Ref. Ares(2022)2422330 - 31/03/2022

This project is funded by the European Union
G.A No 101016906

Project Number: 101016906
Start Date of Project: 2021/01/01
Duration: 48 months

Type of document D5.4 – V1

VR Mobile-Robot Component

Dissemination level	PUBLIC
Submission Date	2022-03-31
Work Package	WP5
Task	T5:4
Type	Report
Version	1
Author	PALEBLUE
Approved by	KTH, UNICLAM, UNIROMA3

DISCLAIMER:
The sole responsibility for the content of this deliverable rests with the authors. It does not necessarily reflect the opinion of the European Union. Neither the Directorate-General for Communications Networks, Content and Technology, Artificial Intelligence and Digital Industry nor the European Commission are responsible for any use that may be made of the information contained therein.

This project is funded by the European Union
G.A No 101016906

Executive Summary

This document describes the functionality of a standalone robot. The way the model is created based on an URDF is described. The physics implementation and interaction with forces of the robot with its environment is detailed. The available ROS interfaces available to control and monitor the robot are listed and the possibility to use the sensors to test human safety is discussed.

This project is funded by the European Union G.A No 101016906

Table of Content

1 Introduction 5
2 URDF 6
2.1 Collisions 6
2.2 Runtime URDF update 7
3 Kinematic and dynamic model 8
3.1 Joints 8
3.2 Physics: force application and feedback 10
3.3 Hardware interface simulation 11
3.3.1 Effort interface 12
3.3.2 Velocity interface 12
3.3.3 Positional interface 12
3.3.4 Comparison to the real robot 13
4 ROS interfaces 14
4.1 Mobile base controller and state 14
4.2 TF2 Publisher 15
4.3 Dual-arm Joint State Publisher 15
4.4 Dual-arm Joint Trajectory Controller 15
4.5 Dual-arm Joint Group Velocity Controller 16
5 Sensors 17
5.1 Human simulation for safety validation 17
6 A grape-vine 19
7 References 20

This project is funded by the European Union G.A No 101016906

Abbreviations and Acronyms

GPS	Global Positioning System
GPS-RTK	GPS with real time kinematics
IMU	Inertial measurement unit
PID controller	proportional, integral, and derivative controller
RGB camera	Red, green, blue. Also: colour camera
RGB-D camera	Red, green, blue and depth camera
ROS	Robot Operating System
TF2	Transform v2
URDF	Unified Robot Description Format
WP	Work Package
Xacro	XML Macros
XML	Extensible Markup Language

This project is funded by the European Union
G.A No 101016906

1 Introduction

In this document the simulation of the single robot dual arm will be described. This robot simulation will be used to test the manipulation of the vines, the pruning of branches and harvesting of the grape bunches. The simulation will be used to design and test physical and safety-related robot-human interaction algorithms.

The dual-arm will host a number of sensors which enable the perception of the environment. This includes the recognition of the grape bunches and the peduncles which should be cut to separate the grape bunches from the branches and the perception of humans in the environment.

The mobile base carrying the dual arm robot is described in deliverable [D3.4]. More details on the sensors on the robot are found in deliverable [D4.3].

This project is funded by the European Union G.A No 101016906

2 URDF

The robot in the simulation is mostly generated from an URDF file. Such URDF files are created manually from the provided Xacro files for the robot (Distributed by Pal Robotics). This enables us to change the robot in the simulation easily by re-importing an updated URDF file.

2.1 Collisions

In the Unity 3D engine, collisions are implemented using colliders which are attached to meshes. There are three main types of colliders:

- Mesh colliders;
- Convex mesh colliders;
- Primitive colliders.

Mesh colliders can exactly match the actual visible meshes, but are very computationally intensive and collisions between mesh colliders are not supported, i.e., nothing happens when two mesh colliders collide.

A convex mesh collider cannot have inward bending curves but does support collisions with mesh colliders. Convex mesh colliders may not be able to match the visible mesh. These colliders can use faster algorithms, compared to mesh colliders, enabling collisions between convex mesh colliders. However, the collision check is very computationally intensive. This is given by the fact that every triangle in the mesh collider needs to be checked for collision for the simulation. If there is a convex mesh collider near another convex mesh collider you pay this cost combinatorically: $n*m$ checks need to be performed where n and m are the number of triangles in the two meshes.

Primitive colliders include sphere-, capsule- and box colliders and have low computational overhead and support collisions in all cases.

When importing the robot model, by default, any concave mesh is transformed to a convex mesh collider automatically to enable collisions of the robot with the environment. The resulting convex colliders extend beyond the visible mesh of the robot (see Figure 2). With these collider-meshes unexpected collisions can happen while some poses of the robot are no longer possible because, for example, the arms collide with the extended torso collider.

Because of the above reasons we use primitive colliders (see Figure 1). These colliders are placed manually because the best collider to use for the arms is a capsule collider which is a cylinder with rounded ends (see the arm in Figure 1). This type of primitive collider is not supported in URDF. URDF

does have a cylinder, but these can result in collisions between the links of the arm because the ends are not rounded.

2.2 Runtime URDF update

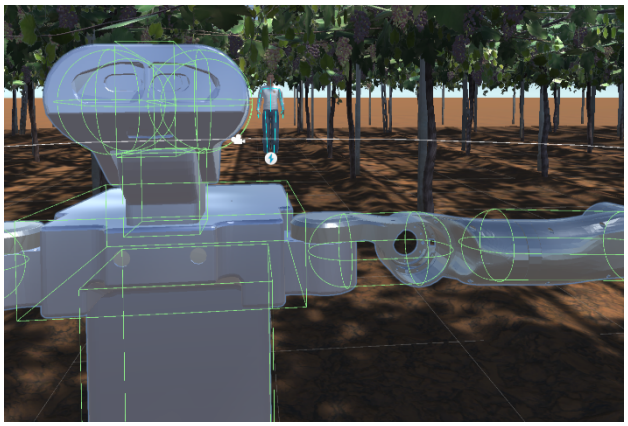

Figure 1 Robot with primitive colliders

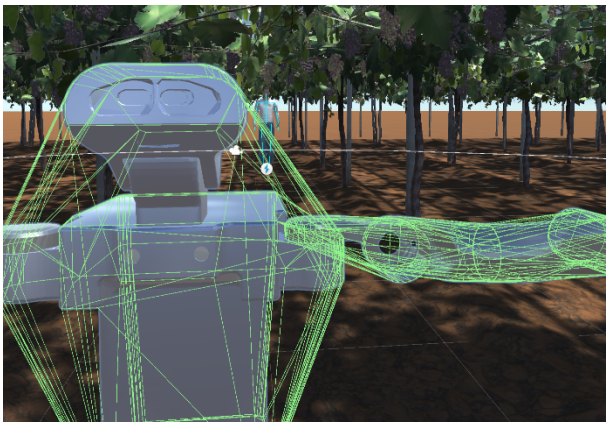

Figure 2 Robot with convex mesh colliders

The simulation starts with a default URDF model which is based on the latest robot configuration as provided in the Xacro format. When the application starts, it will look if an alternative URDF schema is provided. When this is the case, it will parse this URDF before the simulation starts. The robot's configuration will be adapted to this new URDF which makes it possible to change positions of sensors.

The information in the URDF will be used only partially. The fixed parts of the mobile base like the body and the tracks cannot be changed using the URDF. The main reason for this is that the tracks cannot be modelled in the URDF in such a way that they are simulated correctly in the Unity3D engine (see [D3.4]).

The location of the sensors on the mobile base and the dual arm can be changed with the URDF. It is also possible to change the configuration of the dual arm itself like the orientation of the bracelet on the end-effort carrying the RealSense camera.

The kinematic, dynamic and visual model will not be changed at runtime with the URDF. The reason for this is that the simulation and networking needs manual adjustments like the colliders and additions which cannot be done at runtime. Each joint of the robot will be checked to the joint definition in the URDF file. The local position and orientation will be updated from the *origin* information in the joint element. If the *parent* element in the joint is different, the joint will be relocated within the transform hierarchy based on this. This makes it possible, for example, to change the position of a sensor from the mobile base to the top of the robot's torso. Note that the positions and orientations are not checked for realism: it is possible to place object in the free air or within other parts which may lead to unexpected collisions and movements.

It is possible to add one RGB camera to the robot using the URDF. When a joint with *child* link name 'rgb_cam_link' is added to the URDF with specific *origin* and *parent* values, a standard RGB camera is attached to the robot at the given location. This camera will then publish colour images using the */camera0* node.

3 Kinematic and dynamic model

The robot is simulated with full dynamic model based on the kinematic and dynamic properties specified in the URDF.

3.1 Joints

Each link is implemented using an Articulation Body[1] in Unity (see Figure 4) which implements properties for the link and optionally the link drive properties. This includes the transform position and rotation, mass, joint axis, and limits. Various joint types are possible, but for the dual arm we only have fixed, prismatic (linear translation: for the torso lift) and revolute (rotation) joints. The latter two are joints with 1 degree of freedom (See Figure 3).

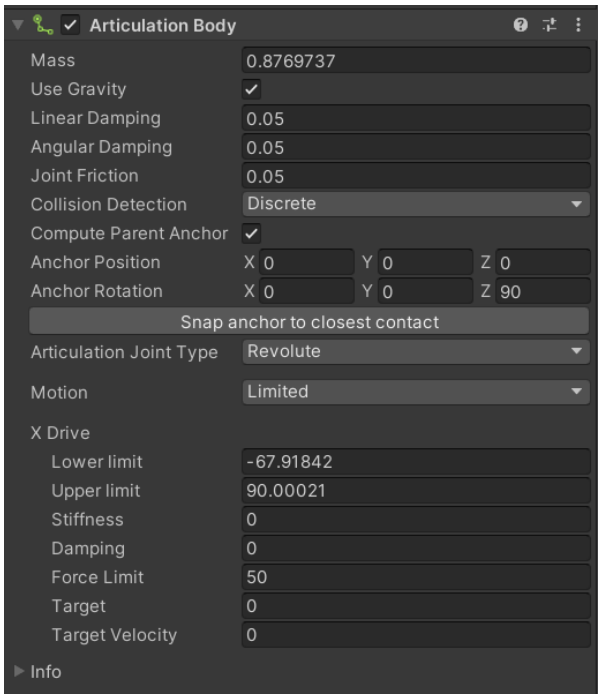

Figure 4 Articulation Body for the left arm 1 link

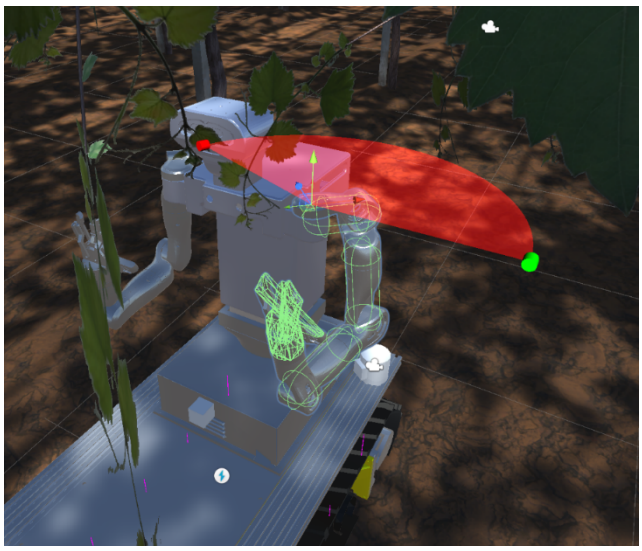

Figure 3 Rotation limitations of the first arm joint

In general, the Articulation Body uses an Articulation Drive for each degree of freedom to simulate the actuator itself. As we have at most one degree of freedom, only an *X Drive* is visible in Figure 4.

[1] https://docs.unity3d.com/2020.1/Documentation/Manual/class-ArticulationBody.html

As an illustration, a spherical joint has three degrees of freedom: 2 swing rotations and one twist rotation (see Figure 5). Spherical joints are not used for the dual arm simulation.



Figure 5 Spherical Joint with 3 degrees of freedom

The inertial data from the URDF file is set separately using a specific URDF Inertial component (see Figure 6).

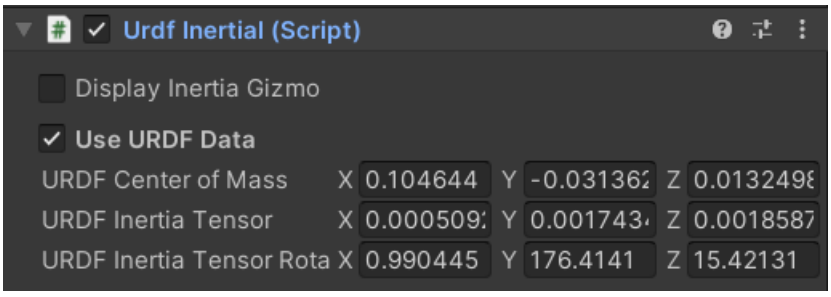

Figure 6 Inertial data for the left arm 1 link

This project is funded by the European Union G.A No 101016906

The only limitation in the simulation is how to set velocity limits which is not directly supported by the Articulation Body. We plan to investigate how this limitation can be included as part of the hardware interface implementation (see Section 3.3).

3.2 Physics: force application and feedback

The Articulation Bodies and Drives are implemented using a dynamic physics model. The joints are controlled using forces (for the prismatic joints) and torques (for the revolute joints). This implementation is an integral part of the whole dynamic physics simulation. This means that the various parts of the robot will collide with static objects like the vineyard support structure and exchange forces with dynamic objects like the grapes, humans, and other robots. It is also possible that the robot collides with itself.

The robot will not be able to destruct itself in any way. Movements which would be potentially self-destructive in the real world will typically result in unstable or unrealistic movement. For instance,

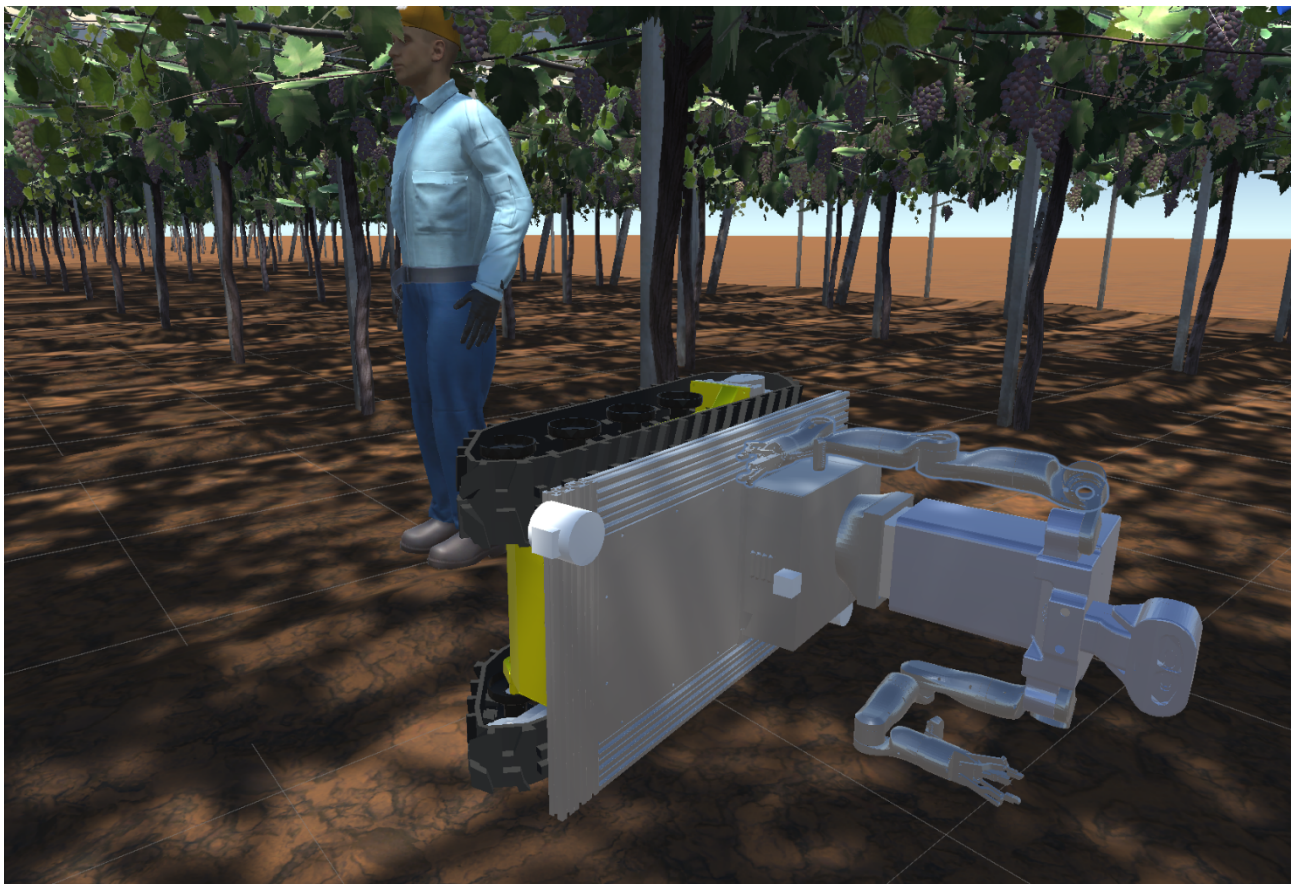

Figure 7 Robot on its side as a result of harmful movements of the arm

the robot could fall over (see Figure 7) or joints may become dislocated. However, in most cases where unbounded movements are requested the movements are stopped by the joint limitations without any consequence. We will investigate whether it is possible to include a safety switch on the robot for the cases where the forces or torques exceed a predefined threshold.

It is possible that a desired position or velocity of a joint cannot be met because of a collision with the environment. In this case the robot will experience a reactive force in each affected joint.

In addition to that, the joints will also experience forces as a result of gravity: the weight of the various parts of the arm and the objects the gripper is holding, like grape bunches, will result in a force in the joints of the robot. This will have an effect in the PID controllers for the joints in the measured positional or velocity error (depending on the interface used). The force feedback values will also be reported as part of the joint state publisher (see Section 4.3). The result may be that the end-effector does not reach the desired position in case the maximum force which can be applied by the joint drive is reached.

The joints themselves also have properties which affect the result of a force applied to a joint. As can be seen in Figure 4, damping and friction can be set to simulate these effects.

Finally, we have the inertia of the robot's parts when moving around resulting in centrifugal and Coriolis forces. The inertia's configuration uses the information in the URDF about the mass and its distribution across the various links of the robot.

These forces are all calculated together, and the resulting force is being reported as part of the joint state. This can then be used to implement a control feedback loop to ensure an optimal control of the robot. How this is used depends on the hardware interface used for the joint simulation.

At the time of writing, this force feedback is not yet available as this requires support in the Unity engine which is not yet released for production. We have tested this using a beta version of Unity (version 2022.1.0 beta 7) and we confirmed that this functionality will indeed be implemented as needed. We will switch to this Unity version as soon as it becomes production ready.

3.3 Hardware interface simulation

The ROS hardware interface implements the way the joints can be controlled. Possible interfaces are:

- Positional interface,
- Velocity interface,
- Effort interface.

All these interfaces are actually simulated by applying torques and forces to the joints as the low-level robot simulation is using the dynamic physics engine. Depending on the interface, an intermediate controller is used to transform these interfaces to the correct forces and torques.

For this purpose, we implemented a Custom Articulation Drive which can implement these interfaces.

3.3.1 Effort interface



Figure 8 Effort Interface for an Articulation Body

This interface forwards the incoming efforts in Newton (N) (for the linear or prismatic joint in the torso) or Newton-meter (for the rotational joints) directly to the Articulation Bodies in the simulated robot (see Figure 8).

3.3.2 Velocity interface

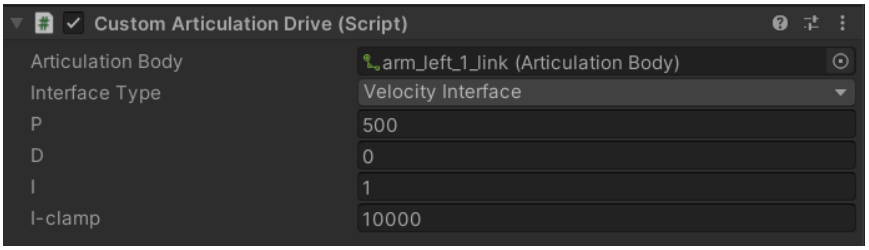

Figure 9 Velocity Interface for an Articulation Body

The current velocity interface simulation (see Figure 9) uses a proportional controller. The difference between the current velocity and the desired velocity (the velocity error) is used to calculate the needed effort to reach the desired drive. The proportional factor is a pre-set value and cannot be changed outside the simulation as we assume that the real joint actuators do not change their properties either. We can make these factors configurable using a configuration file or via ROS parameters if the need for that arises. It is used to simulate the behaviour of the various joints or actuator in the real robot as closely as possible.

The proportional controller is a very basic implementation which gives reasonable results. We chose to do this because the velocity interface of the real robot we are trying to simulate is still in development. When this actual implementation becomes final, we will re-evaluate our simulation to minimise any possible gap between the simulation and the actual movements.

3.3.3 Positional interface

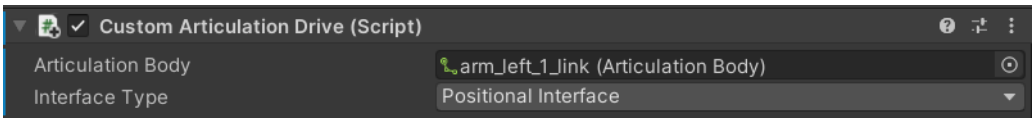

Figure 10 Positional Interface for an Articulation Body

For the implementation of the positional interface a spring model is used. It is implemented using the following formula[2]:

*Effort = stiffness * (jointPosition - targetPosition) - damping * (jointVelocity - targetVelocity)*

where effort is the resulting force or torque applied to the joint. The stiffness and damping values are pre-set and currently cannot be changed outside the simulation, but possibly in the future (see

[2] https://docs.unity3d.com/ScriptReference/ArticulationDrive.html

This project is funded by the European Union G.A No 101016906

the remark for the velocity controller above). They are used to simulate the behaviour of the different joints or actuators in the real robot as closely as possible.

Alternatively, we could use a PID controller instead of the spring model to translate the desired positions and velocities. This is the approach which is taken in Gazebo. When using that implementation, we found that the behaviour of the robot did not match the behaviour observed in Gazebo when using the same values for the PID controller. We hypothesize this is because of differences in the physics model between Gazebo and Unity. We plan to investigate this further such that we can improve our simulation. As the spring model results to a behaviour closer to that observed in Gazebo, we chose to use the spring model for now.

3.3.4 Comparison to the real robot

In simulating the joints with the various interfaces above, we concluded that the implementation of the joint controller itself plays a crucial part in achieving realistic movements. We found that it is not possible to take the Gazebo simulation and port it over to our simulation directly because of the differences in the physics engine used between the two simulators.

Therefore, we plan to do a side-by-side comparison with the real physical robot as soon as it becomes available. This will enable us to tune the simulation under different conditions such that it behaves in the same way.

This project is funded by the European Union G.A No 101016906

4 ROS interfaces

The ROS interfaces are an important part of the simulation as this is the place where the simulation connects to the robot control software which needs to be tested. We strive to match the available interfaces to the interfaces of the real robot as closely as possible. The following interfaces are currently foreseen as detailed below:

- Mobile base controller and state,
- TF2 publisher,
- Dual-arm joint state publisher,
- Dual-arm joint trajectory controller,
- Dual-arm joint velocity controller.

More interfaces may be added later when needed. In that case, these shall be added to this document as an appendix.

4.1 Mobile base controller and state

The mobile base can be controlled using *track* or *twist* messages.

The *track* messages will control the torque of the left- and right tracks independently with a value between -1 and 1 where 1 is maximum speed forward and -1 maximum torque backward. By using different value for the two tracks, the mobile base will rotate.

The *twist* interface is the same as is offered on the real mobile base which enables you to control the forward/backward speed with the linear field and the angular speed is controlled with the angular field.

At current, the mobile base control is open loop. The input values map to torque values applied to the left and right track, but there is no direct feedback loop on the experienced reactive forces or how fast the tracks move as a result of these torques. Next step, if deemed necessary to improve the accuracy of the simulation of the mobile base, would be to introduce the closed-loop control which could be realized by resorting to the same kind of controller interfaces as per the description of the dual-arm.

Apart from the positional and orientation feedback from the GPS and IMU sensors, the actual movements of the tracks of the mobile base are also reported with *odometry* messages. These values are derived from the movements of the tracks themselves. As these tracks are implemented using a dynamic physics model, these movements show feedback from interaction with the ground, including slipping. For this reason, the reported odometry values do not represent the actual movement of the mobile base.

For testing purposes, we also plan to publish a ground truth state of the mobile base position and orientation on the map. This can be used to compare the position and orientation derived from sensory input like GPS, IMU and odometry to the actual position of the simulated mobile base.

4.2 TF2 Publisher

The simulation implements a TF2 publisher which publishes information about the whole transform tree of the robot(s) including the mobile base. The links in these messages are based on the links specified in the URDF of the robot.

For the fixed joints, the links are reported on a */tf_static* topic which should normally not change[3]. All other links are reported on the */tf* topic periodically. These messages will reflect the pose of all movable joints in the robot. The update rate will be 10Hz by default which is the same as the update rate in Gazebo. This frequency can be changed if needed.

4.3 Dual-arm Joint State Publisher

The Joint State Publisher will publish the Joint State for each joint in the dual-arm robot. It publishes the position, velocity, and effort of all joints in the robot. The publish rate of the joint states can be set with a ROS Parameter.

4.4 Dual-arm Joint Trajectory Controller

Like the real robot, the dual-arm provides three Joint Trajectory Controllers:

- Torso controller;
- Arm left controller;
- Arm right controller.

It provides the *command* topic which receives the trajectories to execute and publishes on a *state* topic publishing the desired and actual state of the robot and the error between the two. Currently, the action interface is not supported yet as direct support for action interfaces is lacking the ROS connector we use. We can make a custom implementation for that when desired though. However, we understand that the robot is more likely to be controlled by a velocity controller like the Joint Group Velocity Controller so we focused on that first.

It will be possible to use the trajectory controller with a positional, velocity or effort interface[4]. The positional interface is default. It can be switched by enabling the velocity or effort interface in the simulation settings file. The ROS parameter *joints* for the trajectory controller determines which joints will be controlled by the trajectory controller.

The simulation of the Joint Trajectory Controller is based on the Gazebo simulation of the Tiago++ robot from PAL[5] which forms the base of the Canopies dual-arm robot.

In case of a positional interface, the positions and velocities in the command are directly forwarded to the simulated positional interface.

Like in the Gazebo simulation, when using a velocity interface there are in fact two controllers: one for the simulation of the velocity interface (see Section 3.3.2) and one for the joint trajectory

[3] Normally, these should be published using a latched topic but, because of a bug in the engine we use, it is currently sent out periodically every second.
[4] At the time of writing, the effort interface is not yet available.
[5] https://github.com/pal-robotics/tiago_dual_simulation

This project is funded by the European Union G.A No 101016906

controller. For the velocity interface, a proportional controller is used, for the joint velocity controller, a PID controller is used to calculate the velocity values for the simulated velocity interface based on the desired position and velocity. Like for a regular trajectory controller, the values for the PID controller are set using the *gains* ROS parameters . Additionally, the *velocity_ff* parameter can be used to control the direct feed-forward of the desired velocity to the velocity controller.

4.5 Dual-arm Joint Group Velocity Controller

With the Joint Group Velocity Controller it is possible to directly control the velocity of a set of joints together. The joints to control are set using a ROS parameter and then every message sent to the command topic sets the target velocity for the configured joints. This controller is using the Velocity Interface directly. With this controller it is not possible to set position target or effort values.

This project is funded by the European Union G.A No 101016906

5 Sensors

The robot will have sensors on various places. The following sensors are foreseen:

- 2 Ouster OS1 lidars on the mobile base,
- 2 Septentrio GPS-RTK receivers on the mobile base,
- 1 SBG Ellipse IMU on the mobile base,
- 2 RealSense D435i RGB-D cameras on the end-effectors of the robotic arm,
- 1 RealSense D435i RGB-D camera inside the head.

The SBG IMU actually fuses the GPS data coming from the two receivers on the mobile base. On the ROS side these sensors are actually represented by one source in the shape of the SBG Ellipse device.

The sensors are included at predefined positions which match the expected positions on the robot itself. It is possible to configure alternative placement for each sensor by feeding an altered URDF configuration at the start of the simulation. One example of this are the RealSense cameras on the end effectors. It is physically possible to rotate the mounting bracket around the longitudinal axis of the end-effector plus the camera can be moved forward and backward on the bracket in order to achieve the optimal position to view the grapes and branches for harvesting and pruning. This rotation and translation can be set using the URDF (see Figure 12 and Figure 11).

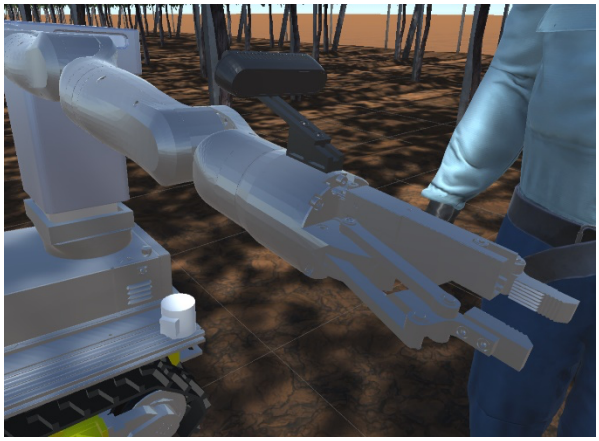

Figure 12 Default position RealSense camera on the end-effector

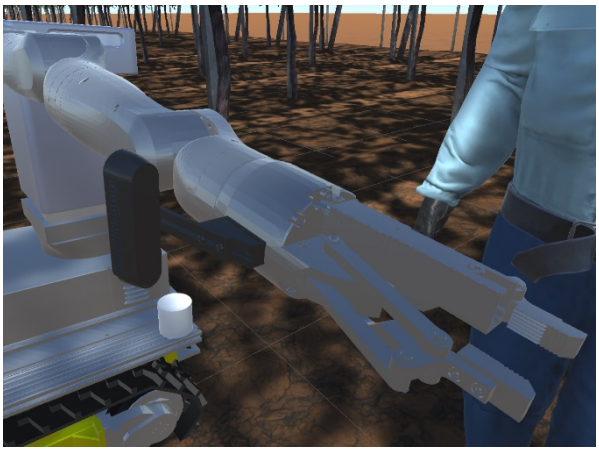

Figure 11 Updated position RealSense camera on the end-effector

It is even possible to set positions and orientations which are not physically possible to enable experiments with alternative mounting before these are actually produced in real life.

The simulation of the sensors themselves are documented in [D4.3] - VR Agronomic-Oriented Perception Component.

5.1 Human simulation for safety validation

For the testing of the detection of humans and safety, human figures will participate in the simulated environment. Multiple full-interactive humans using VR devices will be realized as part of work

package WP6. This will use a network setup to accommodate the multiple clients as described in [D2.1].

To enable early testing with human figures, a basic animated human figure will be included while the networked setup is not yet available (see Figure 13). This human can be controlled with keyboard input. With this input, the human can walk forward, backward, sideward and rotate along the vertical axis. The figure will use the same appearance as the human in the network setup: using a full-body avatar. These human figures will also have colliders such that the robot movements collide with the various part of the body.



Figure 13 Controllable human in the vineyard

The various sensors like the lidars and RGB-D cameras will see these human figures in the regular way and will communicate these as part of their images and point-clouds. This makes it possible to test human body detection and safety routines with these sensors.

This project is funded by the European Union
G.A No 101016906

6 A grape-vine

(Based on a poem of Hieronymus van Alphen)

A robot once saw grapes hanging,
Oh! Like eggs so big.

It looked like the robot wanted to pick,
though his master forbade him.

Here, he said, neither my master,
not the gardener, who sees it:

On a vine, so fully loaded,
one does not miss five, six grapes.

But I want to be obedient,
and don't pick: I ride away.

Would I, to have a bunch of grapes,
be disobedient? No.

Further went the robot: but his master,
who has listened to him silently,

Met him in walking
on the middle aisle.

Come, my robot, said the master,
come my little heart thief!

Now I will pick you grapes;
now the master loves his robot.

Then the master shook the vine,
the robot picked up quickly;

The robot got his hat full of grapes,
and ride away at a galop.

7 References

Id	Title	Version	Date
[D2.1]	Requirements, Specifications and Benchmarks	1.0	2021-04-30
[D2.2]	Specifications and KPIs for the two CANOPIES robot prototypes	1.0	2021-06-30
[D2.3]	Dimensioning of the Real-World Scenario for Final Demo	1.0	2021-10-29
[D2.4]	VR Farming Environment Specification	1.0	2021-10-29
[D3.4]	Simulated Farming Environment and Basic Robotic Components	1.0	2021-08-31
[D4.3]	VR Agronomic-Oriented Perception Component	1.0	2021-12-30