

Contents

1	SIMLAN, Simulation for Multi-Camera Robotics (3.0.0)	4
1.1	SIMLAN Features [SIMLAN demo]	4
1.2	Installation [Demo]	5
1.2.1	Dependencies	5
1.2.2	Quick Start	6
1.3	GPSS controls (pallet trucks, aruco) [Demo]	6
1.4	RITA controls (humanoid, robotic arm) [Demo]	7
1.5	Advanced options	7
1.5.1	Customized startup	7
1.5.2	World fidelity	7
1.5.3	Filtering log output	7
1.5.4	Older versions	8
1.6	Documentation	8
1.7	Research Funding	8
1.7.1	3.1.0: Improvements (Dec 2025) - Supervisor: Hamid Ebadi	9
1.7.2	3.0.0: Jazzy, humanoid (Oct 2025) - Supervisor: Hamid Ebadi	9
1.7.3	2.1.0: Namespace cleanup and multi-agent navigation (Aug 2025) - Supervisor: Hamid Ebadi	9
1.7.4	2.0.0: GPSS (May 2025) - Supervisor: Hamid Ebadi	9
1.7.5	1.0.6: Collision (Feb 2025)	9
1.7.6	1.0.5: Delivery 4 (Dec 2024)	9
1.7.7	1.0.4: Delivery 3 (Oct 2024)	9
1.7.8	1.0.3: Jackal Robot (Mar 2024) - Christoffer Johannesson	10
1.7.9	1.0.2: Delivery 2 (Feb 2024)	10
1.7.10	1.0.1: Delivery 1 (Dec 2023) - Supervisor: Hamid Ebadi	10
2	Linux Dependencies	10
2.1	nvidia Driver	10
2.2	Docker	10
2.3	nvidia-container-toolkit	10
3	Windows Dependencies	11
3.1	Legal Notice	12
3.2	Contributing	12
3.3	Technical	12
3.3.1	git	12
3.3.2	Misc	12
3.3.3	Documentation	13
3.4	automatically generated parameter files:	13
3.5	World Fidelity	13
3.6	Adding Aruco and camera (static agents)	13
3.7	Using actors in Gazebo	14
3.8	Simulation Specification	15
3.9	AMR	15
3.10	Warehouse	15
3.11	Add warehouse to world-file	16
3.12	Textures	16
3.12.1	Conventions	16
4	Converting FreeCAD meshes to SDF models (for Gazebo)	16
4.1	Script step-by-step	16
4.2	In GUI	17
4.3	Blueprint	19
4.4	AMR	20
4.4.1	Infobot AMR specification	20
4.5	Objects Modeled in FreeCAD	21
4.6	AMR camera	21
4.7	Resources	21
4.8	Blueprint	21
4.9	Install instruction for BIM and Arch workbench	22

4.10	Add warehouse to world-file	22
4.11	Resources	23
5	forklift_robot	23
5.1	URDF	23
5.1.1	contact/collision parameters	23
5.1.2	objects:	23
5.1.3	reference:	23
6	Static_agent_launcher	23
6.0.1	gz_bridge for cameras	24
7	bird-eye-view package	24
8	SIMLAN bringup	25
8.1	Launch arguments	25
8.2	Diagram of launch structure	26
9	pallet_truck	26
10	SIMLAN project	26
10.1	Pallet Truck bringup.	26
10.2	Configuring and spawning robots inside the sim.	26
10.3	automatically generated parameter files:	26
10.4	Pallet_truck_control	27
11	pallet_truck Description	27
11.1	Sensors	27
11.2	documentation on pallet_truck_description	27
11.3	Additions:	27
11.3.1	Collision sensor	27
11.4	Old Bugs	27
11.4.1	xacro/urdf/sdf files	27
11.4.2	Map server	28
11.4.3	Robot_agent navigation	28
11.4.4	Humanoid_navigation	29
11.5	Obstacles detection (update_map_node)	29
11.5.1		30
12	human-gazebo	30
12.1	Human subject with meshes	31
12.2	Maintainers	32
13	aruco_localization package	32
13.1	Important points to pay attention to:	32
13.2	Important launch files	32
13.3	Odom	32
14	FAILSAFE for the navigation of pallet trucks	32
14.0.1	Activation of collision sensor	33
14.0.2	Loss of Observability	33
14.1	Behavior tree and direct implementation in aruco_localization pkg	33
14.1.1	Why Nav2 behavior tree plugins was not suitable	33
15	Humanoid	33
15.0.1	Moveit	33
15.0.2	Dynamically updated urdfs	34
15.0.3	Planning scene monitor	34
15.0.4	Figures	34
15.1	Rviz2 visualization	34
15.2	Bugs	34
15.2.1	TF visualization and gazebo simulation not matching	34

15.3	Preparing Data	35
15.3.1	First build the package	35
15.3.2	Add the data after building	35
15.3.3	Important Notes	35
15.4	Config file	35
15.4.1	<code>params.yaml</code> Overview	36
15.4.2	Launching the Processing Step	37
15.5	Sending Data	37
15.5.1	Adjusting Playback Speed	37
15.5.2	Fixing RViz	37
15.6	Summary	37
15.6.1	Warnings and Errors from the node	38
16	Scenario Manager	38
16.1	Overview	38
16.2	Launching the Scenario Manager	38
16.3	Collision Action Server	38
16.3.1	Collision Types	38
16.4	Running a Scenario	38
16.5	Time to Collision (TTC) Calculation	38
16.5.1	Running the TTC Node	39
16.6	Moveit Panda robot	39
16.7	Moveit motion planner using Python API	39
17	MoveIt Resources	39
17.1	Included Robots	39
17.2	Notes	39
18	panda_description	39
18.1	MoveIt Resources for testing: Franka Emika Panda	40
18.2	Random, In Distribution Objects Movement (normal mode)	40
18.3	Random, Out of Distribution Objects Movement (abnormal mode)	40
18.3.1	Deterministic, Disentanglement One-parameter Object Movements (one_object_deterministic mode)	40
19	Humanoid Motion Capture	40
19.1	Terminology:	41
19.2	Dataset	41
19.2.1	Translation of a detected pose to humanoid motion command	41
19.2.2	Neural network design	41
19.3	Project Structure:	41
19.4	Dataset generation	42
19.5	Dataset preprocessing	42
19.6	Model Training	43
19.6.1	Autogluon	43
19.6.2	Pytorch	43
19.7	Optuna	43
19.7.1	Commands	43
19.8	Model Evaluation	44
19.9	Model Prediction	44
19.10	Misc. (IGNORE WHAT COMES NEXT)	44
19.10.1	Notes	44
20	Preprocessing	45
20.1	Issues and Future Works	46
20.1.1	Current Performance Metrics	46
20.1.2	Known Limitations	46
20.1.3	Future Development Areas	46
20.2	The Theory	46
21	Autogluon Model configuration	46

22 Hyperparameter Tuning	47
22.1 gazebo_ros2_control	47
22.2 OpenAL	47
22.3 Command failed: docker compose	48
22.4 Missing nvidia docker runtime	49
22.5 Docker nvidia-container-cli: requirement error	49
22.6 GLIBC_2 issue	49
22.7 docker issues	49
22.8 Advanced options:	49
22.9 Advanced features	49
22.9.1 Running ROS2 commands:	49
22.10 Jackal	50
22.10.1 high-level_diagram_SIMLAN.drawio	50
22.11 Licenses and Credits	50

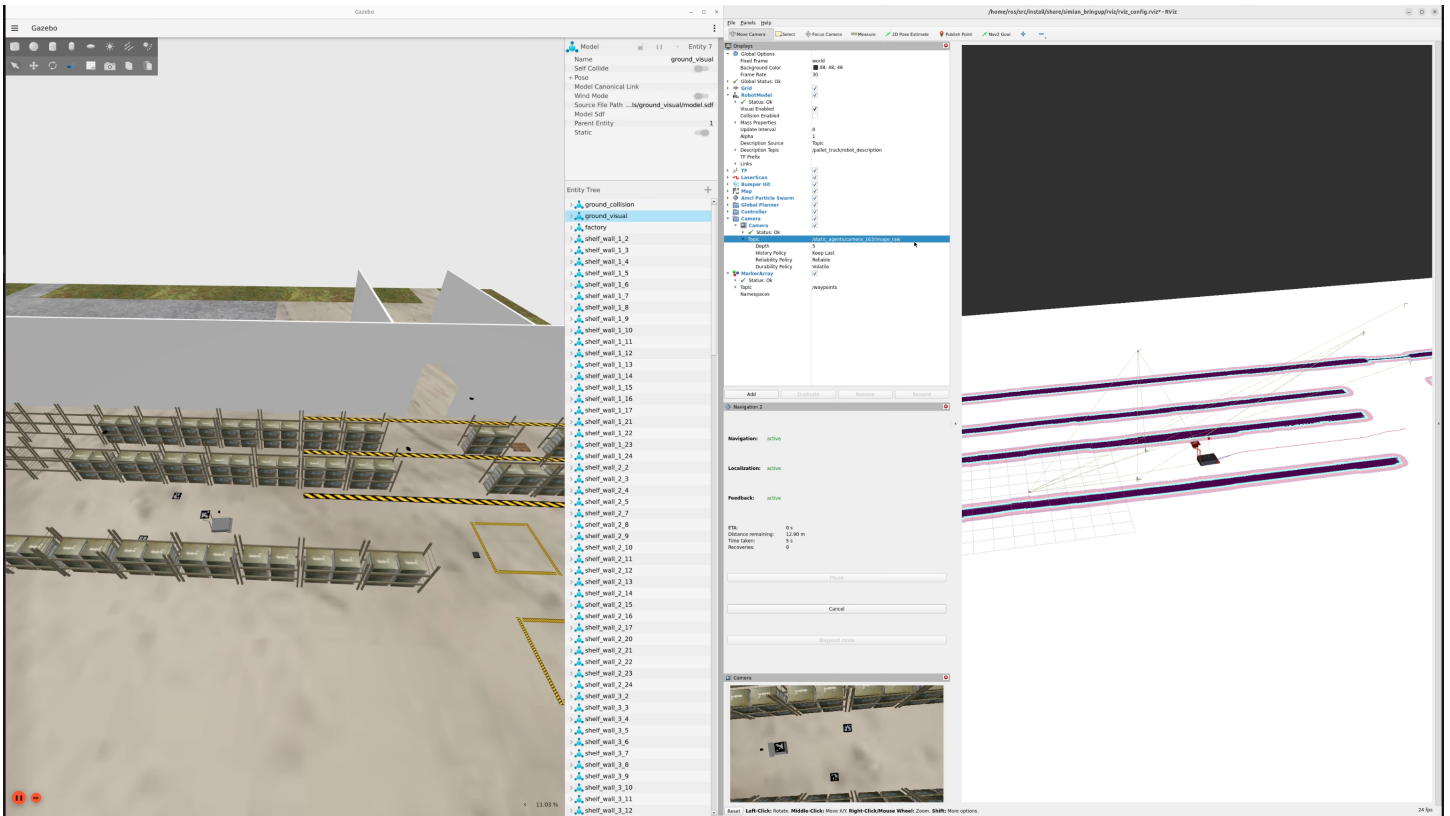
1 SIMLAN, Simulation for Multi-Camera Robotics (3.0.0)

This simulation environment, based on the Gazebo Ignition simulator and ROS2, resembles a Volvo trucks warehouse and serves as a playground for rapid prototyping and testing of systems that rely on multi-camera setup for perception, monitoring, localization or even navigation. This project is inspired by [GPSS \(Generic photo-based sensor system\)](#) that utilizes ceiling mounted cameras, deep learning and computer vision algorithms, and very simple transport robots. [[GPSS demo](#)]

1.1 SIMLAN Features [[SIMLAN demo](#)]

- Ignition Gazebo
- Library of assets
- Real-World environment inspired design (camera position and warehouse layout)
- ROS 2 interfaces (Humble and Jazzy)
- ArUco marker localization
- Simple GPSS (Generic Photo-based Sensor System) navigation
- Multi-Robot localization and navigation using Nav2
- Bird's-Eye view projection
- Multi-Sensor Support (LiDAR, RGB camera, semantic segmentation, Depth etc.)
- Geofencing for safe zones and safestop on collision
- Motion capture for Human-Robot Collaboration/Interaction (HRC/HRI)

Please click the youTube link below to view the SIMLAN demo video:



Here are list of advantages of using SIMLAN Multi-Camera system

- Rapid prototyping and iteration of ML based algorithm. (e.g. reinforcement learning)
- Enhanced monitoring and coordination using bird eye view
- Simplified robot design and maintenance.
- Extendible with additional ML based vision systems
- Safety testing without physical risk or privacy concerns
- Scalable and reproducible testing (CI/CD)
- Cost-effective development

1.2 Installation [[Demo](#)]

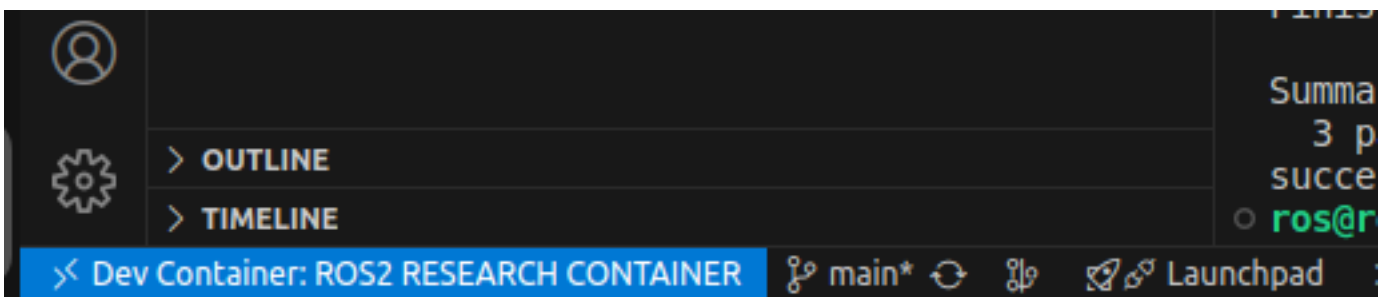
1.2.1 Dependencies

Ubuntu 24.04: use instruction in [dependencies.md#linux-dependencies](#) to install docker and ensure that your linux user account has **docker** access. *Attention:* Make sure to restart the computer (for the changes in group membership to take effect.) before proceeding to the next step.

Windows 11: use instruction in [dependencies.md#windows-dependencies](#) to install dependencies.

Production environment: follow installation procedure used in [.devcontainer/Dockerfile](#) to install dependencies.

Development environment: to improve collaboration we use vscode and docker as explained in [this instruction](#) and [docker files](#). Install Visual Studio Code (VS Code) and open the project folder. VS Code will prompt you to install the required extension dependencies. Make sure the **Dev containers** extension is installed. Reopen the project in VS Code, and you will be prompted to rebuild the container. Accept the prompt, this process may take a few minutes. Once VS Code is connected to Docker (as shown in the image below), open the terminal and run the following commands:



(if you don't see this try to build manually in vscode by pressing **Ctrl + Shift + P** and select **Dev containers: Rebuild and Reopen in container.**)

1.2.2 Quick Start

The best place to learn about the various features, start different components, and understand the project structure is [./control.sh](#).

Attention: The following commands (using [./control.sh](#)) are executed in a separate terminal tab inside *vscode*.

To kill all the relevant process (related to gazebo, ros2), delete build files, delete recorded images and rosbag files using the following command:

```
./control.sh clean
```

To clean up and build the ros2 simulation

```
./control.sh build
```

(optionally, in vscode you can click on Terminal-> Run Task/Run build Task or use **Ctrl + Shift + B**)

1.3 GPSS controls (pallet trucks, aruco) [[Demo](#)]

It is possible for the cameras to detect ArUco markers on the floor and publish their location to TF, both relative to the camera, and the ArUcos transform from origin. The package [./camera_utility/aruco_localization](#) contain the code for handling ArUco detection.

You can also use nav2 to make a robot_agent (that can be either robot/pallet_truck) navigate by itself to a goal position. You can find the code in [simulation/pallet_truck/pallet_truck_navigation](#)

Run these three in separate terminals

```
./control.sh gpss # spawn the simulation, robot_agents and GPSS ArUco detection
./control.sh nav  # spawn map server, and separate nav2 stack in a separate namespace for each robot_agent
./control.sh send_goal # send navigation goals to nav2 stack for each robot_agent
```

If you want to control any robot (pallet truck, humanoid, etc) manually you can run the following command. Remember to specify what robot you want to control by adding its namespace as argument, i.e. [./control.sh teleop pallet_truck_1](#)

```
./control.sh teleop ${YOUR_ROBOT_NAMESPACE}
```

If you want to record any of your topics during the tests you can run the following command. Change the topic in the control.sh script: **ros2 bag record /topic** to whatever topic you want to record.

```
./control.sh ros_record
```

To replay your latest recorded rosbag run the following command:

```
./control.sh ros_replay
```

If you want to do a camera dump and save the image from each camera as a .png run the following command. The images will appear at [/src/camera_utility/camera_number](#).

```
./control.sh camera_dump
```

If you want to take a screenshot of one of the cameras view, run the following command. Replace **###** with the camera you want to take a screenshot of. (163, 164, 165 or 166)

```
./control.sh screenshot ###
```

```
./control.sh birdeye
```

If you want to add the tf links between the cameras and the ArUco markers without running the **gpss** command you can run the following command. This is not that usable as the **gpss** run this as well, but it can be good for debugging.

```
./control.sh aruco_detection
```

Finally, to view the bird's-eye perspective from each camera, run the following command and open **rviz** Then, navigate to the scroll menu to the left, and under "Camera" change the Topic [/static_agents/camera_XXX/image_projected](#) topic to visualize the corresponding camera feed:

1.4 RITA controls (humanoid, robotic arm) [[Demo](#)]

```
./control.sh humanoid
```

To move humanoid around in the simulator

```
./control.sh teleop ${YOUR_HUMANOID_NAMESPACE}
```

1.4.0.1 Arm controls Spawn the Panda arm inside SIMLAN and instruct it to pick and place a box around with the following commands:

```
./control panda
./control plan_motion
./control pick
```

1.5 Advanced options

See [resources/ISSUES.md](#) to learn about additional advanced options and to check known issues before reporting any issue or requesting new features. To start the project **without NVIDIA GPU** please comment out these lines in `docker-compose.yaml` as shown below:

```
# runtime: nvidia
#
# factory_simulation_nvidia:
# <<: *research-base
# container_name: factory_simulation_nvidia
# runtime: nvidia
# deploy:
#   resources:
#     reservations:
#       devices:
#         - driver: nvidia
#           count: "all"
#           capabilities: [compute,utility,graphics,display]
```

`camera_enabled_ids` specifies which cameras are enabled in the scene for ArUco code detection and birdeye view.

1.5.1 Customized startup

In `config.sh` it is possible to customize your scenarios. From there you can edit what world you want to run, how many cameras you want enabled, and also edit Humanoid related properties. Modifying these variables are preferred, rather than modifying the `control.sh` file.

1.5.2 World fidelity

in the `config.sh` script, you can adjust the world fidelity

The active worlds are:

arguments	configuration
default	Contains the default world with maximum objects
medium	Based on default but boxes are removed
light	Based on medium but shelves are removed
empty	Everything except the ground is removed

1.5.3 Filtering log output

In `config.sh` you can set the level of logs you want outputted into the terminal. Per default it is set to “info” to allow all logs. Possible values are: “debug”, “info”, “warn”, and “error”. Setting it to “warn” filters out all debug and info messages. Additionally, to filter out specific lines you can add the phrase you want filtered, inside of `log_blacklist.txt` and setting the `log_level` flag to “warn” or “error” will start filtering out all phrases found in the blacklist.

1.5.4 Older versions

- [gz_classic_humble](#) branch contain code for **Gazebo Classic (Gazebo11)** that has reached end-of-life (EOL).
- [ign_humble](#) branch contain code for **ROS2 humble & Gazebo ignition**, an earlier version of this repository.

1.6 Documentation

Learn more about the project by reading these documents:

- [control.sh](#) script is a shortcut to run different launch scripts, please also see [these diagram](#).
- [config.sh](#) contains information about, which world is loaded, which cameras are active, what and where the robots are spawned.
- [Marp Markdown Presentation](#)
- [Pallet Truck Navigation Documentation](#)
- [Camera Utilities and notebooks](#): (Extrinsic/Intrinsic calibrations and Projection)
- [Humanoid Utilities](#) (pose2motion)
- [Configuration Generation](#)
- [simulation/](#): ROS2 packages
 - [Simulation and Warehouse Specification](#) (fidelity)
 - [Building Gazebo models](#) (Blender/Phobos)
 - [Objects Specifications](#)
 - [Warehouse Specification](#)
 - [Aruco Localization Documentation](#)
 - [humanoid_robot Simulation](#)
 - [Geofencing and Collision safe stop](#)
 - [Visualize Real Data](#) **requires data from Volvo**
 - [Humanoid Control](#)
- [CHANGELOG.md](#)
- [credits.md](#)
- [LICENSE](#) (apache 2)
- [contributing.md](#)

1.7 Research Funding

This work was carried out within these research projects:

- The [SMILE IV](#) project financed by Vinnova, FFI, Fordonsstrategisk forskning och innovation under the grant number 2023-00789.
- The EUREKA ITEA4 [ArtWork](#) - The smart and connected worker financed by Vinnova under the grant number 2023-00970.



SIMLAN project is started and is currently maintained by [Hamid Ebadi](#). To see a complete list of contributors see the [changelog](#).

1.7.1 3.1.0: Improvements (Dec 2025) - Supervisor: Hamid Ebadi

- System integration tests by Pär
- System diagram by Anton
- Pytorch model by Pär
- Improved doc and AutoGluon configuration by Marwa

1.7.2 3.0.0: Jazzy, humanoid (Oct 2025) - Supervisor: Hamid Ebadi

- Collision sensor: Anton Stigemyr Hill
- Automatically generated parameter-files for robot_agents and world fidelity : Anton Stigemyr Hill
- Shared map and obstacle generation for robot_agents: Anton Stigemyr Hill
- Geofencing and safety stop: David Espedalen
- Humanoid motion capture: Casparsson and Siyu Yi, Hamid Ebadi
- Humanoid integration (pymoveit2 to moveit_py): Siyu Yi, Pär Aronsson
- Mocap (humanoid) with multi-camera training pipeline: Siyu Yi, Pär Aronsson
- Panda arm integration: Pär Aronsson
- Dyno-robotics trajectory replay integration, scenario: Sebastian Olsson

1.7.3 2.1.0: Namespace cleanup and multi-agent navigation (Aug 2025) - Supervisor: Hamid Ebadi

- Implemented generic robot-agent control supporting multiple meshes (e.g., pallet truck, forklift) and distinct ArUco IDs: Pär Aronsson
- Robot-agent and navigation2 (nav2) namespace: Pär Aronsson
- Multi-agent navigation capabilities using GPSS: Pär Aronsson

1.7.4 2.0.0: GPSS (May 2025) - Supervisor: Hamid Ebadi

- Upgrade Gazebo from classic to ignition, adapting new sensors: Nazeen Alhosary
- Aruco_localization, added localization and nav2 to new robot: Pär Aronsson
- Navigation: Pär Aronsson
- Bird's Eye View ros package: Converting projection.ipynb to camera_bird_eye_view: Hamid Ebadi, Pär Aronsson
- Deprecated: Infobot, replaced with Pallet_truck (based on Jackal by Clearpath Robotics)

1.7.5 1.0.6: Collision (Feb 2025)

- Add scenario execution library for ros2
- Add action servers for set_speed, teleport_robot and collision
- Add Node for TTC calculation
- Add package for custom messages

1.7.6 1.0.5: Delivery 4 (Dec 2024)

- D3.5 Disentanglement: One-parameter Object Movements.
- Trajectory visualization : Hamid Ebadi
- feat: added DynoWaitFor to make sure Jackal is launched last (synchronisation issue)
- simlan_bringup pkg created : Christoffer Johannesson

1.7.7 1.0.4: Delivery 3 (Oct 2024)

- SMILE-IV and ArtWork projects contributions
- Updated camera extrinsic, fixing OpenCV camera calibration to Gazebo simulator conversion : Erik Brorsson, Volvo
- D3.4 Out distribution data collection : Hamid Ebadi
- D3.3 In distribution data collection : Hamid Ebadi
- D3.2 Images for stitching : Hamid Ebadi
- D3.1 Dataset draft for HH : Hamid Ebadi
- CI/CD and Kubernetes integration by Filip Melberg, Vasiliki Kostara
- Jackal integration : Christoffer Johannesson, Hjalmar Rusck
- Docker/vscode

- Disable GPU support by default
- POC for camera projection to pixel coordinates (jupyter notebook) : Hamid Ebadi
- Camera image dump and image stitching : Hamid Ebadi

1.7.8 1.0.3: Jackal Robot (Mar 2024) - Christoffer Johannesson

- Added dyno fork of jackal repo from Clearpath Robotics.
- Updated to Humble, added bringup and support for namespacing. Jackal can be spawned in Gazebo and controlled through the keyboard.
- Added .devcontainer folder with Dockerfile and devcontainer.json to set up project container in VS Code.
- Added docker-compose to link all needed files and set environment variables.
- Added .vscode folder with settings and tasks for easy building of the project.
- Updated README with info on how to use Docker setup in VS Code, and some features to make it easy to share the same setup with others.
- Features includes: python3 dependency install with pip, cloning of other git repositories and how to make changes to those repositories.

1.7.9 1.0.2: Delivery 2 (Feb 2024)

- Volvo warehouse 0.0.1 : Hamid Ebadi
- Volvo camera calibration in Gazebo 0.0.1 : Hamid Ebadi
- Integrate Infobot_agent 0.0.2: InfoBot differential-drive AMR (Autonomous Mobile Robot) URDF and ROS launcher (GOPAL and forklift): Hamid Ebadi
- Integrate Infobot_cartographer 2.1.5: cartographer for creating PGM maps
- Integrate nav2_commander 0.0.2: ROS package to command Infobot where the destination is : Hamid Ebadi
- Integrate Infobot_navigation2 2.1.5: Standard Nav2 stack launcher : Hamid Ebadi
- Integrate Infobot_teleop 0.0.2: Teleoperation for InfotBot

1.7.10 1.0.1: Delivery 1 (Dec 2023) - Supervisor: Hamid Ebadi

- Basic warehouse model 1.0.0: Anders Bäckelie
- CAD modelling (eur-pallet, boxes, shelf, support_pole, traffic-cone, steel_drum) 1.0.0 : Jacob Rohdin
- Physics (collision, inertia), visuals and Gazebo compatible mesh creation 1.0.0: Anders Bäckelie
- Walking actor using scripted trajectories 1.0.0 : Anders Bäckelie
- Infobot_Gazebo_environment 1.0.0: ROS2 launcher to start Gazebo world : Hamid Ebadi
- static_agent_launcher 1.0.0: Camera and Aruco tags : Hamid Ebadi
- camera-viewer 1.0.0: Python code to get Gazebo camera feed : Hamid Ebadi

2 Linux Dependencies

2.1 nvidia Driver

Use 'nvidia-smi' to ensure that the right nvidia driver is installed. If you have not installed **Additional Drivers** when installing Ubuntu, you need to manually install nvidia drivers.

2.2 Docker

```
sudo apt install curl git
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh
sudo groupadd docker
sudo usermod -aG docker $USER
newgrp docker
```

2.3 nvidia-container-toolkit

To install docker and nvidia-container-toolkit use the following commands:

```
curl -fsSL https://nvidia.github.io/libnvidia-container/gpgkey | sudo gpg --dearmor -o /usr/share/keyrings/nvidia-container-toolkit-keyring.gpg
&& curl -s -L https://nvidia.github.io/libnvidia-container/stable/deb/nvidia-container-toolkit.list | \
sed 's#deb https://#deb [signed-by=/usr/share/keyrings/nvidia-container-toolkit-keyring.gpg] https://#g' | \
sudo tee /etc/apt/sources.list.d/nvidia-container-toolkit.list
```

```

sudo tee /etc/apt/sources.list.d/nvidia-container-toolkit.list

sudo sed -i -e '/experimental/ s/^#//g' /etc/apt/sources.list.d/nvidia-container-toolkit.list

sudo apt-get update

sudo apt-get install -y nvidia-container-toolkit

sudo nvidia-ctl runtime configure --runtime=docker

INFO[0000] Config file does not exist; using empty config
INFO[0000] Wrote updated config to /etc/docker/daemon.json
INFO[0000] It is recommended that docker daemon be restarted.

sudo systemctl restart docker

sudo nvidia-ctl runtime configure --runtime=containerd

INFO[0000] Using config version 1
INFO[0000] Using CRI runtime plugin name "cri"
WARN[0000] could not infer options from runtimes [runc crun]; using defaults
INFO[0000] Wrote updated config to /etc/containerd/config.toml
INFO[0000] It is recommended that containerd daemon be restarted.

Restart the computer to apply the group and user changes.

To check for correct installation docker's nvidia runtime:

docker info|grep -i runtime
  Runtimes: nvidia runc
  Default Runtime: runc

Otherwise you get the following error message in vscode: Error response from daemon: unknown or invalid runtime
name: nvidia

On a host machine's terminal (not inside Visual Studio Code terminal): xhost +local:docker.

```

3 Windows Dependencies

These instruction is tested on Windows 11, docker desktop for windows-ARM64 and VScode.

To get the docker container up and running, download and install Docker Desktop for your system: <https://www.docker.com/products/desktop/>

When this is done, clone the repo and open the folder in VScode. Then you should automatically be prompted to download and install VScode extensions which are needed and recommended for the project.

Once that is completed, make sure Docker Desktop is running and then you should be able to start the container environment in VScode by pressing Ctrl+Shift+p and searching for Dev Containers: Rebuild and Reopen in Container.

To make Gazebo's and Rviz's GUI be visible from the docker container to your screen, download and install the following program: <https://sourceforge.net/projects/vcxsrv/>

When this is done, start the XLaunch program and configure it with these settings:

3.0.0.1 Select display settings Multiple windows Fullscreen One large window One window without titlebar

Display number: -1

3.0.0.2 Select how to start clients Start no client Start a program Open session via XDMCP

Before continuing, make sure Docker Desktop is running.

3.1 Legal Notice

When contributing to this project, you must agree that you have authored 100% of the content, that you have the necessary rights to the content and that the content you contribute may be provided under the project license.

3.2 Contributing

We try to use: - [Docker](#), [vscode + dev-container extension](#) to develop inside a Container - [Markdown](#) for documentation - [Conventional Commits](#) for commit messages - [pre-commit](#) for git pre-commit hooks and basic QA - [Semantic Versioning](#) for versioning - [Draw.io](#) for drawing diagram for documentation

3.3 Technical

- Avoid optimization unless you have a clear, measurable performance problem, as premature optimization can lead to overly complex and unreadable code.
- [PEP 20 – The Zen of Python](#)
 - Simple is better than complex.
 - Readability counts.
 - If the implementation is hard to explain, it's a bad idea.
- Please be generous in giving credit when using an image or a piece of code created by others. Add a link to the original author in [credits.md](#)
- Avoid pushing binary files such as images, models, etc (use `.gitignore`) specially if they are constantly changing.

3.3.1 git

- Try avoid breaking the `main` branch but don't be obsessed with having a perfect merge request. We can always revert back to the working version or fix the issues. That is why we are using `git`.
- Follow this simple git process:

```
git checkout main
git pull
git checkout -b "branch_name"
# update files
git commit -m "conventional_commit_type: conventional_commit_message"
git push
# Create a pull request.
```

common commit_type:

- feat: Introduces a new feature.
- fix: Patches a bug or issue.
- chore: Routine maintenance or changes that don't affect the app's functionality.
- docs: Documentation changes.

conventional_commit_message: - A conventional commit message uses the imperative mood in the subject line

example of - feat(auth): add login functionality - fix(ui): resolve button color issue

<https://www.conventionalcommits.org>

3.3.2 Misc

- Don't use absolute path. Your code should run correctly both inside the VS Code Dev Container and independently outside of it. If you need to modify a configuration file at build time, use relative paths from the project root directory (avoid using `..` as it becomes hard to see which scripts are using a specific directory). For ROS2 resources, use `get_package_share_directory(project_name)` to locate package files.
- When adding a new feature, add it to `control.sh`. Otherwise, `control.sh` should remain unchanged.

- Any configuration that requires modification should be defined in `config.sh`

3.3.3 Documentation

Documentation is done in a markdown file. Try to keep our documentation close to your code. Keep the documentation short and clear.

- `#` : Only used once for the title of the project
- `##` : usually once in each markdown file and usually share semantic with the markdown filename.
- `###` : Different topics, try to split your documentation into at least 4 topics
- `####` : sub topics. Try to avoid when you can use simple paragraphs

Follow this subset of [Markdown](#) tags.

3.4 automatically generated parameter files:

When building the workspaces, the generation-scripts found in the `scripts/` directory are run. These automatically generate the `bt.xml`, `control.yaml`, `gz_bridge.yaml`, `nav2_params.yaml`, aswell as validates the setup of the `ROBOTS` list in the `config.sh` script. This is done to automatically set up the ros2 controllers, navigation stack parameters and other /topics depending on what robots and namespaces are spawned.

Every time the workspace is built, the `generate_XXX` files prints that they were generated and at the top of each generated file, a comment specifies from where the auto-generation occurred. As of now the parameter files are saved in the package in which they're used in instead of directly in the `share/` directory to increase code readability to new users.

Explanation of the different `generate_XXX` files:

file	explanation
<code>generate_bt_xml.py</code>	The <code>bt_navigator</code> needs to know which robot it is checking conditions for, therefore the namespace of the robot is needed
<code>generate_control.yaml.py</code>	The <code>ros2_controller</code> maps <code>/cmd_vel</code> topics to <code>wheel_joint</code> movements. Therefore the correct namespace and <code>frame_id</code> are needed
<code>generate_gz_bridge.yaml.py</code>	Some <code>gz_topics</code> need to be mapped to ros2 topics. Therefore the corresponding namespaces are needed
<code>generate_nav2_params.yaml.py</code>	The individual frame ids and some ros2 topics are needed to define robots and their corresponding maps
<code>validate_robots.py</code>	A converter from string to list of dicts including a sanity check to throw errors if the <code>ROBOTS</code> variable is wrongly configured

3.5 World Fidelity

It is possible to adjust the level fidelity for a world in `config.sh`, there the `world_setup` is sent to `- simulation/simlan_bringup/launch - simulation/simlan_gazebo_environment/launch/simlan_factory.launch.py` (for world generation) - `simulation/simlan_gazebo` (both `original_world` and the `world_setup`)

3.6 Adding Aruco and camera (static agents)

Aruco codes and cameras are all attached to the same link in Gazebo. To create new static agents, go to `simulation/static_agent_launcher`. There you only need to add a new line on the form `<xacro:camera number="1" x="3" y="3" z="6" r="0" p="0" w="0"/>` for a new camera, or a new line on the form `<xacro:aruco number="1" x="3" y="3" z="0.1" r="0" p="0" w="0"/>` for a new aruco. The commands are identical apart from the name.

- `Number` is added to the name to make the static agent unique. For cameras this mean they will publish on e.g. the topic `static_agents/camera_[number]/image_raw`. For aruco the number indicates which aruco png to use.
- `x`, `y`, `z` is the coordinate offset from the link the agents are all attached to.
- `r`, `p`, `w` are the rotation, pitch, and yaw of the camera, dictating in which direction and at what angle the cameras are looking.

Gazebo supports [simulation of camera](#) based on the Brown's distortion model. It expects 5 distortion coefficients `k1`, `k2`, `k3`, `p1`, `p2` that you can get from the camera calibration tools. The `k` coefficients are the radial components of the distortion model, while the `p` coefficients are the tangential components.

- `<aspect_ratio>` : The ratio of the width and height of the camera.

- `<horizontal_fov>`: The horizontal field of view of the camera in radians.

OpenCV uses five parameters, known as distortion coefficients given by like this: `k1, k2, p1, p2, k3` # pay attention to the order

3.7 Using actors in Gazebo

- Placed in sdf or world file
- Actors use the actor tag (as opposed to the model tag most other objects use).
- actor tags contain links like usual, but also a `<script>` tag.
- The script tag contains:
 - `loop`: Whether the script should loop on completion
 - `delay_start`: Time to wait before running the script, also waits between loops
 - `auto_start`: Whether the script should start automatically when the sim starts
 - A trajectory tag, which has an (unique) id and a type (used to couple it with an animation)
 - * Inside the trajectory tags we define waypoint tags which consist of a pose and the time where we are supposed to reach the pose.
 - * Note: The trajectory is smoothed as a whole. This means that you'll get a fluid motion, but the exact poses contained in the waypoints might not be reached.

Script structure:

```
<script>
  <loop>1</loop>
  <auto_start>1</auto_start>
  <trajectory id='0' type='square'>
    <waypoint>
      <time>0</time>
      <pose>-1 -1 1 0 -0 0</pose>
    </waypoint>
    <waypoint>
      <time>1</time>
      <pose>-1 1 1 0 -0 0</pose>
    </waypoint>
    <waypoint>
      <time>2</time>
      <pose>1 1 1 0 -0 0</pose>
    </waypoint>
    <waypoint>
      <time>3</time>
      <pose>1 -1 1 0 -0 0</pose>
    </waypoint>
    <waypoint>
      <time>4</time>
      <pose>-1 -1 1 0 -0 0</pose>
    </waypoint>
  </trajectory>
</script>
```

- Gazebo supports two different skeleton animation file formats: COLLADA (.dae) and Biovision Hierarchy (.bvh).
 - .bvh: Text-based format with section 'HIERARCHY' and section 'MOTION'.
 - .dae: XML-based format, structured into multiple sections.
- .bvh works for Ignition, while .dae works for both Gazebo classic and Ignition
- Skin is similar, simply import a collada file
- `<interpolate_x>true</interpolate_x>` makes the animation match the movement. The animation is done briefly along the x-axis and then it can be interpolated into any direction by gazebo.
- In a top down positive x,y frame of reference: 0 = right, 1.57 = up, -1.57 = down, 3.14 = left
 - Pos/negative matters! 0 to -1.57 does not behave the same as 0 to 4.71!
 - In other words increasing the angle of rotation means rotating left, and decreasing it means rotating right.
- Tension = how strictly it follows the waypoints in a span 0-1 where 0 is not very strict, 1 very strict

3.8 Simulation Specification

Environment (world)

- An in-door factory warehouse
- An out door street/sidewalk (side walk or street) (FUTURE WORK)

Objects

- human actor
- shelves
- [standard euro pallets](#)
- [pallet racks](#)
- traffic cone
- Boxes
- [differential-drive AMR \(Autonomous Mobile Robot\)](#) (FUTURE WORK)
- forklift (FUTURE WORK)

Textures

- Aruco tag on actor and floor
- objects texture
- Floor texture

Placement of sensors

- camera (depth) on ceiling
- 2D Lidar/camera on AMR (FUTURE WORK)

Physics

- mass
- moment of inertia
- collision
- friction (FUTURE WORK)
- damping

Lighting

Agents movements:

- deterministically following waypoints trajectories (in curve or in form of A-B-C...Z)
- AMR differential-drive using nav2 with obstacle avoidance (FUTURE WORK)
- replaying scenario using rosbag (FUTURE WORK)

Additionally these are expected from the simulator:

- following realistic and standard measurements and sizes for warehouse, pallettes, shelves, cart, human body, etc
- agent/robot status (position) and control via ROS
- exposing and recording camera images and agent status (e.g. positions) in Python
- tele-operation with keyboard and programmatically from python

3.9 AMR

differential-drive (casterwheel)/achermann (car) type of robot

/cmd_vel topic accepts `Twist` formatted messages only non-zero value for linear.x (forward/backward) and non-zero value for angular.z (differential drive, rotating in z axis)

3.10 Warehouse

The warehouse was created in FreeCAD's BIM Workbench. This workbench isn't available by default but can easily be acquired by going to Tools -> Addon Manager -> BIM -> Install/update Selected.

Using the measurements found in the blueprint in the documentation folder I drew four lines and turned them into walls (line tool and wall tool respectively). By default the wall will be created around the line, so that the line is in the middle of the wall. This can be changed in the wall properties so that it ends up entirely on one side of the line. As the blueprint lacked walls I used the dimensions specified there as the inner measurements, meaning that the origin point is located at the inner bottom left corner of the wall. All the inner space is in positive x and y coordinates, while the two of the walls

are in negative coordinate space. I used the Aligned Dimension tool from the Annotation tools to confirm that the inner measurements matched those of the blueprint.

While the BIM Workbench has support for door objects, they tend to act as solids when imported into gazebo, so the door is just a hole in the wall. This hole was created by adding a cube object and having it intersect the wall where the door should be located. Then you select the cube and the wall (in that order) in the tree view and press **Remove Component**. This should remove the cube object and create a hole where the cube and wall overlapped. The hole didn't appear in the right place, but it was possible to edit the position of the hole in its properties. I measured the location of the hole using Aligned Dimension to make sure it ended up in the right spot.

Going to the top view, I created a new rectangle object covering the whole warehouse. Then I turned it into a slab with the slab tool to create a floor for the warehouse.

Everything was added to a level object (note: by default this level object is named "Floor", but don't confuse it with the slab that constitutes the physical floor) so that it's grouped together. If we set the wall heights to 0 they will automatically inherit the height of the Level object, so we can change all the walls easily by changing the height of the level.

I added two materials from presets, concrete and wood. I applied the concrete material to all walls and the wood material to the floor. These should be considered to be temporary placeholders.

Finally I exported the project as a Collada file (.dae) and added it into a simple .world file to see if it loaded properly in Gazebo, and the results seemed correct.

Since the warehouse will be static we shouldn't need to define any additional parameters like mass or inertia, visuals and collision should be enough. Textures might need some improvement as currently they're just basic colors but it should be possible to add those in FreeCAD and have them included in the .dae file so we can load them visually in Gazebo later.

I also added windows for slightly better visibility.

3.11 Add warehouse to world-file

The floor of the warehouse goes below $z=0$ so the ground plane was lowered by 0.2 so that the warehouse still rests on top of it. As our simulations will mainly take place inside the warehouse the warehouse floor replaces the ground plane at $z=0$. The warehouse itself was placed in the models directory and loaded into the world with an `<include>` tag. Additionally the maze in the stage4 world was moved away from the origin so that it is fully contained inside the warehouse, though in the future it should be removed altogether.

3.12 Textures

Shelf, pallet, warehouse walls are using CC0 textures from <https://polyhaven.com/textures> Box and warehouse floor are using images from Volvo as a base.

- box: picture from [Volvo-group-packaging-specifications_2015.pdf](#)

3.12.1 Conventions

- Keep your links/joints paired, and use the suffix `_link` and `_joint` (e.g. `arm_link` and `arm_joint`) and maybe follow [REP 120 naming conventions](#)
- Define visual, collision, inertial and Gazebo material (and maybe friction) for all objects

4 Converting FreeCAD meshes to SDF models (for Gazebo)

4.1 Script step-by-step

- Install freecad `sudo apt-get install freecad=0.19.2+dfsg1-3ubuntu1` (Tested with version 0.19)
- Install Blender v3.3 LTS
- Download phobos.zip (tested with [phobos 2.0 "Perilled Pangolin"](#))
- Install phobos by following [the guide on their github](#)
- In your phobos settings (easiest accessed through the blender GUI), change the the models folder to where you want the output to be. You can add a relative path here so that the output folder depends on where you run the script, e.g. setting the model folder to `./phobos_out/` would create a `./phobos_out/` directory as a sub-directory to wherever you run the script and save the exported sdfs there.
- Enter freecad and load your `.FCstd`, then select the body in the tree view and select **File->Export** and choose to export it as a Collada (.dae) file.

- Run `./build.sh`

You should now be able to load the model/directory created by the reformat script directly through Gazebo.

```
wget https://download.blender.org/release/Blender3.3/blender-3.3.0-linux-x64.tar.xz
tar -xf blender-3.3.0-linux-x64.tar.xz
mv blender-3.3.0-linux-x64 blender
alias blender="`pwd`/blender/blender"
blender --version
git clone git@github.com:dfki-ric/phobos.git
cd phobos
git checkout 2.0.0
blender -b --python install_requirements.py
cd ..
./build.sh
```

4.2 In GUI

- Import `.dae` file
- Set phobostype to `visual`
- Select the object and set the `geometry` type
- [Guide](#) says to `Object->Apply->Rotation & Scale` here but I'm not sure what it actually accomplishes
 - This will apply the scaling of objects to their mesh information (vertices) and applies the current orientation to the mesh as well. After this option all objects have zero orientation. If this is not desired one can also only apply the `Scale`.
- Create a “visual” collection and put the model into there, this lets us create a tree structure. (unsure of necessity when we only have one model)
- Select object and press `Create Link(s)`, make sure `selected objects` mode is chosen.
- Select the visual and then `Create Collision`.
- Select the object and then `Create Inertials`.
 - Here we need to manually enter our mass, should check if that can be adjusted somehow.
- Save and export.
- Then we need to move around the resulting files so they're structured in the way Gazebo wants them, i.e.:

```
model
  meshes
    filename.dae
  model.config
  model.sdf
```

In order to accomplish this we'll have to create the `model.config` file, however this is relatively simple as it doesn't contain any unique data between different models aside from the model name.

Example of a successful build:

```
simulation/raw_models$ ./build_models.sh

../simlan_gazebo_environment/models/aruco/materials/textures/0.png
../simlan_gazebo_environment/models/aruco/materials/textures/1.png
../simlan_gazebo_environment/models/aruco/materials/textures/2.png
../simlan_gazebo_environment/models/aruco/materials/textures/3.png
```

Color management: using fallback mode for management

Color management: Error could not find role data role.

Blender 3.0.1

Read prefs: [HOME]/.config/blender/3.0/config/userpref.blend

Color management: scene view "Filmic" not found, setting default "Standard".

/usr/lib/python3/dist-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.17.3 and <1.25.0 is required
warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}")

Checking requirements:

ensurepip is disabled in Debian/Ubuntu for the system python.

Python modules for the system python are usually handled by dpkg and apt-get.

```
apt install python3-<module name>
```

Install the python3-pip package to use pip itself. Using pip together with the system python might have unexpected results for any system installed module, so use it on your own risk, or make sure to only use it in virtual environments.

WARNING: We couldn't do ensurepip, we try to continue anyways

```
Checking yaml
Checking numpy
Checking scipy
Checking collada
Checking pydot
Checking lxml
Checking networkx
Checking trimesh
Checking PIL
Importing phobos
IMPORT: phobos.blender.defs
Parsing definitions from: [HOME]/.config/blender/3.0/scripts/addons/phobos/data/blender/definitions
defaultControllers.yml
defaultSensors.yml
defaultMaterials.yml
defaultJoints.yml
defaultSubmechanisms.yml
defaultMotors.yml
Creating new definition type: joints
Creating new definition type: motors
IMPORT: phobos.blender.display
IMPORT: phobos.blender.io
RELOAD: phobos.blender.model
IMPORT: phobos.blender.operators
RELOAD: phobos.blender.phobosgui
RELOAD: phobos.blender.phoboslog
RELOAD: phobos.blender.phobossystem
RELOAD: phobos.blender.reserved_keys
RELOAD: phobos.blender.utils
Registering operators.selection...
Registering operators.io...
Registering operators.editing...
TypeError: EnumProperty(..., default='mechanism'): not found in enum members
ValueError: bpy_struct "PHOBOS_OT_define_submodel" registration error: 'submodeltype' EnumProperty could not r
Registering operators.naming...
Registering operators.misc...

Registering phobosgui...
... successful.
+-- Collada Import parameters-----
| input file      : ./objects/meshes/eur-pallet.dae
| use units      : no
| autoconnect    : yes
+-- Armature Import parameters ----
| find bone chains: yes
| min chain len   : 0
| fix orientation : yes
| keep bind info  : no
IOR of negative value is not allowed for materials (using Blender default value instead)+-- Import Scene -----
```

```

| NODE id='node0', name='node0'
+-----
| Collada Import : OK
+-----
Error in sys.excepthook:

Original exception was:
File "[HOME]/.config/blender/3.0/scripts/addons/phobos/blender/operators/editing.py", line 1048, in toggleVisu
Error in sys.excepthook:

Original exception was:
File "[HOME]/.config/blender/3.0/scripts/addons/phobos/blender/operators/editing.py", line 1051, in toggleColl
[20231211_08:52:53] WARNING No text file README.md found. (phobos/blender/utils/blender.py - readTextFile (125
Collada export to: [PATH]simulation/raw_models/phobos_out/unnamed/meshes/dae/Body.dae
Info: Exported 1 Objects
Info: Exported 1 Objects
Collada export to: [PATH]simulation/raw_models/phobos_out/unnamed/meshes/dae/Body.dae
Info: Exported 1 Objects
Info: Exported 1 Objects
Info: Phobos exported to: phobos_out/unnamed
Info: Export successful.
Info: Phobos exported to: phobos_out/unnamed
Info: Export successful.

-----
Unregistering Phobos...
Unregistering phobosgui...
Unregistering display...
Unregistering icons...
Unregistering classes...
Unregistering manuals...
... successful.

Blender quit
Error: Not freed memory blocks: 14, total unfreed memory 0.002426 MB

```

4.3 Blueprint

This document contains explanations and motivations for the measurements as well as names for variables that should be used when designing the objects. The categories are mostly self explanatory, the one that might need more of an explanation is the color red. This refers to loose objects such as pallets, barrels, boxes and things that will be placed directly on the ground.

Measurements:

Category	Part / Object	Variable Name	Value [unit]
Shelf	Load Beam	beam_length	2.7 [m]
Shelf	Load Beam	beam_depth	0.05 [m]
Shelf	Load Beam	beam_height	0.14 [m]
Shelf	Load Beam	beam_weight	15.2 [kg]
Shelf	Footplate	footplate_length	0.111 [m]
Shelf	Footplate	footplate_width	0.1 [m]
Shelf	Footplate	footplate_height	0.004 [m]
Shelf	Stand / Pole	stand_length	0.07 [m]
Shelf	Stand / Pole	stand_width	0.08 [m]
Shelf	Stand / Pole	stand_height	3 [m]
Shelf	Stand / Pole	stand_weight	8.5 [kg]
Shelf	Cross Brace	brace_length	0.96 [m]
Shelf	Cross Brace	brace_width	0.01 [m]
Shelf	Cross Brace	brace_height	0.02 [m]

Category	Part / Object	Variable Name	Value [unit]
Shelf	Cross Brace	brace_weight	1 [kg]
Loose Object	EUR-Pallet	EUR_pallet_length	1.2 [m]
Loose Object	EUR-Pallet	EUR_pallet_width	0.8 [m]
Loose Object	EUR-Pallet	EUR_pallet_height	0.144 [m]
Loose Object	EUR-Pallet	EUR_pallet_weight	25 [kg]
Loose Object	Steel Drum	steel_drum_radius	0.3 [m]
Loose Object	Steel Drum	steel_drum_height	0.9 [m]
Loose Object	Steel Drum	full_steel_drum_weight	188 [kg]
Loose Object	Steel Drum	empty_steel_drum_weight	15 [kg]
Loose Object	Traffic Cone	traffic_cone_radius	0.22 [m]
Loose Object	Traffic Cone	traffic_cone_angle	-10 [degrees]
Loose Object	Traffic Cone	traffic_cone_total_height	1 [m]
Loose Object	Traffic Cone	traffic_cone_base_length	0.52 [m]
Loose Object	Traffic Cone	traffic_cone_base_width	0.52 [m]
Loose Object	Traffic Cone	traffic_cone_base_height	0.03 [m]
Loose Object	Traffic Cone	traffic_cone_base_angle	-20 [degrees]
Loose Object	Traffic Cone	traffic_cone_weight	6.5 [kg]
Loose Object	Box	box_length	Varies [m]
Loose Object	Box	box_width	Varies [m]
Loose Object	Box	box_height	Varies [m]
Loose Object	Box	box_weight	Varies [m]
Warehouse	Support Pole	pole_length	0.3 [m]
Warehouse	Support Pole	pole_width	0.3 [m]
Warehouse	Support Pole	pole_hole_side_length	0.27 [m]
Warehouse	Support Pole	pole_hole_side_width	0.13 [m]
Warehouse	Support Pole	pole_height	6 [m]
Warehouse	Support Pole	pole_weight	Undefined [kg]

EUR-pallet

EUR-pallet Measurements

4.4 AMR

AMR

It is defined in XY plane and the center of the AMR is on x,y=0,0 and the bottom of the robot is on z=0.

Define **scale** , **pos** and collision in these files:

We write our AMR description in `simlan_gazebo_environment/urdf` and use `xacro` to create a `urdf` file with gazebo tags (so not a pure urdf file) that can be used both by `state_publisher` and `gazebo` (unlike 2 separate files `model.sdf` that is used for Gazebo and `turtlebot.urf` that is used for state_publisher in original Turtlebot_simulation git project).

There are two actual diffdrive wheels that are named `left` and `right` and four supporting wheel to balance the robot that are named `front_left`, `front_right` and `back_left` and `back_right` (they have no friction and can be moved to different position) with the radius of 0.98 of the main wheels.

Note: The reason for the shake is difference between the radius of the real wheels and caster wheels

4.4.1 Infobot AMR specification

Orientation:

- x : forward
- y: left
- z: up

4.5 Objects Modeled in FreeCAD

All the objects are located in `objects`. Below follows a list of the objects currently available, models created in FreeCAD.

- EUR-pallet
- Shelf
- Modular Shelf - a stackable shelf part that can create shelves of varying size
- Steel Drum
- Traffic Cone
- Support Pole
- Boxes with measurements in mm [Box-Length x Width x Height]
 - Box-160x130x70
 - Box-185x185x75
 - Box-185x185x185
 - Box-210x180x130
 - Box-230x160x85
 - Box-250x195x160
 - Box-430x250x260
 - Box-440x320x175
 - Box-570x380x380
 - Box-1185x785x1010
 - Box-600x800x400 (2 per level on EUR-Pallet)
 - Box-600x400x400 (4 per level on EUR-Pallet)
 - Box-300x400x400 (8 per level on EUR-pallet)

4.6 AMR camera

Camera

Inspired by [CCTV Camera 3G-SDI](#)

In the model: camera base height(70mm) + camera lense height(50mm) = 120mm

4.7 Resources

- [Pallet Rack Specification & Configuration Guide](#)
- [Pallet Rack Estimator](#)
- [Pallet rack configurator](#)
- [pallet](#)
- Shelf typical measurements: [Shelf Measurements](#)
- Forklift height which influences Shelf height: [Forklift Height](#)
- EUR-pallet: [EUR-pallet](#)
- Shelf design: [Shelf design](#)
- Shelf exact measurements: [Shelf exact measurement](#)
- Examples of boxes: [Boxes Examples](#)
- Measurement for steel drum: [Steel Drum](#)
- Full Steel drum weight is based on the oil density of 825kg/m³ and the drum fits 210 liters.
- Some values helping in constructing the traffic cone: [Traffic Cone](#)

4.8 Blueprint

This document contains explanations and motivations for the measurements as well as names for variables that should be used when designing the objects. The categories are mostly self explanatory, the one that might need more of an explanation is the color red. This refers to loose objects such as pallets, barrels, boxes and things that will be placed directly on the ground.

Measurements:

Category	Part / Object	Variable Name	Value [unit]
Warehouse	Walls	warehouse_length	23.2 [m]
Warehouse	Walls	warehouse_width	18 [m]
Warehouse	Walls	warehouse_height	6 [m]

Category	Part / Object	Variable Name	Value [unit]
Warehouse	Aisle	aisle_width	3.7 [m]
Warehouse	Door	door_height	4 [m]
Warehouse	Door	door_width	3 [m]
Warehouse	Warehouse	door_distance_to_wall	2 [m]
Warehouse	Shelf	shelf_distance_side_side	0 [m]
Warehouse	Shelf	single_shelf_wall_distance	0.1 [m]
Warehouse	Shelf	double_shelf_depth_distance	0.2 [m]
Warehouse	Shelf	shelf_total_length	2.88 [m]
Warehouse	Shelf	shelf_total_depth	1.1 [m]
Warehouse	Shelf	shelf_total_height	3 [m]

blueprint

The storage blueprint

4.9 Install instruction for BIM and Arch workbench

The warehouse was created in FreeCAD’s BIM Workbench. This workbench isn’t available by default but can easily be acquired by:

- Installing `pip3 install gitpython`
- Activating Tools -> Addon Manager -> BIM -> Install/update Selected

Using the measurements found in the blueprint in the documentation folder I drew four lines and turned them into walls (line tool and wall tool respectively). By default the wall will be created around the line, so that the line is in the middle of the wall. This can be changed in the wall properties so that it ends up entirely on one side of the line. As the blueprint lacked walls I used the dimensions specified there as the inner measurements, meaning that the origin point is located at the inner bottom left corner of the wall. All the inner space is in positive x and y coordinates, while the two of the walls are in negative coordinate space. I used the Aligned Dimension tool from the Annotation tools to confirm that the inner measurements matched those of the blueprint.

While the BIM Workbench has support for door objects, they tend to act as solids when imported into gazebo, so the door is just a hole in the wall. This hole was created by adding a cube object (3D/BIM -> Cube) and having it intersect the wall where the door should be located. Then you select the cube and the wall (in that order) in the tree view and press **Modify -> Remove Component**. This should remove the cube object and create a hole where the cube and wall overlapped. The hole didn’t appear in the right place, but it was possible to edit the position of the hole in its properties. I measured the location of the hole using Aligned Dimension to make sure it ended up in the right spot.

Going to the top view, I created a new rectangle object covering the whole warehouse. Then I turned it into a slab with the slab tool to create a floor for the warehouse.

Everything was added to a level object (note: by default this level object is named “Floor”, but don’t confuse it with the slab that constitutes the physical floor) so that it’s grouped together. If we set the wall heights to 0 they will automatically inherit the height of the Level object, so we can change all the walls easily by changing the height of the level. Textures for the floor and walls can later be added in Blender.

Finally I exported the project as a Collada file (.dae) and added it into a simple .world file to see if it loaded properly in Gazebo, and the results seemed correct.

Since the warehouse will be static we shouldn’t need to define any additional parameters like mass or inertia, visuals and collision should be enough. Textures might need some improvement as currently they’re just basic colors but it should be possible to add those in FreeCAD and have them included in the .dae file so we can load them visually in Gazebo later.

I also added windows for slightly better visibility.

4.10 Add warehouse to world-file

The floor of the warehouse goes below $z=0$ so the ground plane was lowered by 0.2 so that the warehouse still rests on top of it. As our simulations will mainly take place inside the warehouse the warehouse floor replaces the ground plane at $z=0$. The warehouse itself was placed in the models directory and loaded into the world with an `<include>` tag. Additionally the maze in the stage4 world was moved away from the origin so that it is fully contained inside the warehouse, though in the future it should be removed altogether.

4.11 Resources

Warehouse Aisle width: [Aisle Width](#)

5 forklift_robot

A simple Forklift Robot URDF and PROTO (for webots).

Add it to your catkin workspace and run `catkin build`

5.1 URDF

One can see the URDF by running: `roslaunch urdf_tutorial display.launch model:='$(find forklift_robot_description)`

```
<physics type="ode">
  <ode>
    <solver>
      <type>world</type>
    </solver>
    <constraints>
      <contact_max_correcting_vel>0.1</contact_max_correcting_vel>
      <contact_surface_layer>0.0001</contact_surface_layer>
    </constraints>
  </ode>
  <max_step_size>0.001</max_step_size>
</physics>
```

Two solvers: world step gives an accurate solution if it is able to solve the problem, while quick step depends on the number of iterations to reach an accurate enough solution.

5.1.1 contact/collision parameters

dampingFactor double Exponential velocity decay of the link velocity - takes the value and multiplies the previous link velocity by (1-dampingFactor). maxVel double maximum contact correction velocity truncation term. minDepth double minimum allowable depth before contact correction impulse is applied maxContacts int Maximum number of contacts allowed between two entities. This value overrides the max_contacts element defined in physics.

contact_max_correcting_vel : contact_max_correcting_vel This is the same parameter as the max_vel under collision->surface->contact. contact_max_correcting_vel sets max_vel globally. contact_surface_layer : contact_surface_layer This is the same parameter as the min_depth under collision->surface->contact.

Note: We had issue getting global settings (contact_max_correcting_vel and contact_surface_layer) working. We therefore define object level minDepth and maxVel for each object

“quick” solver parameters min_step_size The minimum time duration which advances with each time step of a variable time step solver. iters The number of iterations for the solver to run for each time step.

5.1.2 objects:

As of now there are many objects in the world file and if you want to move them all it's quite cumbersome. Therefore the move_objects.py was added to be able to increment the poses by x amount if needed. See move_objects.py

5.1.3 reference:

https://classic.gazebosim.org/tutorials?tut=physics_params&cat=physics <http://sdformat.org/spec>

6 Static_agent_launcher

This package launches all the static agents i.e. the cameras in the simulation.

When the workspace is built, the camera_config.xacro file is updated with camera_ids and their corresponding intrinsic and extrinsic values. This can be seen in the buildfunction in control.sh.

The `camera_config.xacro` is then sent to the `robot_state_publisher` node and published to the `/static_agent/robot_description` topic. Then the `ros_gz_sim` package with the “create” executable spawns the `static_agents` by subscribing to the `/static_agent/robot_description` topic and spawning each agent.

6.0.1 gz_bridge for cameras

different configurations of the `gz_bridge` node were tested and evaluated against the simulated RTF (real time factor). In the earlier setups, a `gz_bridge` node was setup for each camera_stream (image, depth ,semantic) for each camera_id. 3 configurations were tested where different amount of nodes were setup and their corresponding RTF was captured and averaged over 5 tries.

SN=single node for all cameras, 1NPC=1 node per camera_id, 1NPSPC = 1 node per stream per camera_id

node setup	number of cameras	number of nodes	camera streams	avg RTF
SN	9	1	image	17,9%
1NPC	9	9	image	16,7%
SN	9	1	image depth semantic	5,14%
1NPC	9	9	image depth semantic	5,2%
1NPS	9	81	image depth semantic	4,7%

doesn't seem like a big difference so will stick with one node i.e. SN.

In the future it can be decided wether to keep the `gz_bridge` node as is or to create a generate `gz_brige_camera.yaml` file as the other generation scripts `/home/ros/src/config_generation`

7 bird-eye-view package

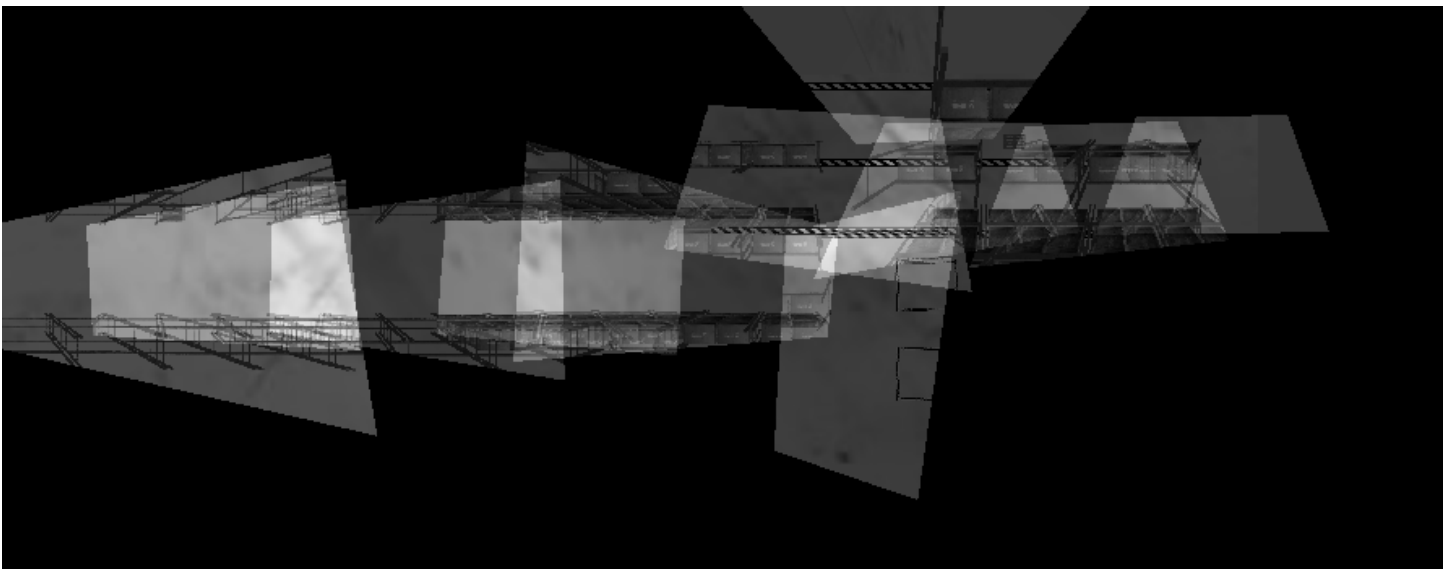
This package aims to contain projection functionality which originates from [projection.ipynb](#) notebook.

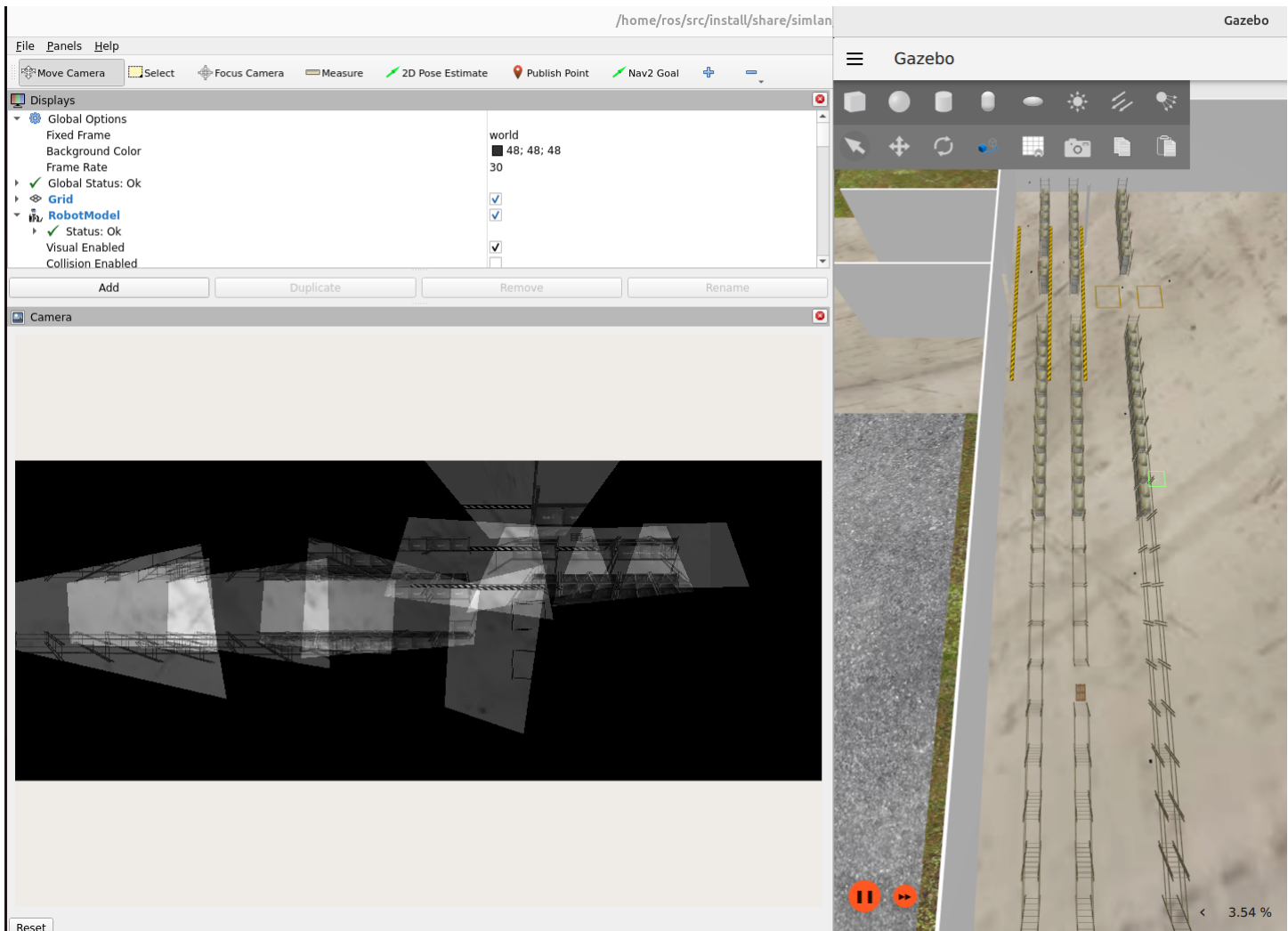
This package features being able to select areas of your choosing and create a bird-eye-view of that area by using available cameras, viewing that area.

Command to run birdeye launch file:

```
./control.sh birdeye
```

Screenshots:





8 SIMLAN bringup

This package is responsible for launching all things related to the simulation. The launch files in the package calls launch files in all other packages to start everything up with the following command

```
ros2 launch simlan_bringup full_sim.launch.py
```

8.1 Launch arguments

There are different launch arguments that can be changed to launch the simulation in different states. These can either be called from the terminal by appending the argument to the launch command above.

```
rviz:=True'
```

Another way to edit what is launched it to change the *default value* in [full_sim.launch.py](#).

```
launch_rviz_launch_argument = DeclareLaunchArgument(          "rviz",          default_value="False",
description="To launch rviz")
```

The launch arguments are then either added as a condition straight to a Node:

```
condition=IfCondition(rviz)
```

or passed downward to the launch file being called from the top level launch file:

```
launch_arguments={"jackal_manual_control":jackal_manual_control,}.items()
```

If you use a specific argument configuration often it is best to create a new launch file, on top of [full_sim.launch.py](#). It can have the correct default values for all arguments and then just call on [full_sim.launch.py](#).

8.2 Diagram of launch structure

This diagram is made in DrawIO and the png contains the xml code. Drop it into [drawio](#) to make changes and add back to this readme. launch_structure

9 pallet_truck

Common packages for pallet_truck, including messages and robot description. These are packages relevant to all pallet_truck workspaces, whether simulation, desktop, or on the robot's own headless PC.

10 SIMLAN project

prefix is used to publishing unique base link names of each robot agent to **/tf**. This way they all visible in rviz and probably it is used for odometry done by aruco localisation.

namespace is used to separate nodes and topic related to each robot agent. This way we can control each robot separately and run a separate nav2 stack.

Have in mind that the value of namespace is used for prefix but they are different concept and usecases.

10.1 Pallet Truck bringup.

This package handles initialization and status of spawned robots.

- [gazebo.launch.py](#) - Spawns robot, handles robot_state_publisher, robot_description
- [keyboard_steering.launch.py](#) - keyboard steering.
- [multiple_robot_spawn.launch.py](#) - contains configurable list of robots you spawn in the sim.
- [sim.launch.py](#) - main launch file for single robot. Make sure gazebo is running, control, twist_mux, keyboard_steering as all launched.
- [rviz.launch.py](#) - runs rviz

The package focuses on pallet_truck for now but could be made modular to spawn other types of robots.

10.2 Configuring and spawning robots inside the sim.

Below is the structure which we use to spawn robots inside of the sim. Please read these notes before setting up a new or editing a robot.

Attributes for spawning a robot:	Attribute	Description	Example
	<code>namespace</code>	Used to differentiate between multiple robots. Follows the format <code>robot_agent_N</code> , where N is the robot ID.	<code>"robot_agent_1"</code>
	<code>initial_pose_x</code>	The robot's initial x-coordinate position. Float value wrapped as a string.	<code>"10.0"</code>
	<code>initial_pose_y</code>	The robot's initial y-coordinate position. Float value wrapped as a string.	<code>"1.0"</code>
	<code>robot_type</code>	Selects the robot mesh or appearance. Options are <code>"pallet_truck"</code> or <code>"forklift"</code> .	<code>"pallet_truck"</code>
	<code>aruco_id</code>	Sets the ID shown on the robot's ArUco marker. Must match the ID in the namespace.	<code>"1"</code> if namespace ID is 1

Example robot setup:

```
{
  "namespace": "robot_agent_1",
  "initial_pose_x": "10.0",
  "initial_pose_y": "1.0",
  "robot_type": "pallet_truck",
  "aruco_id": "1"
}
```

10.3 automatically generated parameter files:

Some parameter files are automatically generated for the pallet_truck_bringup package. The generation scripts can be found in the `~/src/config_generation/` directory. The automatically generated files include the `nav2_params.yaml`, `gz_bridge.yaml` and more. To look at all the automatically generated files, build the workspace and the generated files will be printed in the terminal or look manually in the `~/src/config_generation/` directory.

10.4 Pallet_truck_control

The pallet_truck_control package includes all nodes necessary for the control of the robots. These include the following:

- **node:** description
- **twist_mux:** Takes in velocity topics and orders them in order of relevance. from highest to lowest: /key_vel, /safety_vel, / scenario_vel, /nav_vel, /cmd_vel
- **twist_stamper:** As of ros2 jazzy the ros2_controllers need the twist messages to be stamped, therefore the stamper was introduced
- **ros2_control_node:** Main node which uses the control.yaml file and maps “hardware” to controller actions.
- **spawner-joint_state_broadcaster:** Publishes the joint_state so the robot moves in gazebo and rviz.
- **spawner-velocity_controller:** Accepts and publishes velocity commands

With this setup, the robot_agents can be controlled by running the teleop command in control.sh.

11 pallet_truck Description

This packages contains the meshes and URDF of the pallet_truck robot, its supported sensors, and their supported mounts.

11.1 Sensors

- **GPS:** Novatel Smart6 and Smart7
- **2D LiDAR:** SICK LMS1xx
- **2D LiDAR:** Hokuyo UST-10
- **3D LiDAR:** Velodyne VLP16 and HDL-32E
- **Camera:** Flir/Pointgrey Flea3 and Flea3 Stereo
- **Camera:** Flir/Pointgrey Bumblebee2
- **Camera:** Flir/Pointgrey BlackflyS

11.2 documentation on pallet_truck_description

In addition to the change_log, this description is added to give vital information that may help other developers in the future to debug or update features faster.

11.3 Additions:

Here additions to the urdf.xacro files are mentioned

11.3.1 Collision sensor

A collision_sensor was added and linked to the mesh tag of the pallet_trucks. This is integrated in the pallet_truck.urdf.xacro.

To visualize the topic subscribe to: /namespace/collision e.g. /robot_agent_1/contact.

The topic publishes information about the position, torque and which models are in contact with each other.

The topic name is defined by the automatically generated gz_bridge() which can be found in /home/ros/src/simulation/pallet_truck/p

11.4 Old Bugs

Here is some information of old bugs which may be useful to know for future development

11.4.1 xacro/urdf/sdf files

Gazebo only reads sdf files and if .xacro or .urdf files are used, they are later parsed to .sdf's before being used by Gazebo.

A “bug” for Gazebo harmonic using ros2 jazzy was that when adding the collision sensor for the pallet_trucks, Gazebo was unable to find the collision tag for the mesh that we were using.

After investigation it was found that when the .xacro file was parsed to an .sdf file, the name of the collision tag was updated by the parse plugin. Therefore this lumped renaming of the collision tag was “hard coded” instead of using its original name which is in the pallet_truck.urdf.xacro.

original: “\$(arg prefix)/chassis_link::collision”

hard coded: “\${prefix}_base_link_fixed_joint_lump__collision_collision”

When adding sensors in the xacro file, make sure the plugins are also loaded. There are different world and robot plugins so if system plugins are used they should most likely be added to the .world file

11.4.2 Map server

Instead of starting with an empty map, to get the general layout of the environment and detect the static object in the environment (walls, corridors and hallways) you can **optionally** build a map. To start mapping (otherwise known as cartography), you can use `cartography.launch.py`. It requires a lidar to be mounted on the `robot_agent` and `odometry` to exist, which can be activated from the `camera_utility/aruco_localization` package.

Launch the following package, run gazebo, rviz, `aruco_localization`, and start moving around the simulation with the `robot_agent`. When you are finished you need to save it using this command: `ros2 run nav2_map_server map_saver_cli -f simulation/pallet_truck/pallet_truck_navigation/maps/YOUR_MAP_NAME`. Keep in mind that this step needs to be done only once and the map assumed to be static.

We only have one map for all navigation stack. This simplify making obstacle and other robot agent to all other robot agents.

```
Node( # Manually setting the joint between map and odom to 0 0 0, i.e. identical to each other. map -> odom
    package="tf2_ros",
    executable="static_transform_publisher",
    name="static_world_to_map",
    arguments=["0", "0", "0", "0", "0", "0", "world", "map"],
    ...
)
```

11.4.3 Robot_agent navigation

This package covers the functionality of mapping, localizing, and navigation for the `robot_agent`. Each package is built using code from the `nav2` packages, thus you are able to modify the launch files but to change the node's behaviour you need to modify the config files. The code is tailored for the `robot_agent` and using the `aruco_detection` package that supplies /TF chain for the truck.

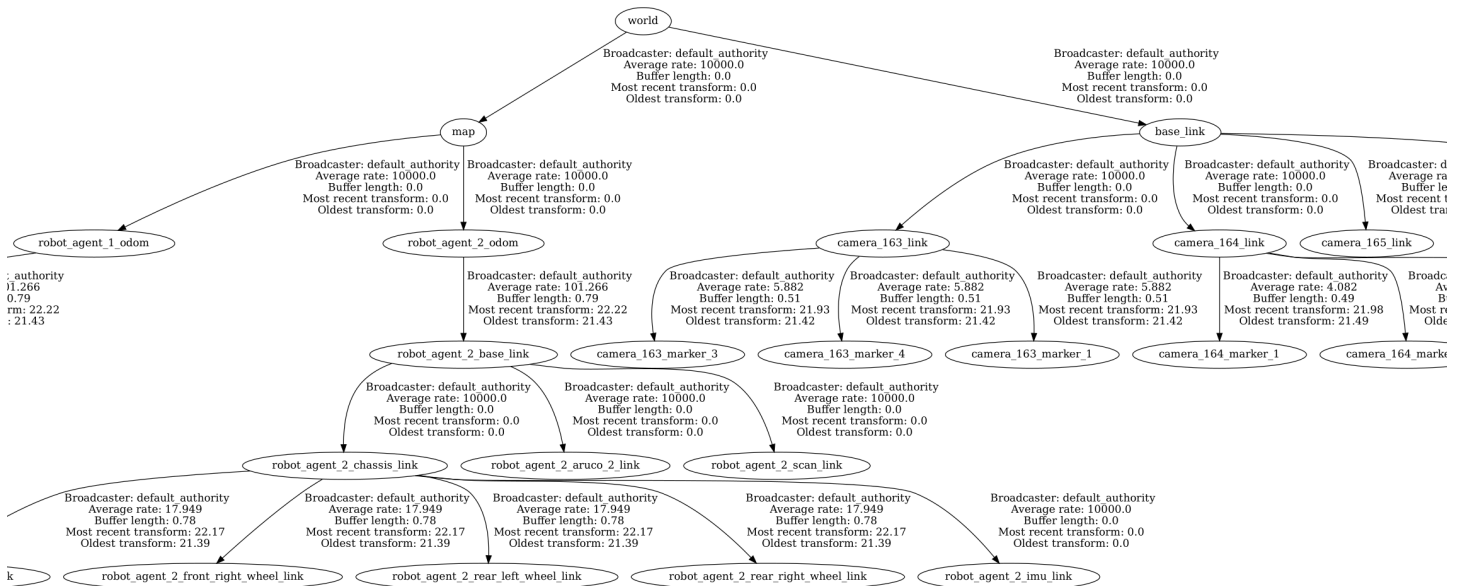
- The `aruco_localization` package runs the aruco localisation.
- The `map_server.py` creates a map for each agent
- The `nav2.launch.py` uses the localisation (from the previous steps) for the navigation of the `robot_agent`.

Keep in mind that the same namespace has to be used when launching the navigation stack which is done automatically: `ros2 launch pallet_truck_navigation nav2.launch.py robots:=ROBOTS`

For navigation to be able to automatically find the robot agent, we have to set a namespace that specified when launching the `robot_agent`. (see above)

Each robot has it own navigation configuration in `nav2_params.yaml` which is dynamically generated on build based on the `ROBOTS` variable in `config.sh`. the `pallet_trucks` and `forklifts` also have an individual `navigate_w_replaning_and_recovery_robot_ag` which is supposed to stop the navigation of the agents if their `aruco_markers` are not detected. The topic which publishes the “seen” `aruco_markers` is called `/aruco_marker_seen` and can be visualized after running `gpss` and `nav`.

We use following /TF structure:



Note that `world` and `map` are static at the same position. `robot_agent_X/odom` is static at the position from where the robots are spawned and the new position of the robots are then determined by `robot_agent_X/base_link` which is a dynamic transform from `robot_agent_X/odom`.

Good video to watch for multi agent navigation: <https://www.youtube.com/watch?v=cGUueuIAFgw>

11.4.4 Humanoid_navigation

The humanoid navigation is implemented in the same way as for the robot_agents with the difference of calling the function `./control.sh nav HUMANOIDS`. Then the same principles apply but with a different `nav2_parameter` file being used as an argument to `nav2` nodes. At the moment there is an issue with the actual TF values in `rvis` and simulation in `gazebo` not matching which is described in `simulation/humanoid_support_moveit_config/README.md` under bugs. When this is fixed, the navigation should work.

There is one major difference between the navigation of the humanoids and the robot_agents. The robot_agents get their `odom` frame from the `aruco_localization` pkg which find `aruco_markers` and publishes their orientation and position. The humanoids on the other hand get their `odom` frame from the `ros2_controller simulation/humanoid_support_moveit_config/launch/la`. This means the robot_agents get their `odom` frame from what the cameras can see and the humanoids get their `odom` frame from what the `ros2` controller publishes.

11.5 Obstacles detection (update_map_node)

We have static objects in the map and initially we updated that single map with dynamic obstacles (Robot_agents). The problem with this single map was that `nav2` navigation stack of `robot_agent_1` sees itself within the obstacle created for `robot_agent_1` (**itself**). Therefore it is decided that each `robot_agent` has its own map that has everything except itself. So this is a node to create a map with dynamic obstacles around all **other** existing robot_agents in the simulation. Each robot is assigned a `map_server` and this node sends an updated map with all other robots_agents as obstacle to the correct namespaced `map_server`.

The node is launched in `pallet_truck/pallet_truck_navigation/launch/nav2.launch.py`

The `/map_updater/update_map_node.py` node works as following:

First it makes a copy of the `warehouse.pgm` map which is a long array of pixel values ranging from (0-255) and saves it as `original_array`. Since the map `nav2` needs has a different setup than the `.pgm` file, a conversion is needed.

The `.pgm` file has these pixels ranges

value	meaning
0	black obstacle
255	White Free space
1-254	ranges of gray, not defined at the moment

value	meaning
-------	---------

The map nav2 needs is set up with pixels ranging from (-1-100) where

value	meaning
-1	Unknown
0	Free space
100	obstacle
1-99	probabilistic occupancy

therefore this has to be parsed to match by doing the following:

```
current_array = self.original_array.copy()
occupancy_array = np.zeros(current_array.shape, dtype=np.uint8)
occupancy_array[current_array.copy()<100] = 100
```

So everything that is supposed to be a wall in the .pgm file is parsed as a wall in the map context

The map is a 2D array, typically stored row-major from bottom-left, but many implementations flip it vertically ($[:, -1]$) to match how image viewers treat top-left as (0,0). and therefore the following is done

```
occupancy_array = occupancy_array[:, :-1, :]
```

Then the position of all other robot_agents are found by their transforms between robot_agent_x/base_link and world and an obstacle is set at their position which is updated with 10Hz to continuously update their positions.

This concept is repeated for all namespaces sent through the ./control.sh nav function.

How often the new path the pallet_trucks can take be update add a speed limiter instead of obstacle, in those cases the robot will slow down instead of replanning and avoiding completely update slower could probably mix with the inflation radius's of the costmaps to make robot take wider turns

Collision Two robots that are approaching each other (from left and right) don't know where each one is planning to go and they can make the decision to pick the path on top lane instead of one going above and one from bottom.

Possible solution: block future path on every other robots map!

11.5.1

Package created based on this [tutorial](#)

Note that only the human with mesh model is installed in the package, see /model/CMakeLists.txt for how to install the non-mesh models.

In the URDF, the fixed links and joints are (mostly) named [muscle name]_[location], for example BicBrac_RUA = biceps brachii_right upper arm. In humanSubjectWithMesh_simplified.urdf all fixed joints have been removed, only the skeleton joints remain.

To open Rviz with the human model:

```
ros2 launch urdf_tutorial display.launch.py model:=/home/ros/src/simulation/humanoid_robot/model/human-gazebo/
```

12 human-gazebo

This repository contains the human gazebo models that are used with [Human Dynamics Estimation](#) software suite. The files are generated using [xsens motion capture](#) data and [mvnx-to-urdf](#). The human model links are made of several simple rigid bodies as shown in the figure below:

Human model image

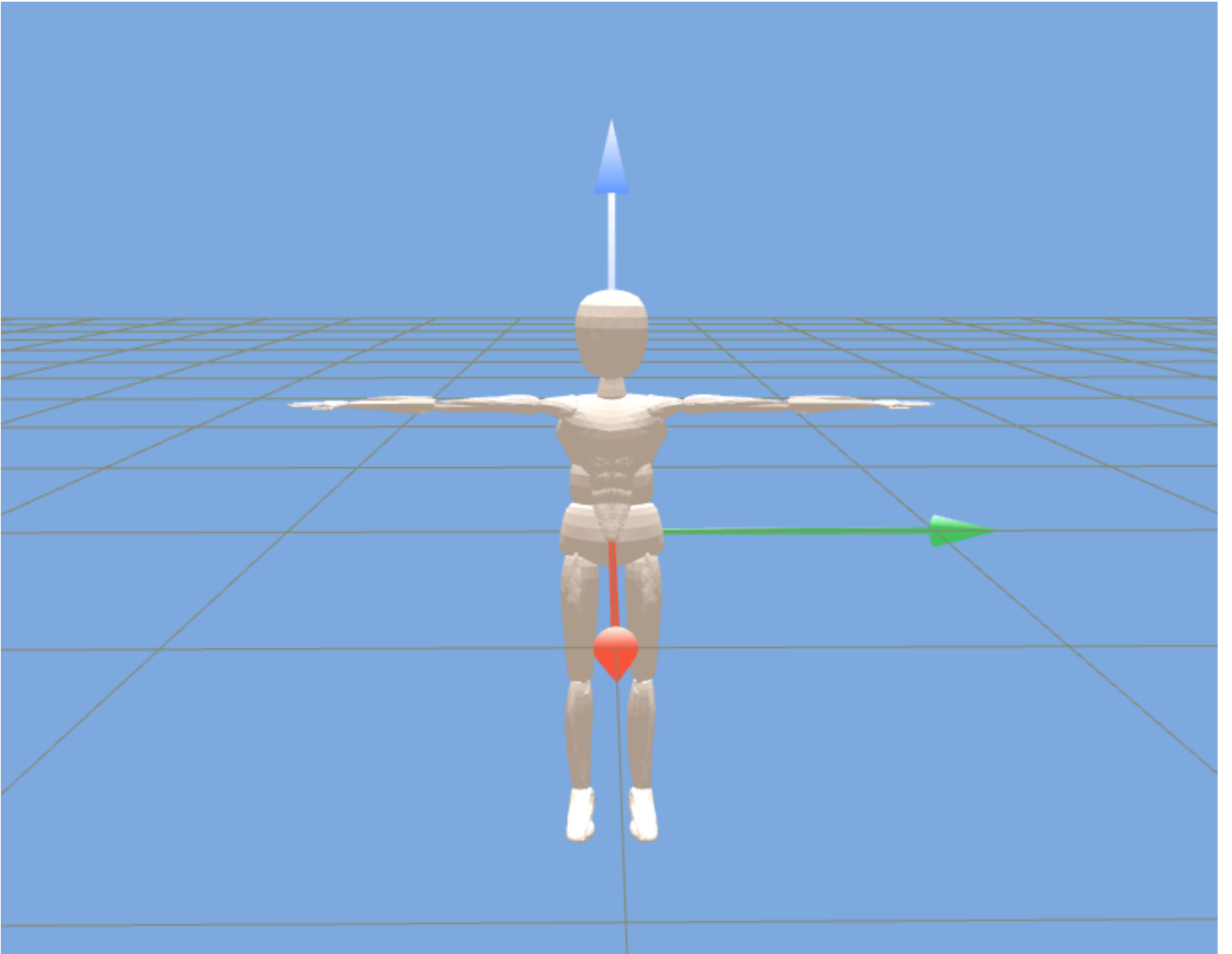
The measurements of each of the human subject are available from the table. Please refer to the human subject data pdf file to know how these measurements are taken. The urdf models are generated from xsens mvnx file generated through xsens mvn studio software suite. The code to generate the model is available [here](#)

Subject	Mass [kg]	Height [cm]	Foot size [cm]	Arm span [cm]	Ankle height [cm]	Hip height [cm]	Hip width [cm]	Knee height [cm]	Shoulder width [cm]	Shoulder height [cm]	Sole height [cm]
1	62.2	168	24	163	8	91	25	48.5	35.4	140	-
2	79.4	176	26	169	8	94	33	48	40	140	-
3	75.4	180	27	190	8	102	28	58	43	148	-
4	72.7	182	26	197	8	102	29	56	42	150	-
5	55	168	24	168	8	98	25	52	38	139	-
6	71.2	179	29	180	8	100	31	49	43	147	-
7	78.9	178	28	192	8	102	30	52	44	148	-
8	55.2	166	25	170	8	90	28	45	37	139	-

Currently, the legacy directory contains files related to joint motor control boards based on [gazebo-yarp-plugins](#) and other configuration files needed to control the human joints.

12.1 Human subject with meshes

In the folder **humanSubjectWithMeshes** there is a **urdf** model of a human subject generated using the code in [human-model-generator](#) with meshes under CC-BY_SA license (<https://creativecommons.org/licenses/by-sa/2.0/deed.en>); all the meshes were trimmed, morphed and totally or partially reconstructed to reach the desired shape and topology. The model is shown in the following figure:



12.2 Mantainers

- Davide Gorbani (@davidegorbani)
- Carlotta Sartore (@CarlottaSartore)

The meshes in this folder have been derived from <https://blendswap.com/blend/11604>.

This module is for the control of the human model in gazebo.

13 aruco_localization package

This is the `aruco_localization` package, which runs a `ros2` node that takes images from camera topics `CAMERA_X/camera_info` and `CAMERA_X/image_raw`, process the aruco code, and publishes the result in publishes the `/TF` of the detected ArUco marker, mimicking Volvo's logic for tracking robot positions and generating navigation routes. A launch file is provided to run multiple nodes, with each node dedicated and assigned to a single camera. `camera_enabled_ids` variable found in `src/control.sh` is used to control which cameras are enabled.

Note: The marker link is named based on the camera. For example, if camera `164` detects an ArUco marker with ID of `12`, the marker link in `/tf` is named `camera_164_marker_12`. This is not the final link that is used. The result from different cameras are accumulated and merged as a single frame with parent as `robot_agent_12_odom` and child `robot_agent_12_baselink`.

Note: The marker ID is tightly linked with the name for the robots, meaning an aruco with marker `ID=1` will detect the robot with name: `{namespace}_{marker_ID}`. For example 'robot_agent_1' Currently the namespace is hardcoded into the package and can be made more abstract.

13.1 Important points to pay attention to:

- If two cameras are pointed at the same ArUco code, the system does not alternate between them, but instead shows the midpoint between the two detections.
- Inside the `aruco_node`, a callback logs the relative position of the newly created `aruco_link` from the `base_link`.
- Multiple cameras can be linked to the same ArUco code simultaneously.

13.2 Important launch files

- `aruco_detection_node.py`: Handles ArUco detection and publishes the TF between the camera and marker. (e.g. `camera_164_marker_12`)
- `aruco_pose_pub.py`: A new node that listens to all transforms of `camera_164_marker_12` as input, and outputs the transform to either TF or ODOM as `robot_agent_12_odom` and child `robot_agent_12_baselink`.
- `multi_detection.launch.py`: Launch file for the new node.

13.3 Odom

These has to be valid for both humanoid and pallet trucks

- `world` : is the origin (0,0,0)
- `odom` : should not be in the world frame but where the robot is spawned
- `base_link`: tracks the movement of the robots relative to the odom

Coordinate

14 FAILSAFE for the navigation of pallet trucks

The SIMLAN systems support a failsafe and geo-fencing mechanism for the pallet trucks navigation. The main idea is that the pallet trucks should stop the navigation as soon as a dangerous situation below are detected:

In this approach for a new Behavior Tree **condition node** named `StopRobotCondition` is defined that can be triggered when a safety issue is occurred. We then added a `Behavior Tree plugging` that when this condition is triggered, a standard action called `CancelControl` .

Currently these safety controlled triggers `StopRobotCondition`:

14.0.1 Activation of collision sensor

When the collision is detected by the simulator, the collision's physical properties (such as the force and the object that pallet truck collided) are published in the pallet trucks `/contact` topic.

14.0.2 Loss of Observability

To implement geofencing and the safety situation in which a pallet truck is not observable in any camera. `aruco_localization` pkg under `aruco_localization/aruco_pose_pub.py` continuously published the list of pallet truck that are not observable in `/aruco_marker_lost` topic.

14.1 Behavior tree and direct implementation in `aruco_localization` pkg

At first we define a custom `behavior_tree.xml` in `src/simulation/pallet_truck/pallet_truck_navigation/config/navigate_w`. This xml files defines which bt plugins we want to use during the navigation and which are ran continously during the navigation. In this xml-file the `StopRobotCondition` and `CancelControl` plugins are defined in a fallback function. A fallback function works as its run the first stated plugin, and when that plugin fails it moves over to the second one an executes that plugin. So in this case we firs run the `StopRobotCondition` plugin until the robot is out of bounds and the plugin fail. And then the `CancelControl` plugin executes and stops the pallet truck.

In order to know whether a pallet truck is out of bounds or not is determined by looking at the aruco marker on the pallet truck and decide if its seen by any of the cameras or not. This is implemented in the `aruco_localization` pkg under `aruco_localization/aruco_pose_pub.py`. As soon as the aruco marker on the pallet truck isn't seen by any of the cameras the node publishes the robot namespace in a list to the topic `/aruco_marker_lost`.

Same for the collision sensor. It publishes to the topic `/robot_agent_X/contact` when the pallet trucks collide with something.

Step two in the failsafe is to stop the pallet truck when it gets out of bounds or collide with something. This is done by a custom made behavior tree plugin `StopRobotCondition` which is find in `SIMLAN/simulation/bt_failsafe/src/stop_robot.cpp`. This plugin listen to the `/aruco_marker_lost` and `/robot_agent_X/contact` topic and as soon it publishes it fails and the `CancelControl` behavior tree starts, which finally stops the pallet truck. The `CancelControl` plugin is a existing built in plugin in Nav2.

14.1.1 Why Nav2 behavior tree plugins was not suitable

We tried to use only built in plugins which most likely is the most robust and secure way to do it on as the plugins is updated accordingly to Ros2 and Nav2.

We tried to use several plugins to detect if the pallet trucks are out of bounds, but non of them really suits this project and purpose.

The first one we tried to use is the `TransformAvailable` plugin (https://github.com/ros-navigation/navigation2/blob/main/nav2_behavior_trees/behavior_trees/transform_available.xml). This plugin checks if a certain tf exists and if its missing the plugin returns `FAILURE`. So in this case we thought we could look at the TFs between the cameras and the aruco marker, and if non of them exists it should fail and stop the pallet truck. Unfortunately we couldn't use this plugin because it only look at the tf from the very beginning of the navigation. And if it exists from the beginning it will always succeed and will never return `FAILURE`. Therefore it cant be use in our case because we have TF from the beginning, and our TF disappears after some time during the navigation. You could probably modify the plugin to work for our case as well, but that isn't a way we wanted to go with this.

The second plugin we tried is the `IsStuckCondition` (https://github.com/ros-navigation/navigation2/blob/main/nav2_behavior_trees/behavior_trees/is_stuck.xml). This plugin checks if the robot is stuck by calculating the deacceleration of the robot. If the deacceleration is to big it returns `FAILURE`. This wasn't anything we could use because the acceleration is set to zero as soon as the robot get out of bounds. So this is probably not suitable at all for a case like ours.

15 Humanoid

In this file, the basics of the humanoid structure will be described. In the future all nodes will be namespaced to add the functionality of spawning and controlling multiple humanoids.

15.0.1 Moveit

For moveit to work 3 component must have correct information:

- URDF link names
- SRDF joint definitions
- TF frames published by robot_state_publisher

There was an issue when trying to add namespaces to the humanoid project. Since the humanoid is made of 40+ links, it was decided to add the "frame_prefix": f"{namespace}" in the robot_state_publisher node. This however made a conflict since the URDF, SRDF and TF frames no longer matched. This was solved by adding base_link in between the world and namespace/base_link frames and creating a static transform between them. **This means that all further humanoids will be spawned under 'base_link frame.** (otherwise the warning below again appears)

15.0.2 Dynamically updated urdfs

to make the multiple_humanoid_spawn work we need to be able to launch different namespaces in the robot_description. this is done by running:

```
moveit_config = (
    MoveItConfigsBuilder("human_support", package_name="humanoid_support_moveit_config")
    .robot_description(
        file_path="config/human_support.urdf.xacro", mappings={"namespace": namespace}
    )
    .to_moveit_configs()
)
```

this updates the namespace argument inside the .xacro files

15.0.3 Planning scene monitor

When working with the moveit package, some bugs or undesired features were found. When moveit is launched, the PlanningSceneMonitor and PlanningFrame subscribes to all TF_frames that exists and assumes it can transform any known object or sensor data into its planning frame which is base_link in this case. If this cannot be done it will throw a warning saying the following:

[WARN] [humanoid.moveit.moveit.ros.planning_scene_monitor] [id]: Unable to transform object from frame 'unconn

to limit this a wait function was added: ld.add_action(TimerAction(period=5.0, actions=[rsp_node])) to make sure the transform is published before the rsp_node starts

15.0.4 Figures

My Robot Diagram *Figure 1: Humanoid frames visualization*

15.1 Rviz2 visualization

If you want to visualize the movement in rviz you need to configure the config/moveit.rviz file and change all /humanoid_Xinstances to the namespace you want to visualize.

15.2 Bugs

15.2.1 TF visualization and gazebo simulation not matching

When visualizing and comparing the actual TF data and simulated locations of the humanoids when doing navigation nad teleoping it can be seen that these do not match. This is a major issue since that means the location of where the humanoids think they are in the map and the location of where the humanoids actual are in gazebo will be different. This can be visualized by turning the humanoid 360 degrees in gazebo with the ./control.sh teleop humanoid_1 command and comparing to rviz. There the humanoid has turned closer to 300 degrees.

This problem probably comes from some urdf descriptions not matching the actual geometry or specifications the humanoid has in /home/ros/src/simulation/humanoid_support_moveit_config/config/human_support_wheels.urdf.xacro. By tuning the mass, mu and inertial values in the <xacro:macro name="wheel" params="wheel_prefix *joint_pose"> xacro tag it was possible to tune the degree mismatch.

For proper navigation and control this needs to be fixed!

15.3 Preparing Data

15.3.1 First build the package

for example with:

```
colcon build --merge-install --symlink-install --packages-select visualize_real_data
```

After that, you can add the required data into the `share/` folder that is created during the build process.

If everything has been built correctly you should find the following folder-structure:

```
install/  
  share/  
    visualize_real_data/  
      data/  
        images/  
          images  
        data
```

The empty `data` and `images` placeholder files ensure that ROS 2 recognizes and preserves the folder structure during the build and should **not** be removed.

If you have already built the package before there might be another directory inside the `data/` folder called `rosbags`. More on that folder later.

15.3.2 Add the data after building

After the build completes, you can add your data files to the generated `share/` folder inside your package.

By default the package expects to find the trajectory data inside the `data/` folder as `.json` and the images inside the `images/` folder as `.jpg`. Like this:

```
install/  
  share/  
    visualize_real_data/  
      data/  
        images/  
          image1.jpg  
          image2.jpg  
        window_2012_1746008869618_1746008890118.json
```

Notes:

- The **folder-names are configurable**, as long as the settings in `params.yaml` match.
- You can:
 - Rename the `images/` folder to something else.
 - Rename the trajectory JSON file to any other valid filename.
- Just be sure to update the corresponding fields (`images_folder`, `json_file_name`) in `params.yaml`.

15.3.3 Important Notes

- You do *not* need to rebuild the package after adding or modifying data in `share/`.
- In fact, **rebuilding the package could overwrite or remove any manually added files** in the `share/` folder, so avoid rebuilding once your data is in place.

15.4 Config file

When your data is in place, you may want to review the config parameters in `params.yaml`. This step is often optional, as default settings typically work well.

15.4.1 params.yaml Overview

15.4.1.1 prepare_data section:

- The Parameters here have to be set before `prepare.launch.py` is run. Changes made afterwards will not effect the preparation process (image->PointCloud2).

Parameter	Description	Default Value
<code>pointcloud_topic</code>	Topic to publish the PointCloud2 (image data).	<code>pointcloud_topic</code>
<code>images_folder</code>	Name of the folder containing images.	<code>images</code>
<code>image_scale</code>	Resize factor for .jpg images (useful for RViz display).	<code>0.04</code>
<code>set_frames</code>	If <code>true</code> , limits processing to <code>frames_to_process</code> ; otherwise, all available images will be used.	<code>false</code>
<code>frames_to_process</code>	Number of frames to process if <code>set_frames</code> is <code>true</code> .	<code>1</code>
<code>preprocess_all_data</code>	If <code>true</code> , preprocesses all available data before playback. May not work for extremely large recording windows.	<code>true</code>
<code>fake_orientation</code>	If <code>true</code> , automatically fakes object orientations during data preparation (used by the scenario replayer).	<code>true</code>

15.4.1.2 send_data section:

- Parameters here can be changed after the prepare-stage to alter the playback. *You do not need to rebuild the package or re-run `prepare.launch.py` if you change something here.*

Parameter	Description	Default Value
<code>playback_rate</code>	Controls playback speed. Default is 1 (real-time based on <code>extracted_fps</code>).	<code>1</code>
<code>frame_position</code>	x,y,z coordinates for visualization frame.	<code>x: 15.35, y: 6.5, z: -0.2</code>
<code>bag_name</code>	Name of the rosbag to play. If empty, the most recent one is used.	<code>(empty)</code>

15.4.1.3 scenario_replayer section:

Parameter	Description	Default Value
<code>gazebo_teleport_service</code>	Service name for teleporting robots in Gazebo.	<code>/world/default/set_pose</code>
<code>frame_id</code>	Frame in which entities are visualized.	<code>real_data</code>
<code>use_cmd_vel</code>	If <code>true</code> , robots are driven with velocity commands; otherwise, teleportation is used to reenact the scenario. When set to <code>true</code> it is important the simulation runs close to 100% real-time.	<code>true</code>

15.4.1.4 shared section:

- Parameters here have to be set at the prepare-stage and can't be changed afterwards.

Parameter	Description	Default Value
<code>frame_id</code>	Name of the frame in which all data is displayed.	<code>real_data</code>
<code>namespace</code>	Namespace used for the node.	<code>visualize_real_data</code>
<code>entity_topic</code>	Topic to publish the <code>MarkerArray</code> (trajectory data).	<code>entity_topic</code>
<code>json_file_name</code>	Name of the file containing trajectory data.	<code>(empty)</code>
<code>extracted_fps</code>	FPS of the extracted data for playback.	<code>10.0</code>
<code>processing_time_limit</code>	Max time allowed per frame for consistent playback. If exceeded, a warning appears.	<code>0.8</code>

15.4.2 Launching the Processing Step

Once configured, run:

```
ros2 launch visualize_real_data prepare.launch.py
```

This will first start the `orientation_fixer` node which adds an orientation to the data based on the angle between points in the trajectory. Afterwards the data processing is started and generates a rosbag file stored in a `rosbags/` folder inside the package's `share/` directory. > **NOTE:** If orientation is already given, change

- Rosbags are named using the name of the JSON file used and timestamps (e.g., `my_recording_20250722_153045` if the JSON file is named `my_recording.json`).
- Existing rosbags **are never overwritten**.
- You can delete old rosbags manually from the `rosbags/` folder if needed.

15.5 Sending Data

After processing the data you can send it to RViz:

1. Make sure you've added the following displays to RViz:

- `PointCloud2`
- `MarkerArray`

2. Run:

```
ros2 launch visualize_real_data send.launch.py
```

This command sends the most **recently created rosbag** to the topics defined in `params.yaml`.

- To play an earlier dataset, you must manually delete newer rosbags as the latest one (by timestamp) is always selected automatically.

15.5.1 Adjusting Playback Speed

You can modify the `playback_rate` parameter in `params.yaml` to control how quickly the rosbag is replayed.

- A value of 1 reflects real-time playback based on extracted timestamps.
- Slower or faster playback is supported, but *extreme values haven't been tested*.

15.5.2 Fixing RViz

- Usually the displays in RViz have to be configured to subscribe to the correct topics, and the `PointCloud2` displays size have to match the `image_scale` defined in the `params.yaml` config file to look right.
- You can also change `Alpha` in the `PointCloud` display to see through the image.
- If the images aren't showing up, sometimes the QOS (Quality of Service) settings might be mismatched between the `rosbag player` and RViz. The intended QOS settings can be found in the `recorder_qos.yaml` file in the `config` folder together with the `params.yaml` file. These are the settings that the rosbag player uses, and the easiest fix is to make sure that RViz mirrors these settings for the respective displays.

15.6 Summary

1. Build package using `CTRL + SHIFT + B` VS Code task.
2. Add the data you want to show in RViz in the package's `share/` folder
3. Configure parameters under 'prepare_data', 'shared', and/or 'scenario_replayer' (*optional*)
4. Run `prepare.launch.py`
5. Open RViz2 and add the displays `PointCloud2` and `MarkerArray`
 - Make sure the QOS-settings match between the rosbag player and the displays
6. Configure parameters under 'send_data' (*optional*)
7. Run `send.launch.py`
8. Subscribe to the relevant topics (as defined in `params.yaml`) in RViz2

Now the data should be visualized.

15.6.1 Warnings and Errors from the node

Warnings and Errors thrown by this package are intended to inform you when something unexpected occurs during execution. In most cases, the process can continue without interruption, although the output may be affected.

It is strongly recommended to resolve the underlying issue based on the warning/error message and then **re-run the process** to ensure reliable and consistent output.

16 Scenario Manager

16.1 Overview

The `scenario_manager` package is a ROS2 package designed to manage and execute various robot scenarios involving teleportation, speed setting, and collision simulations. It provides action servers to control robot behavior and a tool for calculating Time to Collision (TTC).

16.2 Launching the Scenario Manager

To start the `scenario_manager`, launch it using the following command:

```
ros2 launch scenario_manager scenario_manager.launch.py
```

This initializes three action servers: - **teleport_action_server**: Handles teleportation of robots. - **set_speed_action_server**: Controls robot speed. - **collision_action_server**: Configures scenarios where robots collide (only applicable to `jackal` and `pallet_truck`).

16.3 Collision Action Server

The **collision_action_server** configures collisions by setting speeds and initial positions for two robots to ensure a collision occurs. It requires: - **Angle**: The approach angle. - **Speed of pallet_truck**: The movement speed of the pallet truck. - **Collision Type**: An enum from the `simlan_custom_msg` package specifying the type of collision.

16.3.1 Collision Types

There are three predefined collision types:

Collision Type	Enum Value	Description
HEAD_ON	0	Robots collide head-on.
PALLET_TRUCK_SIDE	1	Pallet truck collides into the side of Jackal.
JACKAL_SIDE	2	Jackal collides into the side of the pallet truck.

Below are images illustrating the different collision types:

Left: Head-On Collision | Middle: Pallet Truck Side Collision | Right: Jackal Side Collision

16.4 Running a Scenario

To execute a scenario, use the following command:

```
ros2 launch scenario_execution_ros scenario_launch.py scenario:=<scenario_file>
```

Replace `<scenario_file>` with a specific scenario file, such as:

```
ros2 launch scenario_execution_ros scenario_launch.py scenario:=simulation/scenario_manager/scenarios/case1.os
```

This runs a collision action client, executing multiple collision simulations with varying angles and speeds.

16.5 Time to Collision (TTC) Calculation

The package includes a **TTC node** that logs the **Time to Collision (TTC)** and **Closest Point of Arrival (CPA)** for two robots assuming constant speed and direction.

16.5.1 Running the TTC Node

```
ros2 run scenario_manager ttc
```

This node calculates and logs: - **TTC**: The time at which the closest approach occurs. - **CPA**: The closest distance between the two robots assuming constant speed and trajectory.

Note: As of version **1.0.6 (February 2025)**, the TTC node logs data but does not publish it elsewhere.

16.6 Moveit Panda robot

This package contains code for a robot called “Panda” which includes a description/urdf’s (see `panda_description/`). Inside the `panda_moveit_config/` exists all code and config files that allow you to spawn a panda robot in gazebo and be able to plan and execute motions. Inside `config/` the are several `*.yaml` files which acts as configurations, and many `*.srdf`, `*.urdfs` `*.xacro` files. These are all related to the same panda robot. The main description file is `panda.urdf.xacro`.

To run our package make sure that gazebo simulator is running. Then we can run the `panda_moveit_config/launch/demo.launch.py` which will do the following:

- Spawn a panda robot in gz and publish its `robot_description`.
- Start rviz with a preset moveit config
- Start the `move_group_node` which handles all planning and executing of motions.
- Load all controllers for panda so that it can be controlled in Gazebo

16.7 Moveit motion planner using Python API

The folder `custom_motion_planning_python_api` has scripting files that plans and executes motions for a panda robot. It uses the official moveit2 python API to achieve this. In this package’s `scripts/` folder exists these python files. Look at the demo scripts for inspiration. To run any of the scripts you run the launch file: - `motion_planning_python_api_planning_scene.py` is an original demo from moveit2 - `motion_planning_python_api_tutorial.py` is an original demo from moveit2 - `demo_pick_and_place.py` is custom made.

`moveit2_py` package is not currently available for Humble but Jazzy supports it. The pre-requisites to run a motion planner script is to run these in parallel terminals: `gazebo sim & panda_moveit_config/launch/demo.launch.py`. When these to are finished initializing, run the motion script with: `motion_planning_python_api_tutorial.launch.py`

17 MoveIt Resources

This repository includes various resources (URDFs, meshes, moveit_config packages) needed for MoveIt testing.

GitHub Actions:  

17.1 Included Robots

- PR2
- Fanuc M-10iA
- Franka Emika Panda

17.2 Notes

The benchmarking resources have been moved to https://github.com/ros-planning/moveit_benchmark_resources.

18 panda_description

Note: This package contains a `panda.urdf` and a newer `panda.urdf.xacro`. The XACRO has been created to support finding package resource files dynamically which is needed for Gazebo. The URDF is still needed by `RobotModelTestUtils` which doesn’t support xacro yet.

The URDF model and meshes contained in this package were copied from the frankaemika `franka_ros` package and adapted for use with `moveit_resources`.

All imported files were released under the Apache-2.0 license.

18.1 MoveIt Resources for testing: Franka Emika Panda

A project-internal moveit configuration for testing in MoveIt.

Use the official `panda_moveit_config` if you actually want to work with the robot!

Dyno-robotics/Infotiv delivery

Setting `self.MODE = "abnormal/normal"` in the main script causes the object movement to deviate from the normal distribution specified above.

To reproduce

```
git checkout ....
# In three separate terminals:
./control.sh clean ; ./control.sh build ; ./control.sh sim
./control.sh move_object
./control.sh camera_dump
```

18.2 Random, In Distribution Objects Movement (normal mode)

Data collection from cameras according to [requirements](#)

- Random `self.objects` are placed within the position range `self.grid_x`, `self.grid_y`: REQ.ID.1
- Random rotation along the z axis.
- You can use `camera_config ID.xacro` for placement of several cameras in the intersection. You can alternatively use `camera_config ID_noise.xacro` to slightly changes the camera settings (REQ.ID.2) as below:
 - at most ± 10 degree (± 0.1) rotation around one of the axis
 - at most ± 10 cm (± 0.1) change in x,y,z coordinates

18.3 Random, Out of Distribution Objects Movement (abnormal mode)

Data collection from cameras according to [requirements](#).

- Addition of other objects (spotlight, support pole, traffic cone): REQ.OD.1, REQ.OD.2.
- With some probability, the objects don't leave the scene and may coexist and collide with new objects: REQ.OD.3.
- Objects are at least rotated by $\pi/4$ (45 degrees, upside-down object): REQ.OD.3, REQ.OD.4.
- The spotlight is tilted by $\pi/6$ (30 degrees): REQ.OD.1.
- Objects are not placed on the ground and instead are dropped from a height of 2-3 meters: REQ.OD.4.

18.3.1 Deterministic, Disentanglement One-parameter Object Movements (one_object_deterministic mode)

- Object: forklift
- Camera: 1
- (#Parameter: orientation of forklift)
- Annotation: Position and rotation of the forklift. Annotation in filename.
- 256x256 RGB non compression

```
./control.sh sim ./control.sh move_object ./control.sh camera_dump
```

19 Humanoid Motion Capture

This project develops a system for translating human pose detection to humanoid robot motion in simulation environments. Using Google MediaPipe for pose landmark detection from camera input, the system maps detected human poses to corresponding joint movements executed by a humanoid robot in Gazebo simulator. The implementation leverages ROS2 Ignition and MoveIt2 for motion planning and control, with a data generation pipeline that creates training pairs of pose landmarks and robot joint configurations. This approach provides a foundation for safety monitoring applications in industrial simulation (SIMLAN), where human pose analysis can be integrated for workplace incident detection. The work is based on master thesis "Human Motion Replay on a Simulated Humanoid Robot Using Pose Estimation" by Tove Casparsson and Siyu Yi, Supervised by Hamid Ebadi, June 2025.

Pose translation

19.1 Terminology:

- **Forward Kinematics (FK)**: The process of calculating the position and orientation of a robot's links given the values of its joint parameters (e.g., angles or displacements). In other words, FK answers the question: "Where is the robot's hand if I know all the joint values? (usually have one answer)"
- **Inverse Kinematics (IK)**: The process of determining the joint parameters (e.g., angles or displacements) required to achieve a desired position and orientation of the robot's links. In other words, IK answers the question: "What joint values will place the robot's hand here? (usually have many answers)"
- **Pose** : 3D pose landmarks (MediaPipe) extracted by Mediapipe from an 2D image of human posture
- **Motion** : A kinematic instruction (joint parameters) sent to the robot (via MoveIt) for execution. While "kinematic instruction" would be a more accurate term, we continue to use "motion" for historical reasons (used in the word motion-capture), even though , "motion" often refers to the difference/movement between two postures (e.g., posture2 - posture1).
- **Motion Capture**: Here it means using 2D images to find the motion(kinematic instruction/joint parameters) to instruct a humanoid robot to mimic the human posture.

19.2 Dataset

To find the implementation of how the dataset is created, go to [pre_processing/ directory](#). We have two dataformats for multi and single camera prediction.

For the single-camera dataset, each row in the CSV file represents **one pose** sample and its corresponding robot motion from a specific camera. - The first columns are the 3D coordinates (x, y, z) for each of the 33 MediaPipe pose landmarks, named like `cam500_0_x`, `cam500_0_y`, `cam500_0_z`, ... - The remaining columns are the robot joint positions (motion targets) for that sample, with names like `jRightShoulder_rotx`, `jLeftElbow_roty`, etc.

For the multi-camera dataset, each row in the CSV file represents **multiple poses** samples **from each camera** and its corresponding robot motion. - the first columns are the 3D coordinates (x, y, z) for each of the 33 MediaPipe pose landmarks and **for each camera**, named like `cam500_0_x`, `cam501_0_x`, `cam502_0_x`, `cam503_0_x`, `cam500_0_y`, `cam501_0_y`, `cam502_0_y`, `cam503_0_y`, `cam500_0_z`, `cam501_0_z`, `cam502_0_z`, `cam503_0_z`, ... - The remaining columns are the robot joint positions (motion targets) for that sample, with names like `jRightShoulder_rotx`, `jLeftElbow_roty`, etc.

19.2.1 Translation of a detected pose to humanoid motion command

This project employs a deep neural network to learn the mapping between human pose and humanoid robot motion. The model takes 33 MediaPipe pose landmarks as input and predicts corresponding robot joint positions.

19.2.2 Neural network design

- Input Layer: 33 MediaPipe pose landmarks for each camera (x, y, z coordinates).
- Hidden Layers: Multi-layer perceptron with pose normalization preprocessing
- Output Layer: Robot joint position sequences for humanoid motion control, this results in a total of 47 joints to be outputed.
- Training: Supervised learning on pose-motion paired datasets.

19.3 Project Structure:

- `humanoid_ml_requirements.txt`: Contains the humanoid machine learning requirements
- `input/` : This folder has the pose data to be predicted
- `output/` : This folder saves the predicted motion data and intermediate results
- `/DATASET_RAW/TRAIN`: Contains motion, pose and image files generated by `./control.sh dataset TRAIN`
 - `motion_data` : Corresponding random motion (request)
 - `camera/pose_data` : Mediapipe pose (result) for each camera
 - `camera/pose_images`: Mediapipe annotated images and original images for each camera
- `/DATASET_RAW/EVAL`: Contains motion, pose and image files generated by `./control.sh dataset EVAL`
 - `motion_data` : Corresponding random motion (request)
 - `camera/pose_data` : Mediapipe pose (result) for each camera
 - `camera/pose_images`: Mediapipe annotated images and original images for each camera
- `pre_processing/`: contains all pre-processing files, used to convert the JSON data into tabular data. This will be the input of the model.
 - `mp_detection.py`: Handles pose detection from actual images or videos using MediaPipe

- `load_json_data.py`: Build a csv file that contains both pose and motion data from the json files and outputs a default csv format.
- `csvtowide.py`: Convert the default csv format into single and -multi camera csv files. (this data is the actual model input).
- `pose_to_motion/`: Contains the ML training pipeline for pose-to-motion prediction
 - `autogluon/`: directory containing model implementation for autogluon model. Handles training, evaluation, and predicting motions.
 - * `model.py`: Main file, contains the implementation for train, evaluation, and prediction.
 - * `utils.py`: Utils file, contains helper methods, and model definition. The `model.py` use this.
 - * `output/`: Contain reports of ran sessions and saved model states. Saved models will be found here
 - `saved_model_states/`: Saved autogluon models are stored here
 - `reports/`: Manually generated reports file containing training/evaluation run information. Good for reproducibility. The reports are generated by `generate_report.py` and the new report row is saved in `train_report.csv` or `eval_report.csv`
 - `pytorch/`: directory containing model implementation for pytorch model. Handles training, evaluation, and predicting motions.
 - * `model.py`: Main file, contains the implementation for train, evaluation, and prediction.
 - * `utils.py`: Utils file, contains helper methods, and model definition. The `model.py` use this.
 - * `output/`: Contain reports of ran sessions and saved model states. Saved models will be found here
 - `saved_model_states/`: Saved autogluon models are stored here
 - `reports/`: Manually generated reports file containing training/evaluation run information. Good for reproducibility. The reports are generated by `generate_report.py` and the new report row is saved in `train_report.csv` or `eval_report.csv`
 - `optuna_studies/`: Specific for pytorch. Optuna studies are stored at this folder. See [optuna section](#) for more information about Optuna.
- `generate_report.py`: Contains methods to generate report files, used after a training or evaluation session has been done for any selected model. These generated reports contain information for reproducing runs and how a model performed against metrics. A new report is generated every time we run train or eval on a model and every report is saved in a single CSV file where every row is a report. The files are named: `train_report.csv` or `eval_report.csv`.
- `metrics.py`: Contains all functions for calculating metrics for models for any selected model.
- `humanoid_config.py`: it contains the humanoid configuration like the number of joints and the number of pose landmarks.

19.4 Dataset generation

The below image describes how the dataset generation system works.

Dataset generation overview

To create a humanoid dataset (paired pose data, motion data and reference images) in the `DATASET_RAW/TRAIN` directory:

```
./control.sh dataset TRAIN/
```

19.5 Dataset preprocessing

The diagram below shows the overall workflow of this project. Pose and motion data are first collected from JSON files and combined into a single CSV file. These processed csv files are ready to be fed into a model of your choice, Autogluon or pytorch. See below to find more information about the dataset or more about the models.

Overall workflow

The dataset, input and outputs are in `humanoid_utility/DATASET_RAW/` directory.

The resulting files are stored as parallel `.json` files in multi-camera folders `camera_*/pose_data`, `camera_*/motion_data`, `camera_*/pose_images`. To avoid creation of `pose_images` that are only used as the ground truth and debugging (specially if you are building your training data), comment out then call to `self.save_pose_image()` in `camera_viewer.py`.

Finally to merge them all in tabular `.csv` file run the following command.

```
./control.sh convert2csv
```

Results in merging:

- DATASET_PROCESSED/train.csv to DATASET_PROCESSED/TRAIN/single_train_cam501.csv
- DATASET_PROCESSED/train.csv to DATASET_PROCESSED/TRAIN/single_train_cam502.csv
- DATASET_PROCESSED/train.csv to DATASET_PROCESSED/TRAIN/single_train_cam500.csv
- DATASET_PROCESSED/train.csv to DATASET_PROCESSED/TRAIN/single_train_cam503.csv
- DATASET_PROCESSED/train.csv to DATASET_PROCESSED/TRAIN/multi_train.csv
- DATASET_PROCESSED/eval.csv to DATASET_PROCESSED/EVAL/single_eval_cam500.csv
- DATASET_PROCESSED/eval.csv to DATASET_PROCESSED/EVAL/single_eval_cam501.csv
- DATASET_PROCESSED/eval.csv to DATASET_PROCESSED/EVAL/single_eval_cam502.csv
- DATASET_PROCESSED/eval.csv to DATASET_PROCESSED/EVAL/single_eval_cam503.csv
- DATASET_PROCESSED/eval.csv to DATASET_PROCESSED/EVAL/multi_eval.csv

You can replay_motion each motion data separately: `./control.sh replay_motion DATASET_PROCESSED/EVAL/motion_data/AAAA`

19.6 Model Training

19.6.1 Autogluon

We use [Autogluon tabular predictor](#). This model is trained as an ensemble (collection) of models where each separate model aims to predict a single joint given the complete input poses. The result from each model is then merged together and becomes a predicted list for each target joint.

19.6.2 Pytorch

We have more control over the Pytorch model. It has its own implementation of its dataset and model definition, located inside of [its utils dir](#) folder. What is specific about using the pytorch is the possibility to use optuna as a hyperparameter optimization tool. This tools help find the best suitable set of hyperparams given its training data. Autogluon has its own internal optimization thus we only use this for pytorch.

19.7 Optuna

Optuna is a hyperparameter optimization framework available as a python library. It acts as a wrapper around an existing training framework where it uses the same setup of dataset, train_loop and hyperparameters, so it has the same context as when you would run it yourself. Optuna starts by defining an objective function. The goal of the objective function is either to minimize or maximize a value. You specify this value yourself, for this setup we have typically used the validation MSE to use as a goal to minimize. We also need to define the scope of values for the hyperparameters. We define for every hyperparameter, a range of values which Optuna is allowed to select from. i.e. BATCHSIZE being in range(4,128). When we have done this for every param we want to find the optimal value for, we start the start optuna by running a “study”. Optuna runs N number of training sessions, each independent and uses internal algorithms to find the best set of values for the list of hyperparameters. In the end we get the training session that scored the best and can use its hyperparameters.

Note: We only use this for pytorch since autogluon has its own hyperparameter optimization. This is why we only see studies inside `pytorch/optuna_studies/`.

19.7.1 Commands

Keep in mind that we use a separate virtual environment to install machine learning related pip packages called `mlenv` with a separate [requirements.txt](#). Build the environment first using `./control.sh mlenv`

Running `control.sh single_train` or `control.sh multi_train` in the terminal runs the training pipeline. Single train expects data from 1 camera source whereas multi train can use unlimited amount of camera input. Both models use 1 target set of joints.

The interface to modify the runtime arguments, you modify variables inside of `config.sh`. The runtime variables you can change are these: - **model_type**. Possible selections: pytorch, autogluon - **pose_to_motion_model_name**. If you want to reuse a model, specify its name here. Keep blank if you dont want to save. - **pose_to_motion_session_name**. Optional if you want to describe your session.

After a training run is complete, the model is saved inside of `pose_to_motion/${model_type}/output/saved_model_states`. A summary report of the session is also generated and will be saved inside `pose_to_motion/${model_type}/output/train_report.csv`

There are two options for using the data and training a model. The resulting model is saved in `${model_type}/output/saved_model_states`. Training a model using 1 camera source, reads data from `DATASET_PROCESSED/TRAIN/single_train_cam5XX.csv` and trains our model.

```
./control.sh single_train
```

Training a model using multiple camera sources, reads data from `DATASET_PROCESSED/TRAIN/multi_train.csv` and trains our model:

```
./control.sh multi_train
```

19.8 Model Evaluation

Running `control.sh eval` in the terminal runs the evaluation pipeline. This will trigger the selected `model_type`'s evaluation pipeline. The same run time variables found in `config.sh`, used for training is used for evaluation where the key variables `model_type` and `pose_to_motion_model_name`.

The evaluation will run metrics defined inside of `metrics.py`, currently MSE and MAE will be calculated. The results is then saved in a summary report located in `pose_to_motion/${model_type}/outputs/eval_report.csv`.

To evaluate a trained model, there is a command that evaluates based on a set of metric:

```
./control.sh eval
```

19.9 Model Prediction

Running `control.sh predict` will run input all poses for each camera, in JSON format and for each `pose_data`, output a predicted motion, which is replayable in RViz along with a ground truth `pose_image`.

The following command uses MediaPipe to create pose data from an image or a video in `input/` to populate `output/` with mediapipe landmark pose and images.

```
./control.sh image_pipeline
./control.sh video_pipeline
```

To use the model on each generated pose, run the following command to generate predicted motions which will be saved inside `predict_data/`, inside of the input folder.

```
./control.sh single_prediction output
```

To use the model on each pose inside of any `DATASET_RAW` directory for example: `DATASET_RAW/EVAL/camera_500/pose_data`, run the following command below for single camera using the `/EVAL` dataset and camera 500 as input:

```
./control.sh single_prediction DATASET_RAW/EVAL
```

This will save the predicted motions to `DATASET_RAW/EVAL/predict_data`. Afterwards it opens up a image viewer with the ground truth mediapipe detection as well to quickly check the performance of our ml model:

To use it for multi camera use the command below to predict motions. This works the same as `single_prediction` but instead use multi-camera datasets.

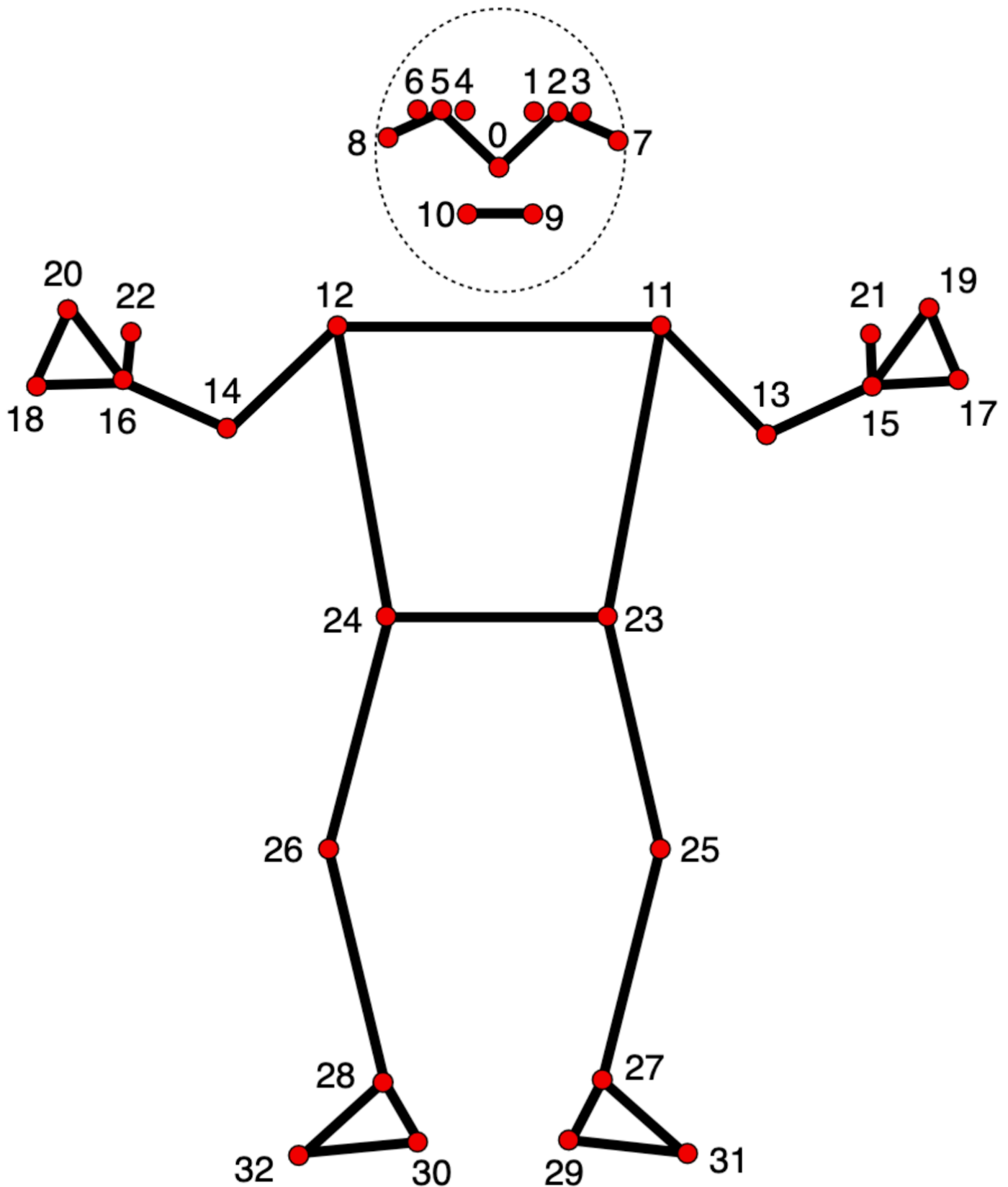
```
./control.sh multi_prediction DATASET_RAW/EVAL
```

19.10 Misc. (IGNORE WHAT COMES NEXT)

19.10.1 Notes

- It is a “feature” (not a bug) that some cameras cannot detect the pose (NaN values). It helps to be still able to detect pose when a part of body is masked.
- Mediapipe cannot reliable detect the depth, back and front
- Experiment different options for NaN values (average of other rows, zero, etc)

20 Preprocessing



https://ai.google.dev/edge/mediapipe/solutions/vision/pose_landmarker

Three steps of normalization (feature engineer) of mediapipe landmarks: - Position normalization - Scale normalization - Rotation normalization

20.1 Issues and Future Works

20.1.1 Current Performance Metrics

The upgraded multi-camera system demonstrates the following evaluation results: - **Mean Squared Error (MSE)**: 0.206
- **Root Mean Squared Error (RMSE)**: 0.454

20.1.2 Known Limitations

- MediaPipe exhibits difficulty distinguishing between front and back orientations, leading to potential left/right body side confusion in pose detection
- Current preprocessing includes normalization for single-camera data (using hip midpoint as origin), but multi-camera normalization requires further development
- Depth estimation reliability remains limited in MediaPipe’s 2D-to-3D pose conversion

20.1.3 Future Development Areas

- Enhance multi-camera data preprocessing with normalization, also probably replace the `pose_landmarks` into `pose_world_landmarks` in `camera_viewer.py`
- Replace pre-processing with pose-processing, and pay attention to whether Mediapipe’s `static mode` is enabled.
- Consider applying feature engineering to enable the model to gain a deeper and more accurate understanding of the data.

20.2 The Theory

-
- I : an image. (I_s : from simulator, I_r from real world)
- P : a pose (mediapipe output)
- M : a motion (moveit2)

then - $SIM(M) \rightarrow I$: Simulator(gazebo) using inverse kinematics(moveit2) to convert the motion M to create image I - $PE(I) \rightarrow P$: Pose estimator(mediapipe) takes the image I , to find human pose P - $Q(P) \rightarrow M$: machine learning model Q , takes pose P and tries to replicate that pose estimator(mediapipe) pose using motion (moveit) M

Assumption: Pose estimator(mediapipe) performs good enough that it can detect human poses from both simulator and real-world domains: - $PE(I_r) \rightarrow P_r$ - $PE(I_s) \rightarrow P_s$

Our goal is to - pass random M to build the dataset of pairs : $\langle M, PE(SIM(M)) \rangle = \langle M, P \rangle$ - Use the dataset above to find $Q()$ which is the inverse of this $PE(SIM())$

No we can do motion capture (replicate real human movements) by $Q(PE(I_r))$

21 Autogluon Model configuration

This section describes how AutoGluon is configured for a multi-label regression problem in our current setup.

To use AutoGluon for multi-label regression problem, first we need to create a MultilabelPredictor by setting - **labels**: the labels that we want to predict. - **problem_types**: the problem type for each TabularPredictor. - **path**: path to directory where models and intermediate outputs should be saved. - **consider_labels_correlation**: Whether the predictions of multiple labels should account for label correlations or predict each label independently of the others.

For **consider_labels_correlation**, we set it to `FALSE` in order to disable using one label as feature for another. The reasons for this is: - Each joint has its own degree of freedom and control and it is independent from the rest of joints - Training stability without error propagation - When set to `False`, the training will be faster - more general and simple model

For training the model, we should take into consideration the following hyperparameters related to : - **Stacking**: which is an ensemble technique where multiple models are trained and then a “meta-model” learns how to combine their predictions. If we use dynamic Stacking, it will automatically determines the optimal number of stacking layers and which models to include. - **Bagging**: multiple training versions of the same model on different subsets of data and combining their predictions. - **preset**: which condense the complex hyperparameter setups. For example, We can use a small model size by using `medium_quality` which will lead to a faster training but with less prediction quality. Or we can use large model by setting `preset` to `best_quality` which will lead to a better performance but much longer training time.

The current autogluon uses the following hyperparameters: - **dynamic_stacking=False**: Disables the automatic stacking optimization. - **num_stack_levels=0**: no stacking level. - **auto_stack=False**: Disables automatic ensemble stacking. - **num_bag_folds = 0** and **num_bag_sets = 1** which meaning no bagging.

So here is the current training flow for each joint : - Train NN_TORCH, GBM and XGB models - No bagging: Each model trains on full dataset once, no multiple training versions of the same model on different subsets of data and combining their predictions. - No stacking: No meta-models combining predictions - Best model selection: Choose best performing model per joint

Eliminating Dynamic stacking and multi-level stacking will lead to 50% to 70% time saving.

Overall, this configuration prioritizes training speed and stability over ensemble complexity, making it suitable for fast iteration and independent joint predictions.

22 Hyperparameter Tuning

In The current AutoGluon model, we are training NN_TORCH, GBM and XGB models by using their default built-in hyperparameter settings. The next step for performance improvement is to add the **hyperparameter_tune_kwargs** argument to enable AutoGluon's internal hyperparameter optimization. This will allow AutoGluon to automatically search for the best hyperparameters for each model type, balancing training time and prediction quality.

22.1 gazebo_ros2_control

```
[gzserver-1] [ERROR] [1727951819.671190014] [jackal.gazebo_ros2_control]: controller manager doesn't have an u
No solution
```

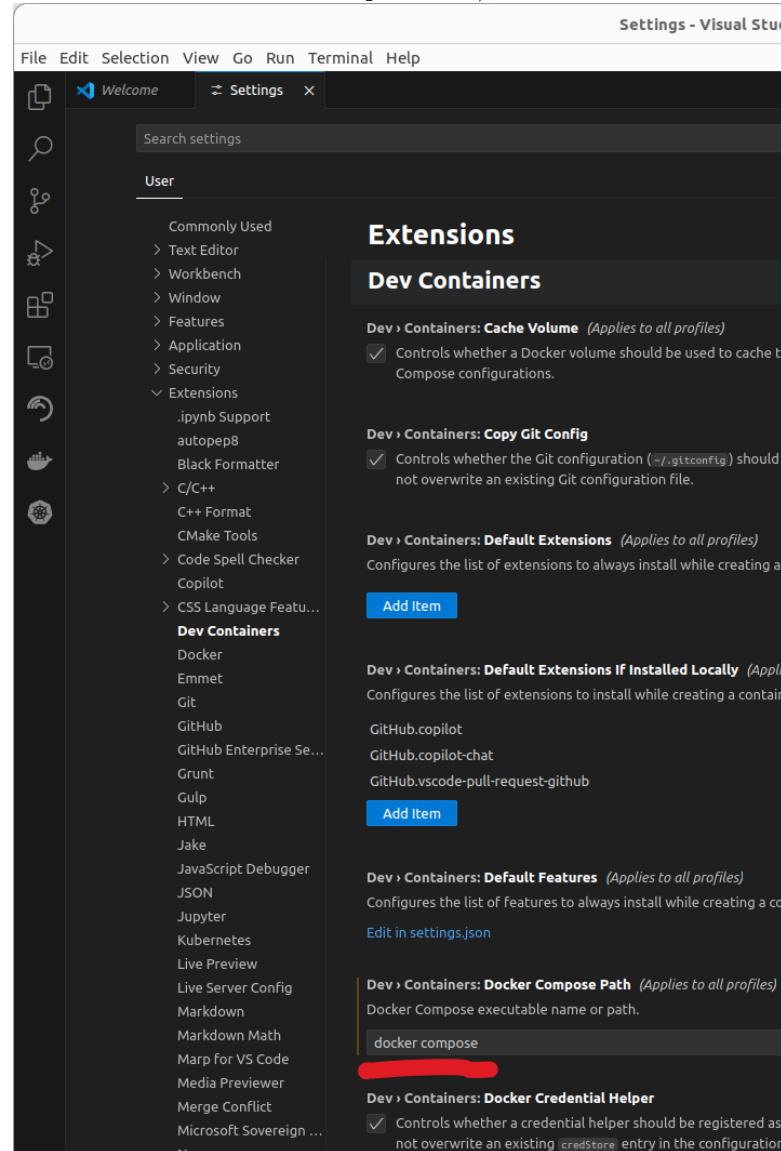
22.2 OpenAL

```
[gzserver-1] [Err] [OpenAL.cc:84] Unable to open audio device[default]
```

Related to support for audio inside docker container. It will not be resolved

22.3 Command failed: docker compose

If you have any issue with docker incompatibility (e.g. Error: Command failed: docker compose ...), make sure that



`docker compose` or `docker-compose` is set correctly in the setting.

In `docker-compose.yaml`, uncomment the `factory_simulation_nvidia` section:

```
factory_simulation_nvidia:
  <<: *research-base
  container_name: factory_simulation_nvidia
  runtime: nvidia
  deploy:
    resources:
      reservations:
        devices:
          - driver: nvidia
            count: "all"
            capabilities: [compute,utility,graphics,display]
```

and update `.devcontainer/devcontainer.json`:

```
{
  "name": "ROS2 RESEARCH CONTAINER",
  "dockerComposeFile": "../docker-compose.yaml",
  "service": "factory_simulation_nvidia"
  ...
}
```

22.4 Missing nvidia docker runtime

Solution: <https://stackoverflow.com/questions/59008295/add-nvidia-runtime-to-docker-runtimes>

22.5 Docker nvidia-container-cli: requirement error

If you get this error when building the docker container:

nvidia-container-cli: requirement error: unsatisfied condition: cuda>=12.6, please update your driver to a new

A solution to try is first: re-install nvidia-container-toolkit. Or if that does not work, update your nvidia-drivers.

22.6 GLIBC_2 issue

Sometimes the wrong nvidia-container-toolkit results this issue below:

```
$/usr/lib/x86_64-linux-gnu/libc.so.6: version `GLIBC_2.38' not found (required by /usr/lib/x86_64-linux-gnu/libc.so.6)
$/usr/lib/x86_64-linux-gnu/libc.so.6: version `GLIBC_2.38' not found (required by /usr/lib/x86_64-linux-gnu/libc.so.6)
```

If so, try downgrading nvidia-container-toolkit. You can use these commands:

```
$sudo apt-get remove --purge nvidia-container-toolkit nvidia-container-toolkit-base libnvidia-container*
$sudo apt-get update
$sudo apt-get install nvidia-container-toolkit=1.17.4-1 nvidia-container-toolkit-base=1.17.4-1 libnvidia-container*
```

22.7 docker issues

Make sure docker is installed correctly by following these two instructions:

- [Install docker using the convenience script](#)
- [Linux post-installation steps for Docker Engine](#)

22.8 Advanced options:

To record camera images for available cameras we use a simple python code `./camera_utility/camera_subscriber.py` that continuously record camera images in `camera_utility/camera_data/` :

```
./control.sh cam_dump
```

22.9 Advanced features

22.9.1 Running ROS2 commands:

To avoid conflict between ros nodes in the same network, after each build a new random ROS2 domain is created. This means that you need to adjust this random domain before running any ros2 commands. For convenience you can use

```
./control.sh cmd YOUR_ROS2_COMMANDS
```

Here is an example of extracting tf2 hierarchy

```
./control.sh cmd ros2 run tf2_tools view_frames
```

These features and commands are under development and not fully supported yet and therefore are subject to change.

Cartographer: With both Gazebo and rviz running you can start creating a map of a robots surroundings

To start: `./control.sh cartographer`

The jackal can then be controlled with the computer keyboard by running:

```
./control.sh teleop_jackal
```

To record one screenshot after use:

```
./control.sh screenshot 164
```

The result will be stored in `./camera_utility/camera_data/`.

To record ros messages in ROS bag files to replay the scenario later:

```
./control.sh ros_record
```

To replay the last rosbag recording:

```
./control.sh ros_replay
```

To test the unit tests before pushing new codes:

```
./control.sh test
```

22.10 Jackal

If you want to control the Jackal you add the following lines into the control.sh:

```
elif [[ "$*" == *"jackal_teleop"* ]]
then
    ros2 launch dyno_jackal_bringup keyboard_steering.launch.py
```

And then run:

```
./control.sh jackal_teleop
```

These ones did not work so we put it here in issues

```
elif [[ "$*" == *"move_object"* ]]
then
    ros2 run object_mover move_object
```

```
elif [[ "$*" == *"scenario"* ]]
then
    ros2 launch scenario_execution_ros scenario_launch.py scenario:=simulation/scenario_manager/scenarios/test
```

22.10.1 high-level_diagram_SIMLAN.drawio

“high-level_diagram_SIMLAN.drawio.png” ### Humanoid_mocap_flow.drawio “Humanoid_mocap_flow.drawio.png”
legend.drawio “legend.drawio.png” ### ros2 launch aruco_localization_multi_detection.launch.py.drawio
“ros2 launch aruco_localization_multi_detection.launch.py.drawio.png” ### ros2 launch camera_bird_eye_view
bird_eye_view.launch.py.drawio “ros2 launch camera_bird_eye_view_bird_eye_view.launch.py.drawio.png” ### ros2
launch moveit_resources_panda_moveit_config_demo.launch.py.drawio “ros2 launch moveit_resources_panda_moveit_config
demo.launch.py.drawio.png” ### ros2 launch pallet_truck_bringup_multiple_robot_spawn.launch.py.drawio “ros2
launch pallet_truck_bringup_multiple_robot_spawn.launch.py.drawio.png” ### ros2 launch pallet_truck_navigation
map_server.launch.py.drawio “ros2 launch pallet_truck_navigation_map_server.launch.py.drawio.png” ### ros2 launch
pallet_truck_navigation_nav2.launch.py.drawio “ros2 launch pallet_truck_navigation_nav2.launch.py.drawio.png” ###
ros2 launch simlan_bringup_sim.launch.py.drawio “ros2 launch simlan_bringup_sim.launch.py.drawio.png” ### ros2 launch
static_agent_launcher static-agent.launch.py.drawio “ros2 launch static_agent_launcher static-agent.launch.py.drawio.png”
ros2 launch visualize_real_data_scenario_replayer.launch.py.drawio “ros2 launch visualize_real_data sce-
nario_replayer.launch.py.drawio.png” ### SIMLAN_DIAGRAM.drawio “SIMLAN_DIAGRAM.drawio.png”

22.11 Licenses and Credits

The majority of code for the pallet truck comes from these [turtlebot3](#) and [turtlebot3_simulations](#) repositories (in [humble-devel](#) branch) with [Apache-2.0 license](#) and we continue using the same license: The Turtlebot3 robot project belongs to [robotis.com](#) and accessible in [git repository](#). The authors and maintainers of the original packages that all credits go to are: - [Darby Lim](#) - [Pyo](#) - [Will Son](#) - Ryan Shim In October 2023, [Hamid Ebadi](#) renamed the [package owner information and name](#) for turtlebot3 projects to avoid dependency issues any naming confusion with the original packages and created independent packages for activities within the research project. We also got inspired and used the skeleton code from these open source project and courses: - https://github.com/ros-controls/gazebo_ros2_control - https://github.com/renan028/forklift_robot - <https://github.com/ROBOTIS-GIT/> - <http://turtlebot3.robotis.com> - [Articulated Robotics](#) - “[ROS2 for Beginners Level 2 - TF | URDF | RViz | Gazebo](#)” Udemy course - “[ROS2 Nav2 \[Navigation 2 Stack\] - with SLAM and Navigation](#)” Udemy course - [Visual Servoing in Gazebo grobot](#) - The modification of [Human-Gazebo](#) in [/src/humanoid_robot/model/human-gazebo](#) is licensed under LGPL-2.1. - [Moveit2](#) is licensed under BSD-3-Clause license. ## Resources - [Gazebo official video playlist](#) - [Getting Ready to Build Robots with ROS](#) - [Udemy course on ROS2 and Gazebo](#) - [classcentral course](#) - [theconstructsim course](#) - [Articulated Robotics](#) - [ROS URDF](#) - [FoxGlove studio](#)

(rviz alternative) - ROS2 Tutorials - aws-robomaker - model for factory - Tugbot in Warehouse - logistics in warehouse - aws-robomaker - gazebo_worlds - warehouse - forklift - Docker install guide - Hazard stripes taken from Tugbot warehouse - Deadlock image

Other solutions: - Kollmorgen: How does an AGV navigate? - SwissLog CarryPick - Toyota forklifts - ILIAD Project - GoPal - navigation_oru navigation stack by Örebro University

Specification of items: - SLAM Navigation Compact Pallet Mover Nature Navigation Mini Forklift with Payload 1000KG - Driverless Lifting System: Single & Double Scissor Lift | AGILOX - Wholesale Pallet Agv Trucks Jack Automated Autonomous Forklift - Volvo modular containers - Volvo Emballage Specifications Volvo Group Packaging System - Freecad - Blender - example worlds - SDF format - SDF tutorial - FreeCAD RobotCreator Workbench

Coordinates: - odom

Camera projection: - <https://github.com/polygon-software/python-visual-odometry/blob/master/Chapter%20%20-%20Camera%20Projection.ipynb> - https://classic.gazebosim.org/tutorials?tut=camera_distortion - <https://learnopencv.com/rotation-matrix-to-euler-angles/> - <https://www.geeksforgeeks.org/calibratecamera-opencv-in-python/> - <https://docs.opencv.org/4.x/dc/dbb/t> - http://sdformat.org/tutorials?tut=specify_pose

Jackal Robot: The Clearpath Jackal Robot code is forked by Dyno Robotics from Clearpaths [jackal](#) Github. Branch foxy-devel is ported to dyno_humble, where changes for namespacing and a bringup package is added. - To improve collaboration in development environment we use vscode and docker as explained in [this instruction](#) using these [docker files](#). For production environment follow installation procedure used in [.devcontainer/Dockerfile](#) to install dependencies. - The Docker setup is added by Christoffer Johansson (Dyno Robotics), based on Dyno experience working with Docker. ## Licenses Imu_tools is the one with several licenses, using BSD-3, GPLv3, GNU v3, but as we don't change the code so we believe there is no license conflict with the current project license. The dependency wireless (<https://github.com/clearpathrobotics/wireless.git>) from Clearpath robotics is the without a separate license file. There are license names within files, referencing BSD so we believe there is no license conflict with the current project license. ## Docker and VS Code setup The Docker setup added by [Christoffer Johansson \(Dyno Robotics\)](#), based on Dyno experience working with Docker. ## Project maintainer This project is currently maintained by [Hamid Ebadi](#).