.
- I began by reviewing the root farmbot/ directory and manually identifying files and folders that were no longer needed — such as test assets, backup copies, and intermediate files from previous experiments. I retained only the core simulation assets: the Blender .blend file used for generating the terrain mesh, and the finalized .world file used in the Gazebo simulation. All other irrelevant or duplicated resources were deleted to simplify navigation and storage.
- Next, I turned my attention to the install/ directory. Since this folder is automatically generated during builds, I verified that its contents were no longer consistent with my updated file structure. I removed the install/ directory entirely to allow for a clean rebuild from a properly structured workspace.
- With the cleanup complete, I created a new ROS2 package called farmbot_autonomous_pkg. I ensured this package was placed inside the src/ folder of my workspace. During creation, I assigned an appropriate build type (ament_python or ament_cmake, depending on the eventual need) and specified its dependencies according to planned future functionality (e.g., rclpy, std_msgs, gazebo_ros).
- After creating the package structure, I rebuilt the workspace using colcon build from the workspace root. I waited for a successful build and then sourced the generated install/setup.bash file to ensure environment variables were updated for the new package.
- To reflect this change permanently in my shell sessions, I appended the source line to my .bashrc file: source ~/farmbot/install/setup.bash. This allowed the environment to consistently recognize and load the package on terminal startup.
- With the package in place, I manually moved the finalized Blender file and the Gazebo .world file into the appropriate directories within the farmbot_autonomous_pkg folder. The Blender file was placed under a media/ or assets/ directory, while the .world file was moved under a worlds/ subfolder to keep the package modular and clean.
- Finally, I rebuilt the workspace once again using colcon build and sourced the environment to confirm everything was connected correctly. This marked the successful transition from an experimental file pile to a clean, structured, and professional ROS2 package setup — a foundation ready for full autonomous functionality.

- 
Home / farmbot / src
Recent
Starred
Home
Documents
Downloads
Music
Pictures
Videos
Trash
Other Locations
blender
worlds
farmland
wheat
cpr_gazebo
my_farm_sim
sparse_grass_diff_4k.jpg
agriland.dae
sparse_grass_disp_4k.png
agriland.blend1
top-view-artificial-grass-socce...
farmland.dae
farmland.blend
farmland.blend1
fieldnavigator.py
- 
Home / farmbot
Recent
Starred
Home
Documents
Downloads
Music
Pictures
Videos
Trash
Other Locations
build
cpr_gazebo
install
log
src

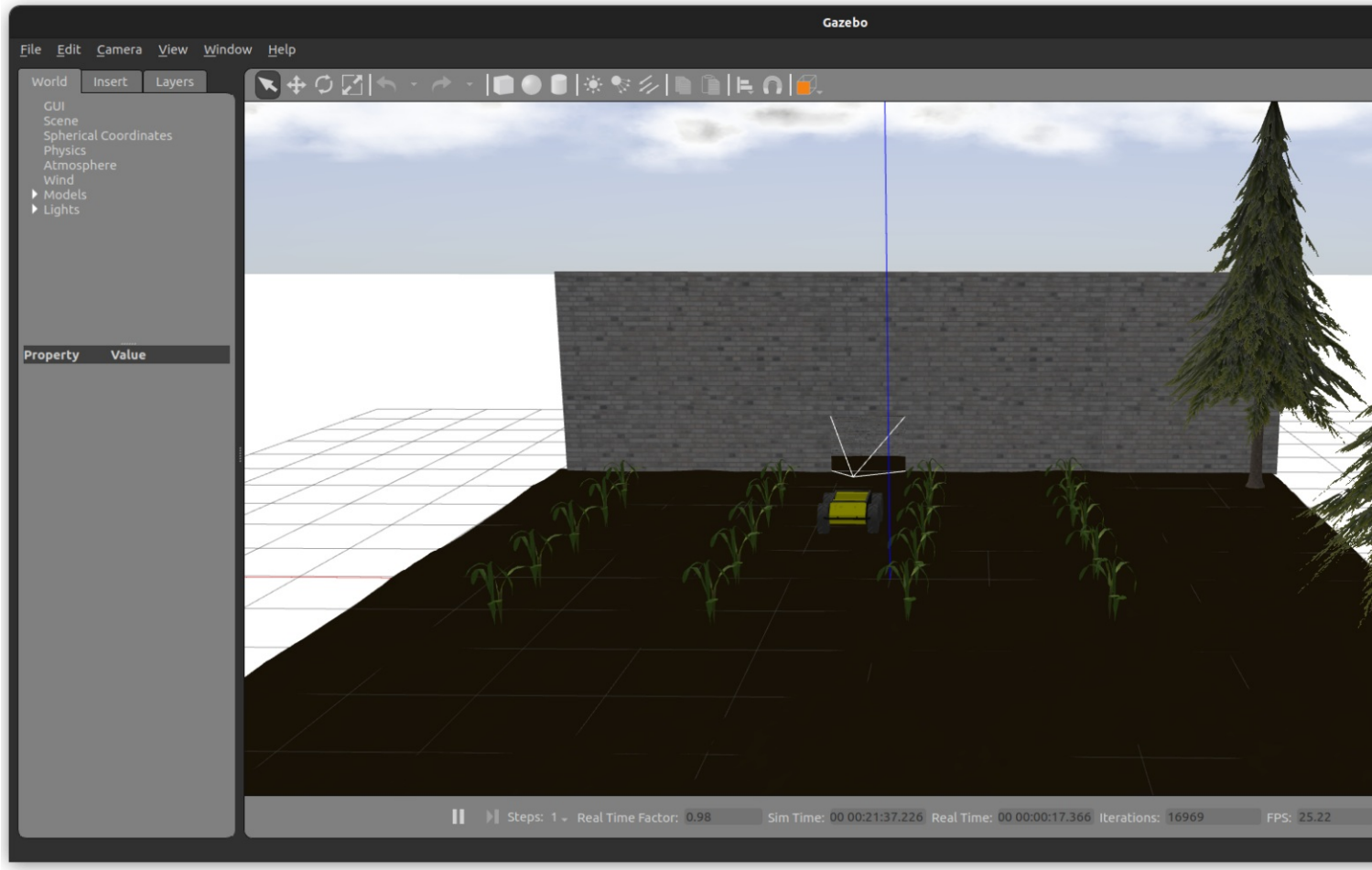


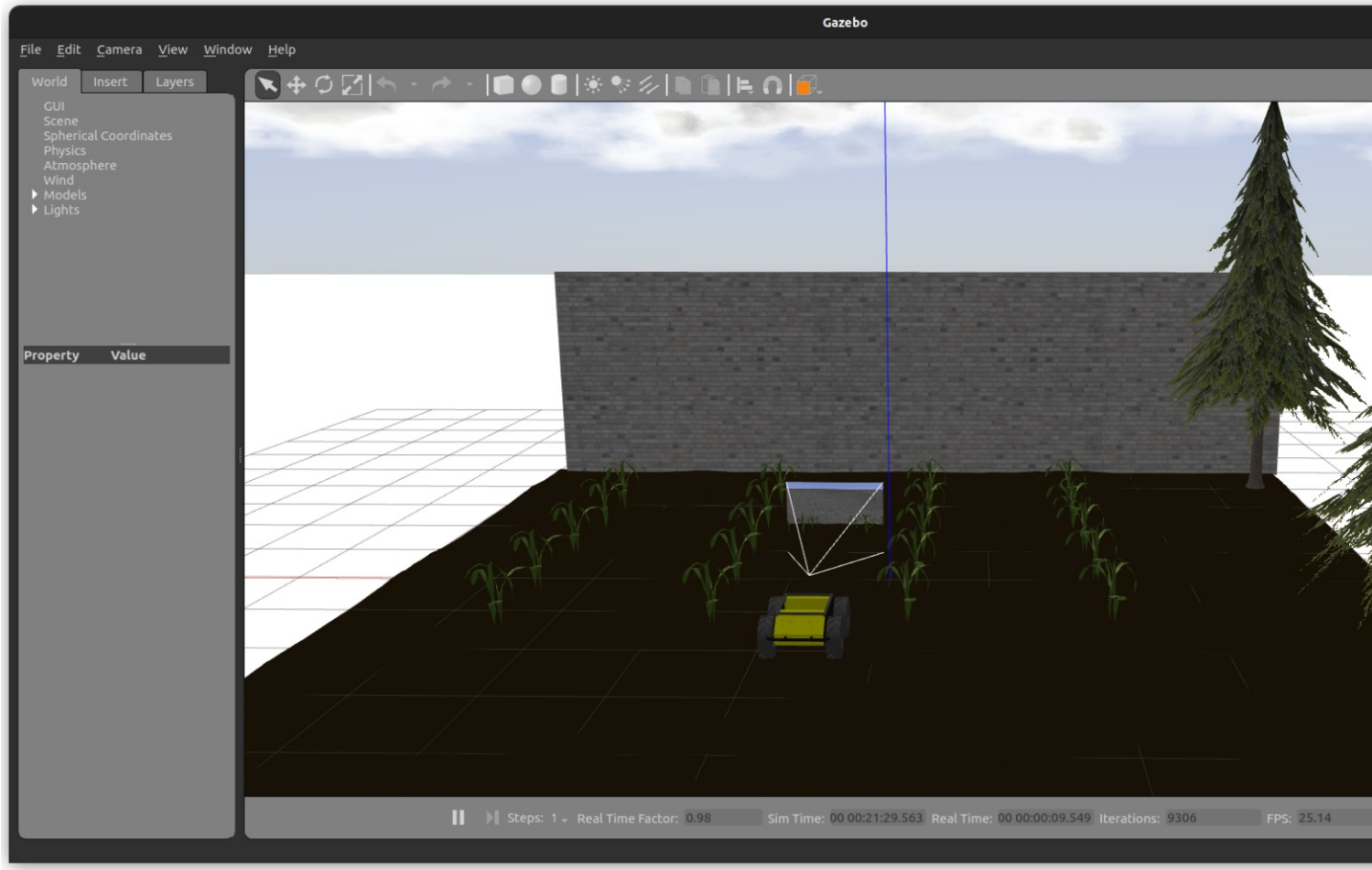


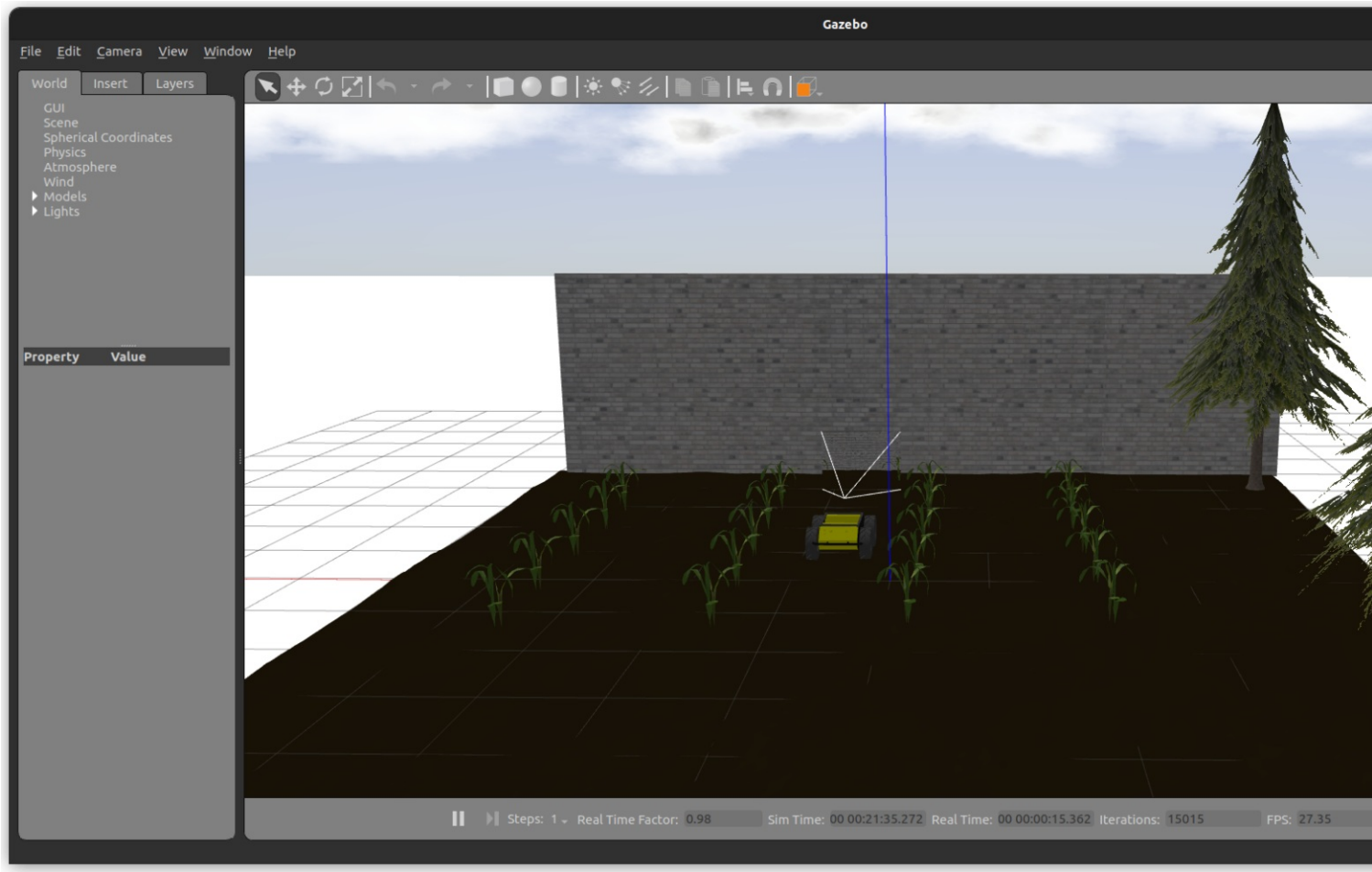
Action Item 2: Debugging Motion Plugin Failures and First successful Teleoperation Control for the Husky Rover – 3 hour(s).

Project Work Summary

- After successfully integrating the Husky robot into the custom farmland Gazebo world, I encountered a major hurdle while attempting to enable robot movement using velocity commands. Initially, the robot remained static, and no `/cmd_vel` topic was present — a clear sign that the differential drive plugin was either missing or not loading properly.
- The initial approach involved using the `libhusky_gazebo_plugins.so` plugin as referenced in Husky's documentation. However, attempts to source or build this plugin for ROS 2 Humble consistently failed. The `husky_simulator` repository did not include a `humble-devel` branch, and attempts to use the `noetic-devel` branch were incompatible due to ROS1-style plugin dependencies and messaging interfaces.
- After confirming that Husky's SDF was correctly formed and that the joints and links were defined properly, I decided to replace the Husky-specific plugin with a more universally compatible ROS 2 plugin: `libgazebo_ros_diff_drive.so`. This plugin is part of the standard `gazebo_ros_pkgs` and works well with ROS 2 Humble when configured correctly.
- My initial attempt to use this plugin failed due to incorrect parameters — I had provided `left_wheel_joints` and `right_wheel_joints`, which are valid in ROS1 but not recognized by the ROS2-compatible plugin. After reviewing plugin documentation and observing errors like `[Missing element description for [left_joint]]`, I modified the configuration to use the ROS1-style `<left_joint>` and `<right_joint>` tags, as the plugin seemed to still interpret the XML format classically.
- Eventually, the plugin loaded successfully when I removed unsupported tags like `<wheel_diameter>` and retained only the expected fields: `left_joint`, `right_joint`, `wheel_separation`, and `wheel_radius`. Once these changes were in place, the plugin loaded without errors, `/cmd_vel` was active, and the robot was ready to receive motion commands.
- To test the setup, I installed and launched the `teleop_twist_keyboard` package. Using the terminal interface, I initiated the teleoperation node and began sending movement commands using the keyboard. The `i` key was particularly effective for forward movement, and with a combination of `w`, `a`, `s`, and `d`, I was able to move and steer the Husky in the Gazebo simulation.
- The robot responded smoothly to velocity commands, and motion was reflected accurately in both the simulation and the topic stream. This successful teleoperation confirmed that the differential drive plugin was functioning correctly and that the robot's motion pipeline was fully operational. This task marked the end of a persistent debugging phase and solidified the foundation for future path-following and autonomous navigation efforts.



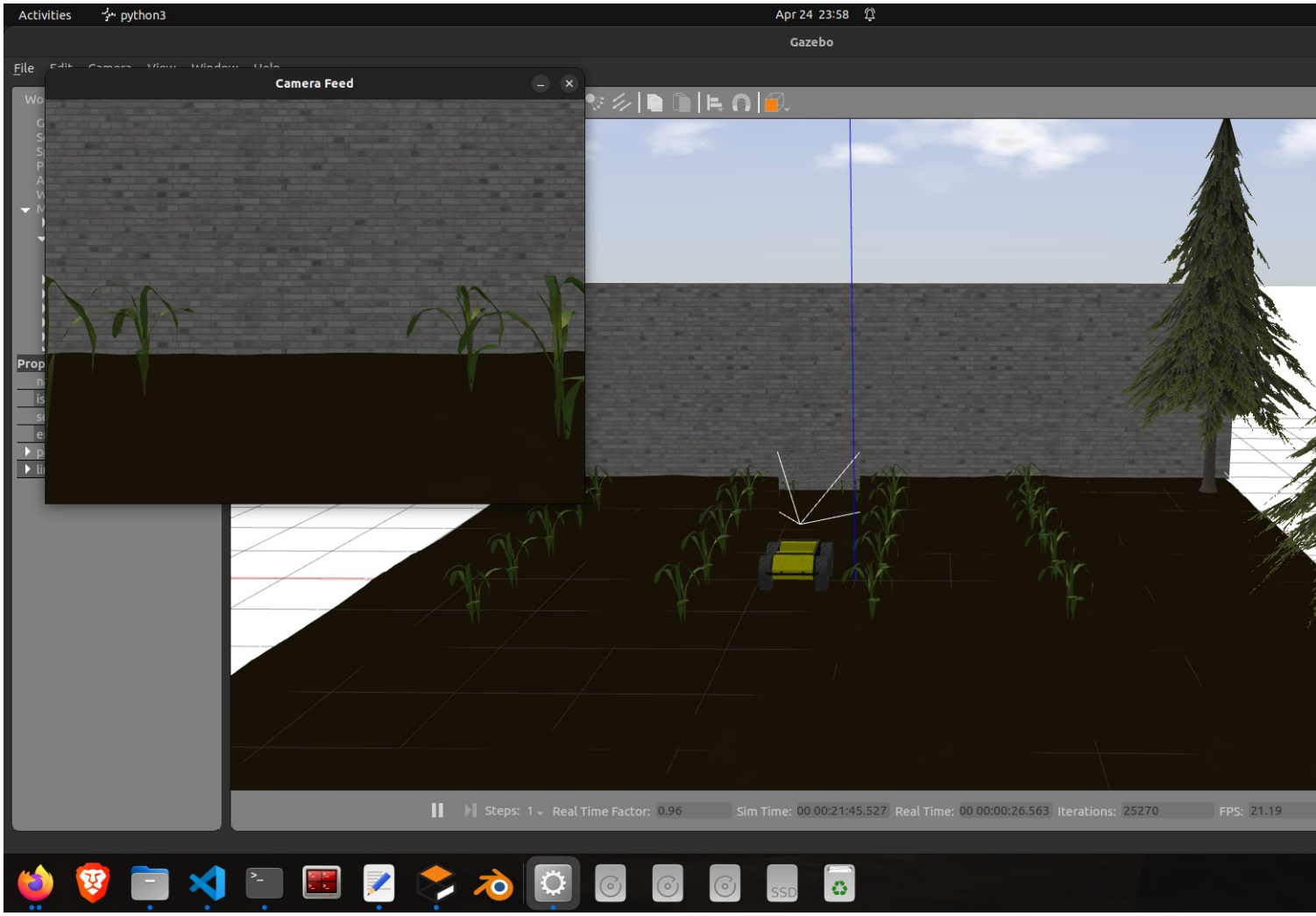


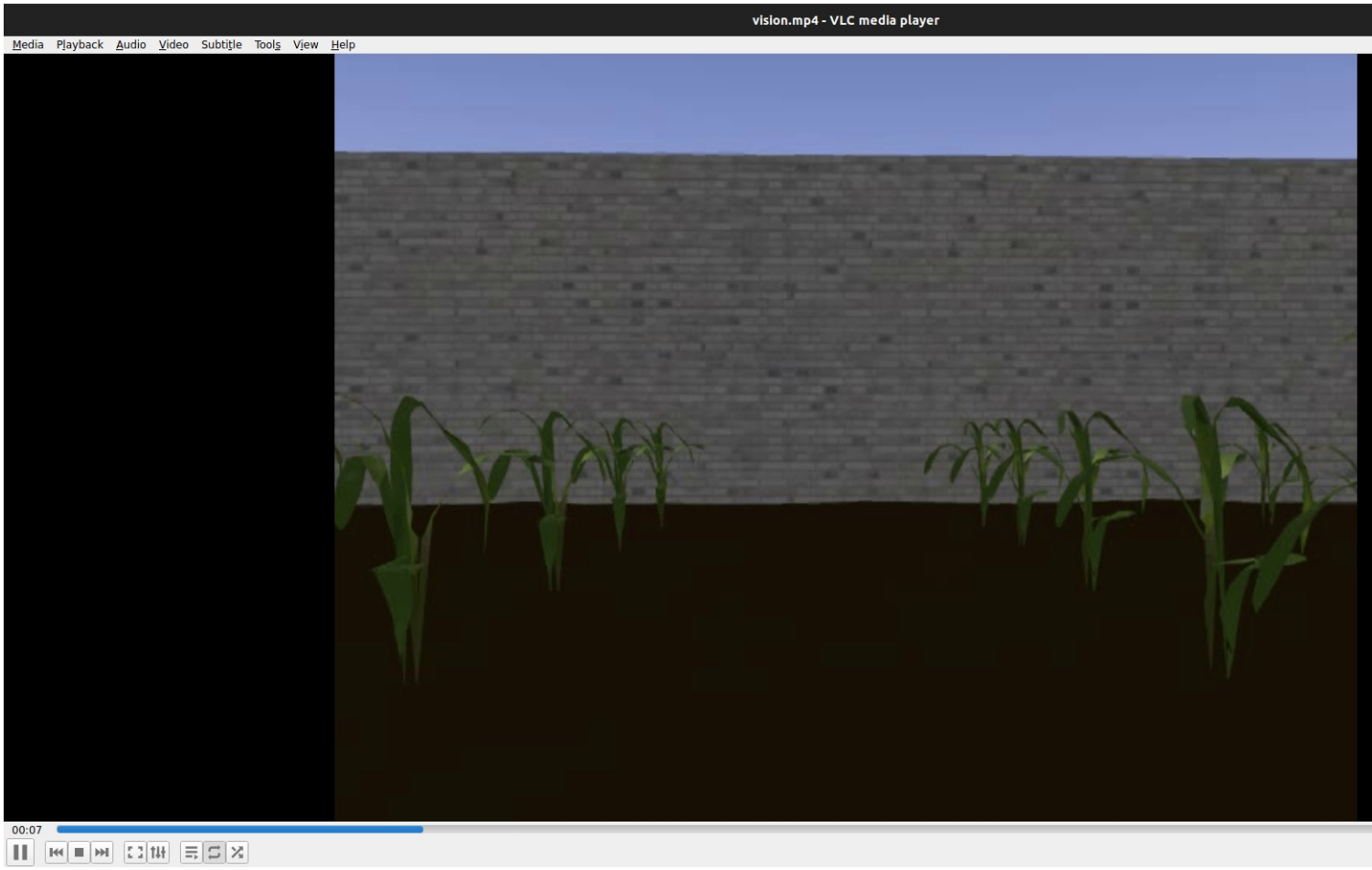


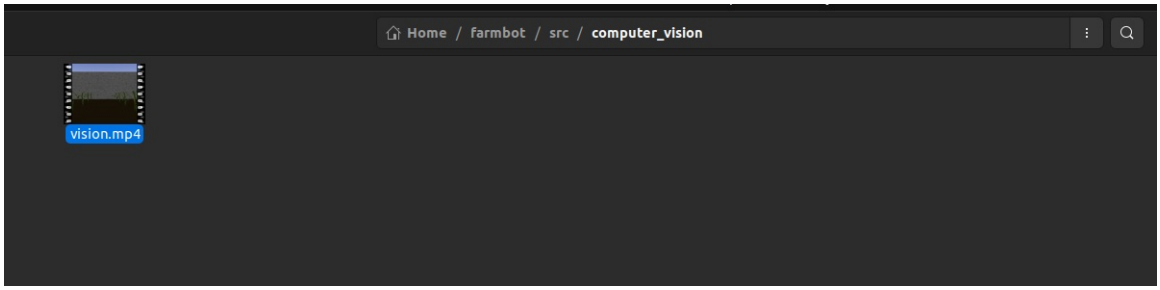
Action Item 3: Developing a Vision Node for Camera Feed Processing and Local Video Recording – 3 hour(s).

Project Work Summary

- Following the successful teleoperation of the Husky rover, the next task focused on leveraging the onboard monocular camera for computer vision use cases. The goal was to subscribe to the camera feed in real time and save the visual data locally as a video file. This would allow for further offline processing, training visual algorithms, or even benchmarking visual odometry pipelines in the future.
- To begin, I created a new ROS 2 Python node named vision_node, designed to subscribe to the /front_camera/image_raw topic. This node utilized the cv_bridge library to convert ROS2 sensor_msgs/Image messages into OpenCV-compatible image frames (bgr8 encoding). This allowed real-time access to raw image data being published by the Gazebo-simulated camera.
- In the constructor of the VisionNode class, I initialized a cv2.VideoWriter instance to handle video recording. The output format was set to MPEG using the 'MJPG' four-character code, and the video was configured to save to a local path under the project directory (vision.mp4). The resolution was matched to the camera's output (640x480), and the frame rate was set to 30 fps.
- The process_data() callback was implemented to run on every image message received. It converted each message to an OpenCV frame, wrote the frame to the video file, and displayed the live feed using cv2.imshow() for visual verification. This allowed me to monitor the camera stream in real-time while also saving the footage.
- The node was integrated with the rest of the system using the existing teleoperation method for motion. As I manually controlled the Husky using teleop_twist_keyboard, the vision node recorded the front camera's view. This produced a clean, time-synchronized video of the robot's field of view during operation, which can now be used as raw input for future VO or computer vision modules.
- The node was finalized with proper ROS 2 lifecycle handling, including shutdown cleanup and camera writer release. With this task complete, I now have a modular and reusable component that enables image capture, live preview, and video storage for any future perception-related task in the project.


Activities
python3
Apr 24 23:58
Gazebo
File
Camera Feed
Steps: 1
Real Time Factor: 0.96
Sim Time: 00 00:21:45.527
Real Time: 00 00:00:26.563
Iterations: 25270
FPS: 21.19

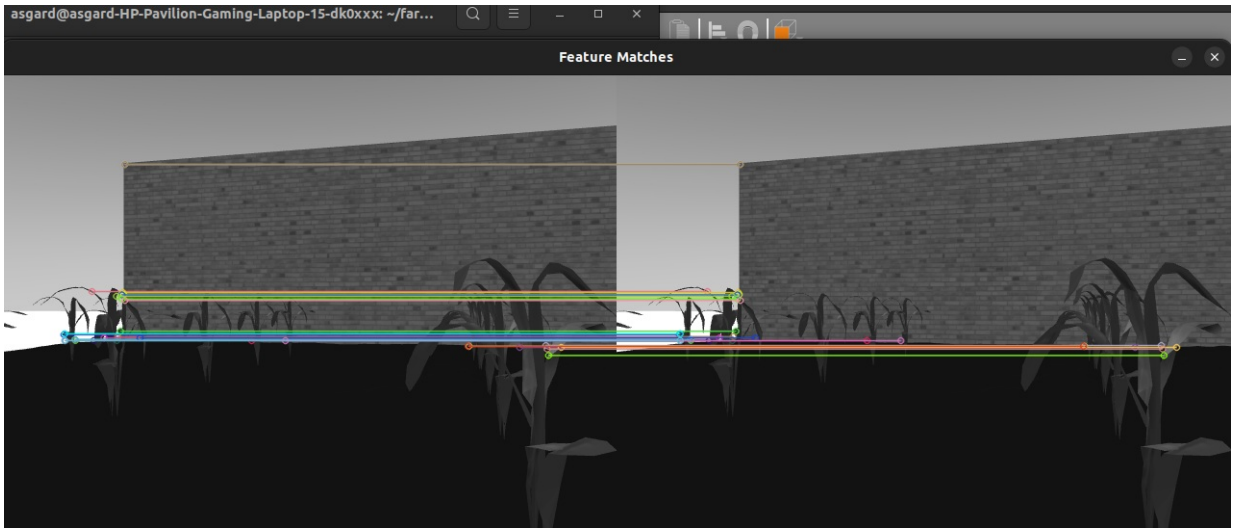


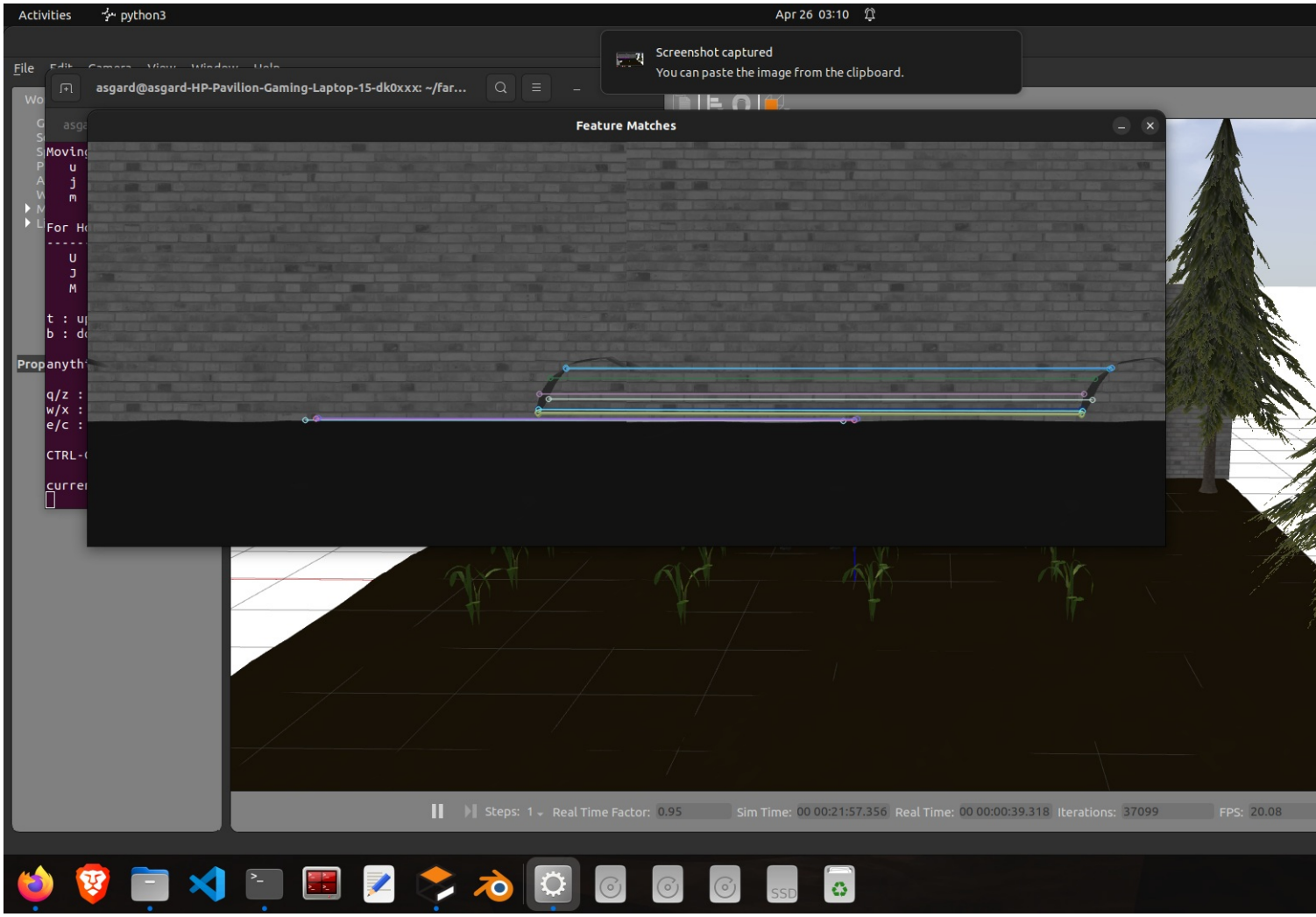


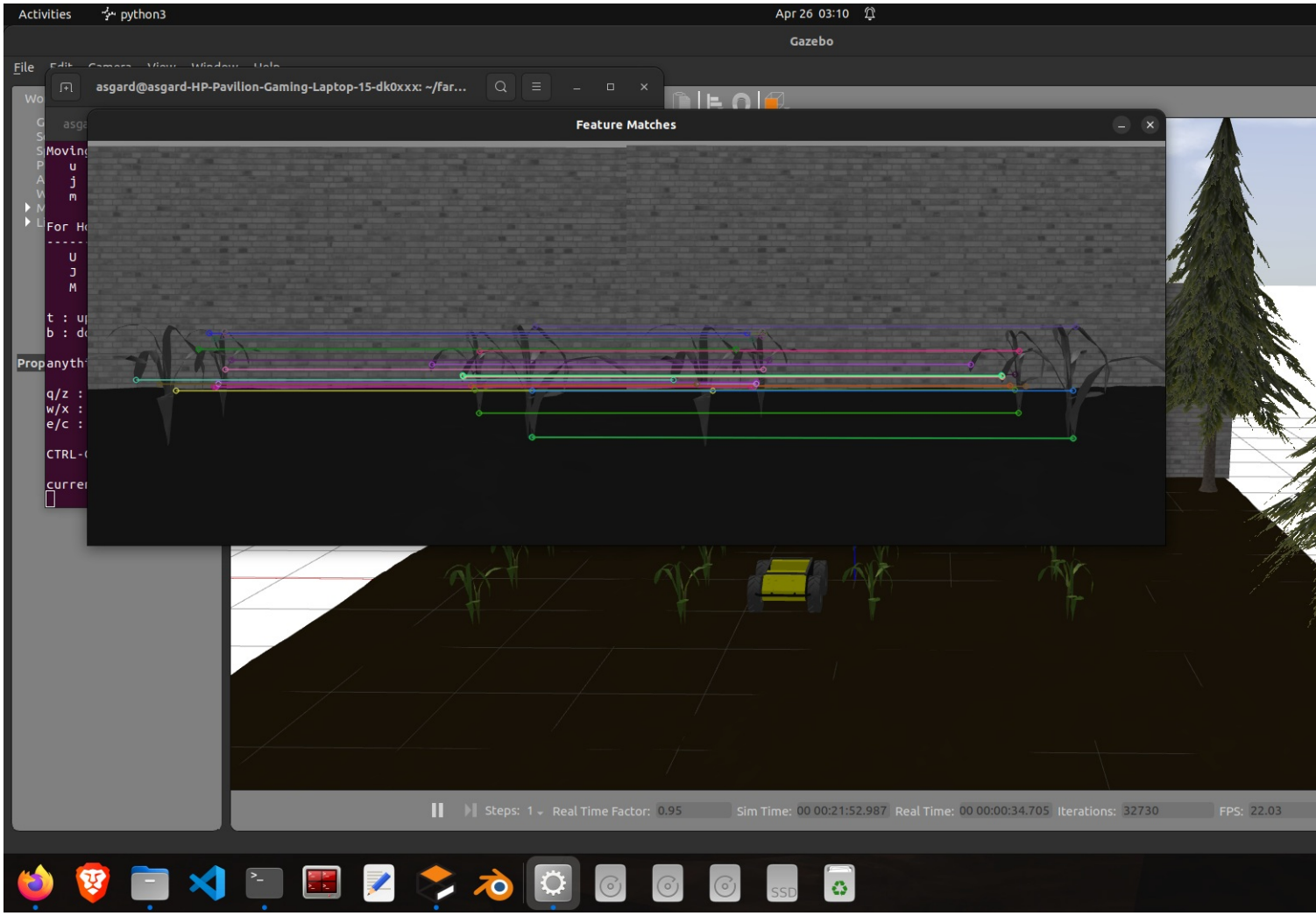
Action Item 4: Integrate and Deploy Visual Odometry Pipeline (Phase 1 – Feature Matching) – 3 hour(s).

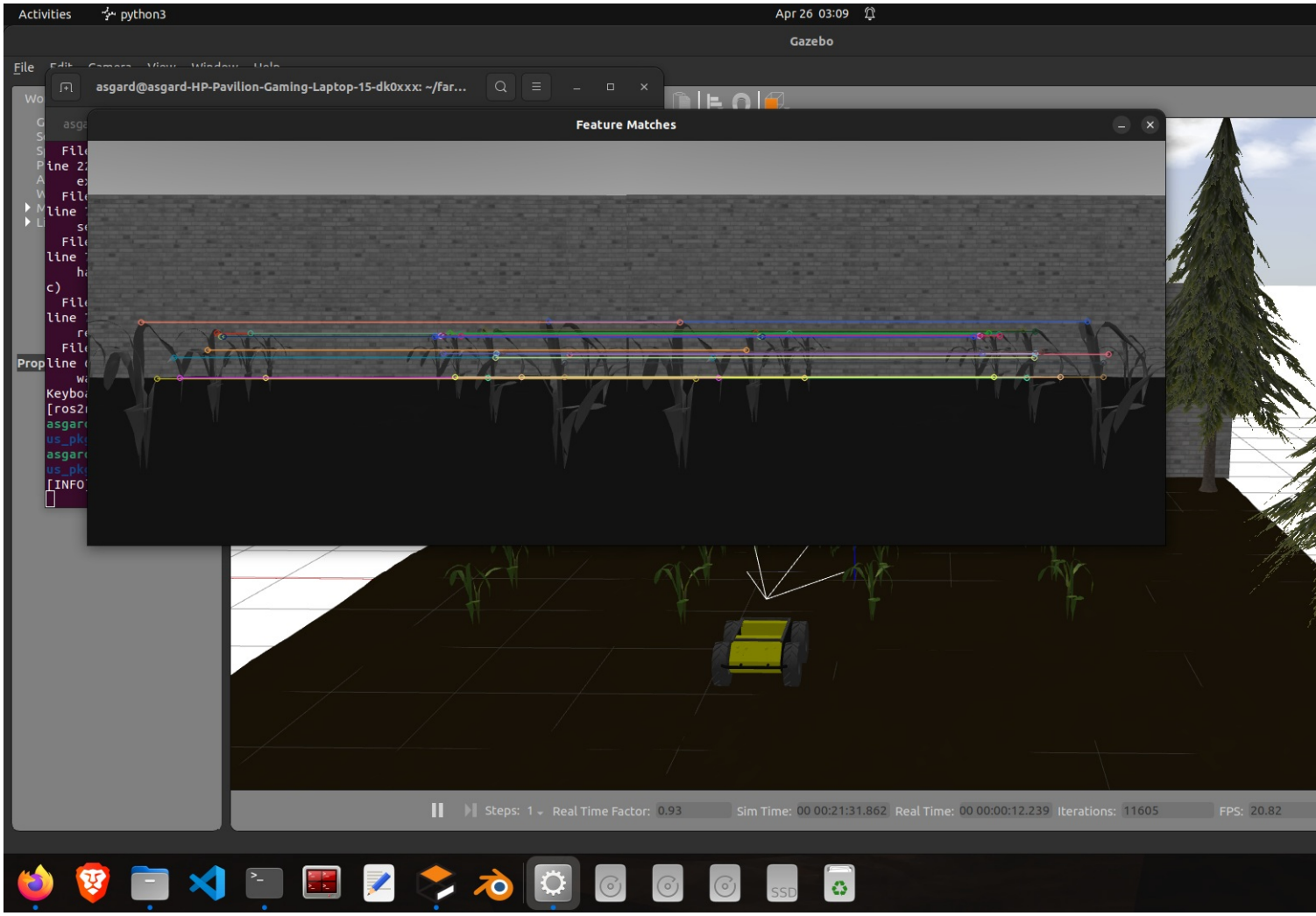
Project Work Summary

- I created a new ROS2 node specifically for visual odometry that subscribes to the monocular camera feed from the Husky robot. This node is built to process real-time image data from the /front_camera/image_raw topic and apply frame-to-frame keypoint matching as a foundation for later motion estimation.
- I implemented ORB-based keypoint detection and description using OpenCV. With every incoming frame, the node identifies key visual features and attempts to find consistent matches between the current and previous images. I used a brute-force matcher with cross-checking to ensure that the matches were reliable and visually trackable.
- To help debug and understand what the robot "sees," I added real-time match visualization using OpenCV's drawMatches. This created a split-screen view of two consecutive frames with colorful lines connecting corresponding features. It was exciting (and slightly surreal) to see the VO pipeline start to come alive visually.
- In parallel, I launched three ROS2 packages: the simulation world (a custom farmland environment with the Husky robot), a teleoperation interface for driving the robot via keyboard commands, and the visual odometry package I just created. This allowed me to control the rover and observe the visual matches update live as the robot moved through the scene.
- I tested the setup by manually navigating the Husky using teleop_twist_keyboard. As I moved the robot, the VO system continuously updated the matched keypoints, especially picking up features on distant walls and corn stalks, which helped confirm that the matching logic was working well even in an outdoor-like simulation.
- The visual odometry system currently estimates only visual feature correspondences and does not yet compute motion. However, it sets the stage for the next step: using these matched points to calculate relative pose via the essential matrix and recover rotation and translation between frames.
- Overall, this step gave me hands-on understanding of how VO starts from raw camera images and builds up toward motion estimation. It also gave me a solid foundation for future integration with IMU and GNSS data for full state estimation.


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


Activities
python3
Apr 26 03:10
Screenshot captured
You can paste the image from the clipboard.
File
asgard@asgard-HP-Pavilion-Gaming-Laptop-15-dk0xxx: ~/far...
Feature Matches
Steps: 1
Real Time Factor: 0.95
Sim Time: 00 00:21:57.356
Real Time: 00 00:00:39.318
Iterations: 37099
FPS: 20.08

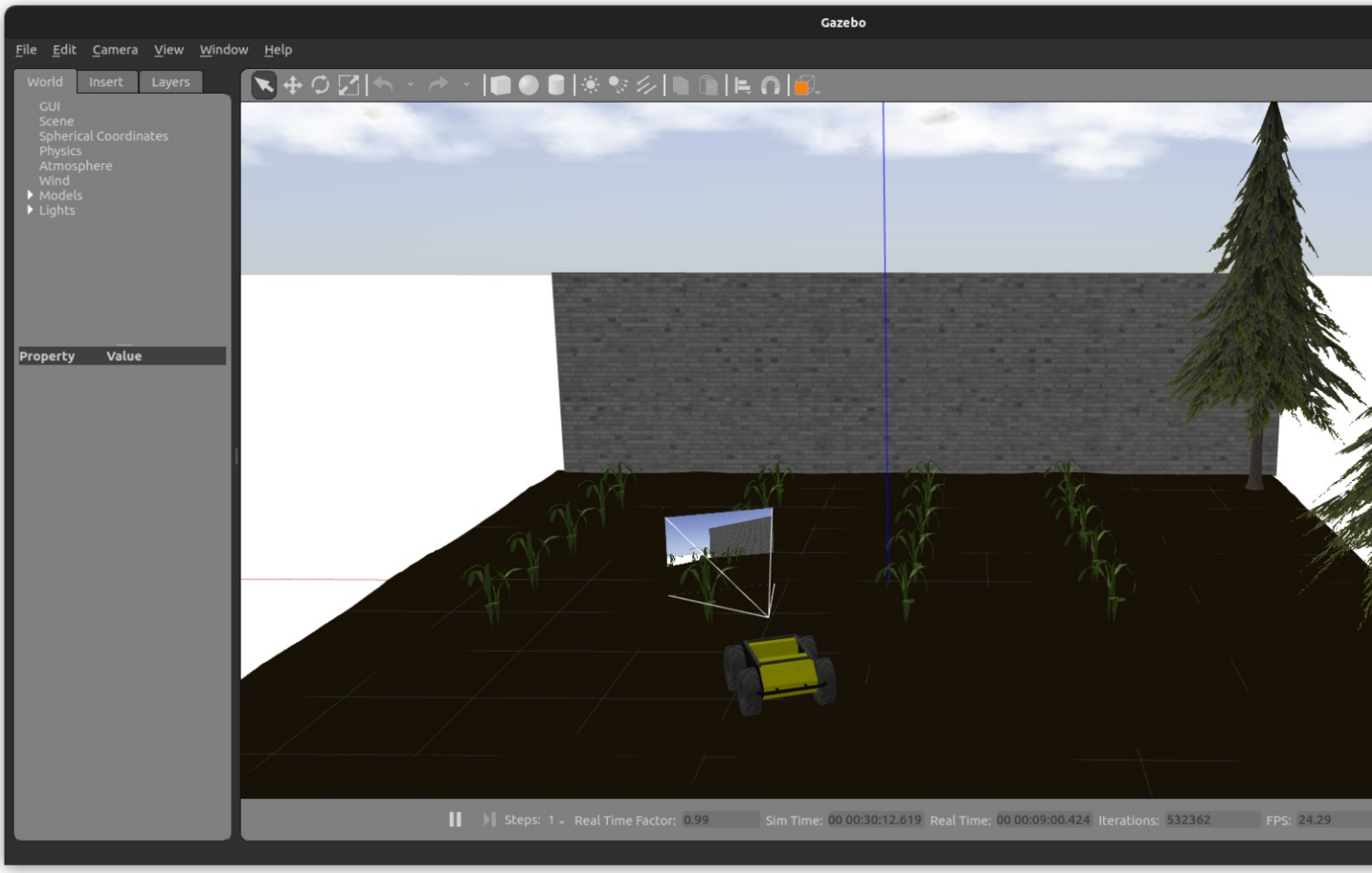

Activities
python3
Apr 26 03:10
Gazebo
File
asgard@asgard-HP-Pavilion-Gaming-Laptop-15-dk0xxx: ~/far...
Feature Matches
Steps: 1
Real Time Factor: 0.95
Sim Time: 00 00:21:52.987
Real Time: 00 00:00:34.705
Iterations: 32730
FPS: 22.03



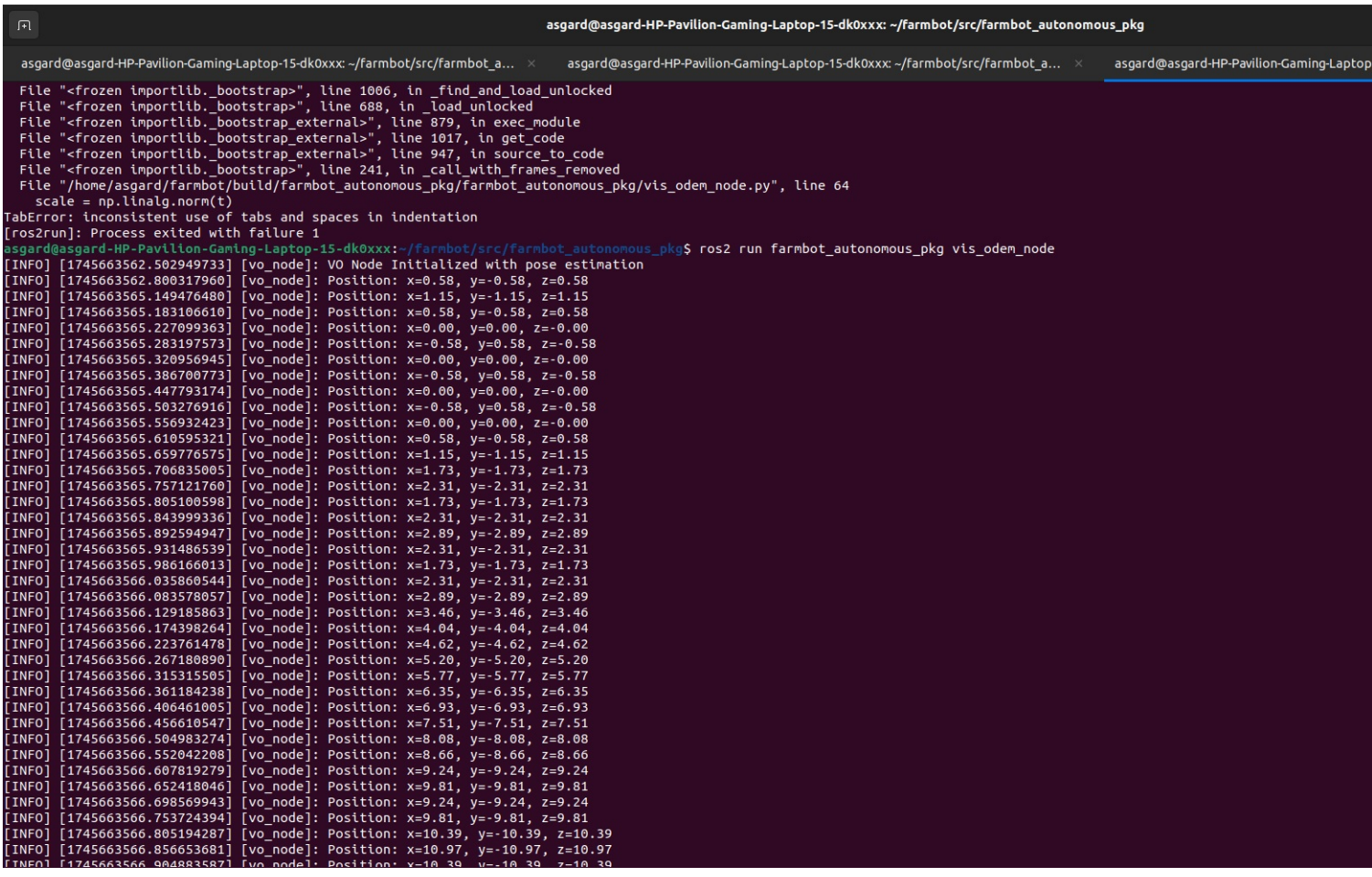
Action Item 5: Pose Estimation from Monocular Visual Odometry – 3 hour(s).

Project Work Summary

- I extended the visual odometry pipeline by implementing relative pose estimation between consecutive camera frames using the Essential Matrix. After matching keypoints with ORB, I used OpenCV's `findEssentialMat` and `recoverPose` functions to extract the robot's rotation and translation between frames. This allowed me to accumulate motion over time and estimate the trajectory of the robot in 3D space.
- To support this, I added a simple camera intrinsic matrix based on the known resolution and field of view of the simulated camera. I composed the estimated rotation and translation into a 4x4 transformation matrix and chained it with the previous pose, updating the cumulative position at each step.
- While testing this setup in the Gazebo farmland world, I observed that the pose estimates were updating even when the robot was completely stationary. This indicated that the system was interpreting minor keypoint shifts—likely due to noise or lighting jitter—as real motion, resulting in incorrect translations being applied frame after frame.
- To address this, I introduced two sanity checks to filter out unreliable pose updates. First, I computed the magnitude of the translation vector and skipped pose updates when the scale was below a small threshold.
- This prevented the system from drifting due to visual noise when the robot wasn't moving. Second, I logged the number of inliers used by the pose recovery step and added a check to ensure enough valid matches were present before trusting the result.
- With these fixes in place, the system now avoids accumulating false motion and behaves more realistically when the robot is idle, setting a cleaner foundation for trajectory estimation and future sensor fusion.



-



-

-

Action Item 6: ORB-SLAM: A Versatile and Accurate Monocular SLAM System – 3 hour(s).

Research

- https://arxiv.org/abs/1502.00956
- ORB-SLAM: A Versatile and Accurate Monocular SLAM System
- Summary of Report
  - This paper presents ORB-SLAM, a real-time monocular SLAM system capable of tracking, mapping, loop closing, and relocalization, all using the same ORB features. What makes it stand out is how every component—from initialization to optimization—relies on a tightly integrated feature pipeline, allowing efficient and robust SLAM even in large or dynamic environments.
  - The authors designed ORB-SLAM with a multi-threaded architecture, where tracking, local mapping, and loop closing run in parallel. This allows the system to continuously build and refine a sparse 3D map while staying real-time. They use an “essential graph” for global pose graph optimization and loop closure, which corrects long-term drift efficiently.
  - Extensive benchmarking is done on indoor (TUM RGB-D), outdoor (KITTI), and campus-scale (NewCollege) datasets. In many cases, ORB-SLAM outperforms both direct methods like LSD-SLAM and older keyframe-based systems like PTAM in terms of accuracy, scalability, and reliability.
- Relation to Project
  - This paper strongly influenced my implementation of visual odometry in ROS2, especially the use of ORB feature matching, Essential Matrix estimation, and pose composition. It gave me confidence that my current ORB + cv2.recoverPose() pipeline was on the right track—even if just a minimal VO version of their full SLAM.
  - The concept of culling weak matches and filtering motion based on inlier count or translation magnitude directly inspired the sanity checks I added to my VO pipeline to avoid fake motion when the robot is stationary.
  - Their emphasis on using a single type of feature (ORB) for all SLAM components validated my decision to stick to ORB for both simplicity and speed. Their findings about ORB's robustness to illumination and viewpoint changes also supported my use case in a farmland simulation, which has open lighting and repetitive textures.
- Motivation for Research
  - I wanted to understand what separates a basic visual odometry pipeline from a full, reliable SLAM system, and this paper gave me a roadmap—from initialization heuristics to global optimization.
  - The tracking robustness and reinitialization strategies described here (especially using bag-of-words with ORB descriptors) motivated me to think beyond just VO and consider scalable, long-term localization.
  - ORB-SLAM’s experiments on KITTI and TUM datasets helped me set realistic expectations for monocular systems and motivated me to start looking into GNSS + VO fusion and IMU integration for scale correction and drift minimization.

Action Item 7: Weekly Task Plan: Advancing VO Pipeline Toward Sensor Fusion – 1 hour(s).

Project Work Summary

- Refactor the current VO node to include a cleaner structure for debugging and selectively logging information such as frame-to-frame scale, inlier count, and cumulative drift. This will help me better understand when the pose estimates are reliable and when they start to degrade.
- Add a simple trajectory visualization tool, either using matplotlib for offline plotting or publishing to a ROS2 topic like /vo/pose or /vo/odom so that I can view the path in RViz2 while the simulation runs.
- Start integrating GNSS data into the system by simulating a GPS sensor using Gazebo's gps_plugin, ensuring that it publishes sensor_msgs/NavSatFix messages with some configurable noise. The goal is to have a parallel stream of global coordinates.
- Build a GNSS-to-UTM or ENU coordinate transformer node so that I can convert NavSatFix into a Cartesian frame that can be aligned with the VO estimate. This will be essential for comparing or fusing GNSS and VO data in a shared frame.
- Write a basic scale alignment node that compares VO translation magnitudes with GNSS-derived position differences over time. This node will calculate an average scale factor to rescale the VO trajectory for rough global consistency.
- Begin preliminary work on fusing GNSS and VO using an Extended Kalman Filter. The goal for this week is just to set up the robot_localization EKF node, feed in VO and GNSS (as Odometry or NavSatFix + tf), and observe the behavior.

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.