EU Horizon 2020 Research & Innovation Program

Advanced Robot Capabilities & Take-Up

ICT-25-2016-2017

Safe Robot Navigation in Dense Crowds

http://www.crowdbot.org

Technical Report

D 5.3: 2nd Updated and Extended Robot System

Work Package 5 (WP 5)

System Architecture & Integration

Task Lead: SoftBank Robotics Europe (SBRE), France

WP Lead: SoftBank Robotics Europe (SBRE), France

DISCLAIMER

This technical report is an official deliverable of the CROWDBOT project that has received funding from the European Commission's EU Horizon 2020 Research and Innovation program under Grant Agreement No. 779942. Contents in this document reflects the views of the authors (i.e. researchers) of the project and not necessarily of the funding source─the European Commission. The report is marked as PUBLIC RELEASE with no restriction on reproduction and distribution. Citation of this report must include the deliverable ID number, title of the report, the CROWDBOT project and EU H2020 program.

Table of Contents

TABLE OF FIGURES 4
EXECUTIVE SUMMARY 6
1. SOFTWARE INTEGRATION 7
1.1. Pepper robot 7
1.1.1. Vanilla controller 7
1.1.1.1. Simplified architecture schema 7
1.1.1.1.1. ALMotion module 7
1.1.1.1.2. DCM Module 8
1.1.1.2. Architecture profiling 9
1.1.1.2.1. Tests cases 10
1.1.1.2.2. Command Propagation 12
1.1.1.3. Summary 13
1.1.2. Boost Controller 14
1.1.2.1. Boost controller API 15
1.1.2.2. Changes in Naoqi Bridge 16
1.1.2.3. Results 17
1.2. Wheelchair 18
1.2.1. ROS nodes and topics 18
1.2.2. Control Systems Integration 19
1.2.2.1. Summary 22
1.3. CuyBot 22
1.4. Qolo 23
2. HARDWARE INTEGRATION 25
2.1. Sensor choice suite 25
2.1.1. RealSense D435 25
2.2. Pepper robot 26
2.2.1. Realsense D435 26
2.2.1.1. Feasibility study 26
2.2.1.2. First prototype 27
2.2.1.3. Second prototype 27
2.2.1.3.1. Heat dissipation 28
2.2.2. BAKI tensioner 29
2.2.2.1. Problem statement 29
2.2.2.2. Approach 30

2.2.2.3. Tensioner v0.1 31
2.2.2.4. Tensioner v0.2 32
2.2.2.5. Results 33
2.2.3. 3D printed fender 34
2.3. Wheelchair 34
2.3.1. RealSense RGDB Camera 35
2.3.2. Hokuyo LiDAR 36
2.3.3. Wheel Encoders 36
2.3.4. IMU 37
2.3.5. Sonar clusters 38
2.3.6. R-Net joystick integration 38
2.3.7. Hardware and software safety tests 39
2.4. CuyBot 41
2.4.1. RealSense D435 42
2.4.2. 2D LiDARs 43
2.4.3. PC and graphics card 43
2.5. Qolo 43
2.5.1. RealSense D435 44
2.5.2. 3D LiDARs 45
2.5.3. PC and graphics card 45
3. EXPERIMENTAL SCENARIO 47
4. CONCLUSIONS 49

Table of Figures

Figure 1. Simplified architecture schema for Pepper robot 9

Figure 2. The control response of the Pepper robot was tested through a joypad controller. Commands from the joypad were interpreted by the ROS joystick node which interfaces with the NAOqi bridge to the platform. 9

Figure 3. Pepper velocity measurements compared to moveToward commands from the joystick, including forward-back, and side-to-side movement. Velocities are in [m/s], moveToward commands are unitless [0, 1]. Commands were transmitted from the joystick ROS node to NAOqi OS via naoqi driver and ethernet. The joystick (x,y) commands are plotted in blue and green, respectively. The measured (x,y) velocities are plotted in cyan and red, respectively. 10

Figure 4. Comparison in velocity responses with and without using the killMove command. Velocities are in [m/s], moveToward commands are unitless [0, 1] 11

Figure 5. KillMove command applied to sideways velocities. Velocities are in [m/s], moveToward commands are unitless [0, 1] 12

Figure 6. Vanilla controller response (green) with respect to the sent command (red) for linear (top) and angular (bottom) velocities. 13

Figure 7. Command propagation delay results 13

Figure 8. New architecture schema for Pepper robot 14

Figure 9. Sketch of the Pepper's wheelbase 15

Figure 10. Boost motion controller response (green) with respect to the sent command (red) for linear (up) and angular (down) velocities 17

Figure 11. ROS nodes currently integrated into the smart wheelchair platform 19

Figure 12. Gazebo simulation showing, verifying our DWA implementation, prior to integrating it into the physical wheelchair platform. 21

Figure 13. ROS diagram of UCL's probabilistic shared control system, currently integrated into the smart wheelchair prototype. 21

Figure 14. The RWTH people tracker is integrated into the wheelchair platform, so pedestrians can be tracked in the surroundings and their motion incorporated into the wheelchair's shared control planner. 22

Figure 15. Image of cuyBot robot 23

Figure 16. Front, rear, up and down views of the Intel RealSense D435 25

Figure 17. Visual summary of the feasibility study for placing the D435 sensor on Pepper 26

Figure 18. First prototype of Pepper's redesigned scalp 27

Figure 19. Second prototype of Pepper's redesigned scalp 28

Figure 20. Readings of the three temperature sensors, in degrees celsius, placed on the camera during an exposure time of 3 hours 29

Figure 21. Representation of the backlash between motor and link gears 29

Figure 22. Combined VI-Sensor (green) and 2D LiDAR (black and gray) mapping and localization. Yellow points show matched features found using ROVIOLI. The RGB and depth data are also projected in the scene. 30

Figure 23. Components for the proof of concept of the tensioner. Attachment of the semi-rigid cable from the holes placed in the rear of the head and the bracket. 31

Figure 24. Components used in tensioner v0.1 32

Figure 25. Tensioner v0.2 prototype with a shape that solves the magnet adhesion issue 32

Figure 26. Tensioner v0.2 distances and angles to ensure adhesion and avoid the cable contact with the surface of the robot 33

Figure 27. Backlash reduction results acquired during testing with and without the BAKI tensioner v0.2. Conventional system (red) is shown in comparison with a backlash mitigation (green). 33

Figure 28. (a) CAD model of the front guard installed around Pepper's base. The fender was designed to be lightweight but structurally rigid. (b) Installation of Pepper's front guard. Only half of the fender is shown in to highlight the underlying structural support. 34

Figure 29. The full setup and installation of the fender and brace, showing isolation from the sensor mounts 34

Figure 30. The original CAD drawings and the integrated robotic wheelchair platform. All the hardware (emergency stop button, ultrasonic sensor clusters, joystick controller) that was used for the initial safety tests is also shown. 35

Figure 31. The Intel RealSense camera, mounted on a custom-built frame, which is then attached to the rear of the wheelchair 36

Figure 32. CAD model of the Hokuyo URG-04LX-UG01 Scanning Laser Rangefinder 36

Figure 33. Custom built wheel encoders are installed on the wheelchair 37

Figure 34. 3D printed wheel encoder cog 37

Figure 35. IMU in custom 3D printed housing, to mount securely on the wheelchair chassis 37

Figure 36. Left – CAD rendering of our ultrasonic sensor cluster. Right – CAD rendering of four sensor clusters, mounted at the four corners of the sensor frame, which in turn is affixed to the wheelchair chassis. 38

Figure 37. Stopping distance (cm) driving at minimum speed 39

Figure 38. Stopping time (s) driving at minimum speed 40

Figure 39. Stopping distance (cm) driving at maximum speed 40

Figure 40. Stopping time (s) driving at maximum speed 40

Figure 41. CuyBot robot 41

Figure 42. Heights of LiDAR (bottom) and both RealSense D435 (top) sensors 42

Figure 43. Field of view of front-facing camera when trunk is placed at the front (left image) or at the back (right image) of the robot's base. 42

Figure 44. Example of navigation in countercurrent flow 47

Figure 45. Example of cocurrent flow. Note that Pepper's top speed is approximately 0.3m/s, which is slower than an average human's walking pace 48

Figure 46. Example of cross-current flow 48

Executive Summary

Work package 5 (WP5) is about developing a coherent theoretical and functional system architecture that can accommodate the targeted scenarios and facilitate the integration of the different work packages across all three robotic platforms. However, this process must be iterative and reviewable after each milestone so that the achievement of such objectives is guaranteed during all project development. In order to reach the previous, coordinating periodic integration activities among partners, updating the platforms and incorporating new developments is a must. Specifically, we want to ensure that we are incorporating all project experiment needs, both in the robot architecture model and the hardware. At the same time, we must ensure the quality of individual components as well as its robustness is enough for posterior commercial and academic purposes. Then, ensuring that the low level sensor data from the robot and/or additional sensors is correctly managed so that high level situation assessment becomes possible.

The focus of this deliverable is to propose a flexible architecture that can be easily adapted for the three robot platforms that are involved in the CROWDBOT project. Therefore, the present document extends the already delivered D52.

1. Software Integration

The present section describes the software solutions designed for three different robot platforms: Pepper, Wheelchair and cuyBot. A review of the different approaches taken is provided; including problem definitions, design choices, technical description of the implementations and results.

1.1. Pepper robot

The software integration section for Pepper robot is highly focused on solving the issues that prevent the platform to perform movements which satisfy the demands of the navigation stack. That is to be able to move more reactively, in contrast with the default's robot controller (Vanilla controller).

1.1.1. Vanilla controller

To be able to react fast is critical for any autonomous system deployed in a crowded scenario. Therefore, key priorities such as safety during displacement, comfort of the surrounding users and naturalness of movements depend on a great manner on the dynamic response of the system. Such considerations are reported in the Periodic Technical Report -Part B, Section 1.2.3: WP3- submitted to the EC for the first year review, and strongly advised as target recommendation in deliverable *D62: Robot Design Recommendations*.

In this section, the Vanilla controller is tested and profiled targeting three main elements:

- Linear and angular accelerations,
- Linear and angular velocities,
- Time responses.

As a consequence, a presentation of the main components involved in the robot motion is provided in order to identify the performance bottlenecks. This section will serve as a preliminary study for the implementation described in the next section *Boost motion controller* and its results.

1.1.1.1. Simplified architecture schema

As described in *D52-First integrated prototype in experimental scenario*, Pepper's internal software architecture is managed by the NAOqi framework. Any command to be executed by the platform or data retrieved from the sensors goes through NAOqi. It is divided into different modules in charge of basic functionalities such as motion, perception, navigation or memory, and they are typically preceded by the prefix AL. Specifically ALMotion module provides methods which facilitate making the robot move.

1.1.1.1.1. ALMotion module

It is responsible for the robot kinematics, dynamics and control and robot reflexes, updating the model every cycle (20ms) at 50Hz, exposing four major groups of methods for operating the platform:

- Joint stiffness: Control the Stiffness of one or several joint(s). The Stiffness of the joint is equivalent to a torque limitation into the motors. If the joint stiffness is set to 0.0, the joint is free. On the contrary, if it is set to the maximum (1.0), the joint is allowed to use full torque power to reach a given position. Between these two extremes, the joint is more or less compliant (tries to reach a position but if the torque needed to move is higher than the limitation of the stiffness, the joint doesn't reach the target position).
- Joint position: controls directly the position of the robot joints. Each joint can be controlled individually, or in parallel with other joints.

- Robot effector in the Cartesian space: It is dedicated to control directly the Effectors of the robot using an inverse kinematics solver. Each effector can be controlled individually, or in parallel with other. It has a classical IK solver which uses only the joints of the effector chain to reach a target and a generalized IK solver (also called Whole Body control) which uses all the robot joints to reach a target.

For the purpose of the evaluation only the following methods are tested:

void ALMotionProxy::**moveToward**(*x*, *y*, *theta*)

Makes the robot move at the given normalized velocity. This is a non-blocking call.

Parameters:

- *x* – normalized, unitless, velocity along X-axis. +1 and -1 correspond to the maximum velocity in the forward and backward directions,
- *y* – normalized, unitless, velocity along Y-axis. +1 and -1 correspond to the maximum velocity in the left and right directions, respectively.
- *theta* – normalized, unitless, velocity around Z-axis. +1 and -1 correspond to the maximum velocity in the counterclockwise and clockwise directions, respectively.

void ALMotionProxy::**moveTo**(*x*, *y*, *theta*)

Makes the robot move to the given pose in the ground plane. This is a blocking call.

Parameters:

- *x* – Distance along the X axis in meters.
- *y* – Distance along the Y axis in meters.
- *theta* – Rotation around the Z axis in radians [-3.1415 to 3.1415].

void ALMotionProxy::**killMove**()

Emergency Stop on Move task: this method will end the move task brutally, without attempting to return to a balanced state. If the robot has one foot in the air, he could easily fall.

Eventually, the results of the computations performed by the previous methods within ALMotion deliver an input to a lower level controller through the DCM.

1.1.1.1.2. DCM Module

The Device Communication Manager or DCM is a software module, part of NAOqi, in charge of the communication with almost every electronic device in the robot (boards, sensors and actuators), except sound and camera devices. It also manages the main communication buses between the head and the body, and the communication inside the head. In fact, it is the link between the “upper level” software (other NAOqi modules) and the “lower level” software (firmware).

Modules like ALMotion directly send commands to the actuators using the DCM, while extractors and other modules use sensor results returned by the DCM in ALMemory. It is important to notice that the DCM is the NAOqi module part of HAL or Hardware Abstraction Layer which runs safety protocols and whose use is mandatory to access the robot hardware. It runs in a constant-time cycle, typically between 1 and 10 ms, depending on the robot.

The working principle of the DCM is simple:

- The DCM API allows sending commands to Actuators. You can update, with one call, one or more actuator(s). Commands include a time information of when the actuator needs to receive the value.

- To get actuator and sensors values, you need to read them in the ALMemory. The DCM updates them as far as they change.

As a result, the schema presented in Figure 1 represents the data cycle when sending a command to naoqi from the NAOqi_bridge ROS node provided by SBRE.

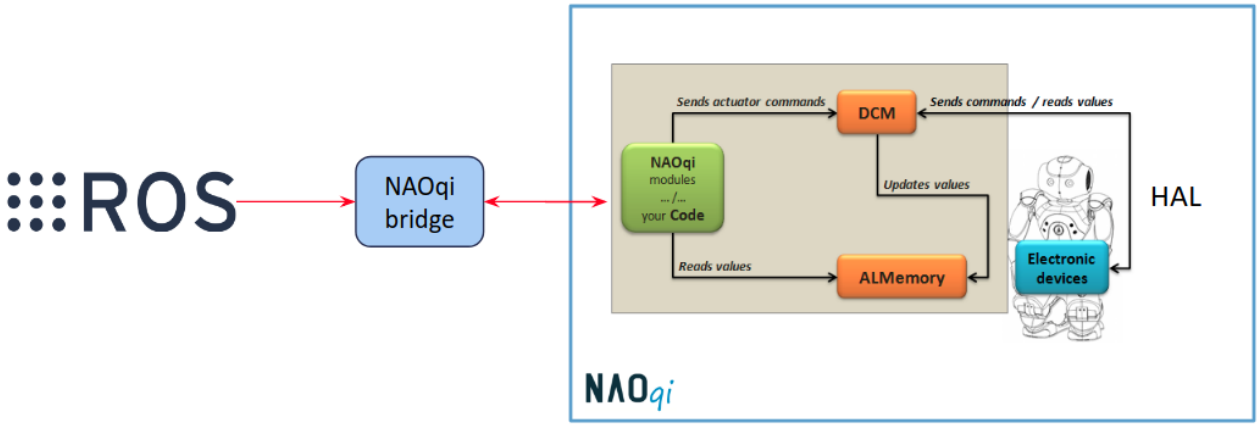


Figure 1. Simplified architecture schema for Pepper robot

1.1.1.2. Architecture profiling

This report details the control behaviors of the Pepper robot. It provides an analysis of the platform response to input commands from a joypad controller. The findings in this section will be used to identify the module bottlenecks in order to suggest approaches to improve time responses, velocity and acceleration.

During the experiment, the Pepper robot was controlled through the ROS interface, specifically the open source naoqi driver ROS package. Commands from the joystick device were transmitted through ROS messages to the naoqi driver ROS node running on an offboard PC (see Figure 2). naoqi driver in turn converted ROS messages to ALMotion commands, according to the NAOqi API. The ALMotion commands were then transmitted through ethernet connection to the Pepper onboard computer, where the running NAOqi OS executed them.

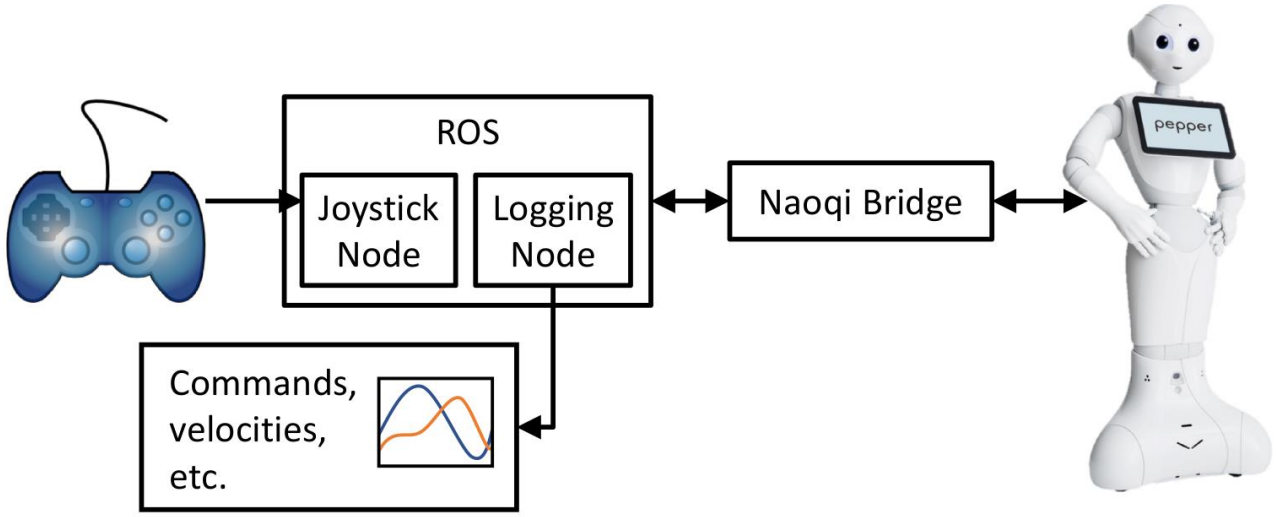


Figure 2. The control response of the Pepper robot was tested through a joypad controller. Commands from the joypad were interpreted by the ROS joystick node which interfaces with the NAOqi bridge to the platform.

The joystick was set to a variable refresh rate, with a minimum of 2Hz and maximum of 100Hz. Changes in the value of an axis or button state immediately trigger a corresponding motion command (within the maximum rate limit), whereas in the absence of changes in axis or button values, the last motion command is repeated at a rate of 2Hz. For motion testing, Pepper was placed in a large room with no obstacle and on soft, grippy surfaces where no significant wheel slip was observed (see Figure 3).

Figure 3. Pepper velocity measurements compared to moveToward commands from the joystick, including forward-back, and side-to-side movement. Velocities are in [m/s], moveToward commands are unitless [0, 1]. Commands were transmitted from the joystick ROS node to NAOqi OS via naoqi driver and ethernet. The joystick (x,y) commands are plotted in blue and green, respectively. The measured (x,y) velocities are plotted in cyan and red, respectively.

1.1.1.2.1. Tests cases

The following tests cases were carried out in order to quantify the control envelope of the Pepper robot:

1) Maximum forward acceleration to top speed.
2) Maximum reverse acceleration from top forward speed to top reverse speed.
3) Maximum side acceleration to top speed.
4) Maximum side acceleration from top speed to top speed in the opposite direction.
5) Stopping from top forward speed with *killMove* command.
6) Stopping from top side speed with *killMove* command.
7) Maximum reverse acceleration from top forward speed to top reverse speed using *killMove*.

During testing, delays are dissected as far as possible into their components and causes, in order to evaluate the influence of various parts of the pipeline responsible on latency.

Velocity response measurements are extracted from NAOqi OS odometry, however their accuracy is not well known. Further testing with external sensors would be necessary in order to acquire a more precise estimate of the velocity response. The response times measured in the velocity tests were however large enough ($1 \cdot 10^3$ ms) that they dominate over estimated measurement bias and standard deviation ($1 \cdot 10^2$ ms).

The acceleration profile average times presented in Table 1 are measured from the start of servo actuation to the realization of target velocity. The exact point for realization is not straightforward to define. We define it as being 50 ms before the first inversion of the velocity derivative within $0.95v_{max} - v_{max}$. For values corresponding to tests 1 to 6, uncertainty was estimated to be in the order of 100 ms, composed of i) uncertainty in the propagation time from joystick to servo actuation, ii) variance due to odometry data noise, iii) variability of the target velocity realization condition.

Table 1. Pepper velocity average time responses

Test set	Average time $[s]$
1. $v = 0$ to forward v_{max}	1.3
2. forward v_{max} to backward v_{max}	2.4
3. $v = 0$ to side v_{max}	1.3
4. side v_{max} to opposite side v_{max}	2.4
5. forward v_{max} to $v = 0$ with killMove	0.9
6. side v_{max} to $v = 0$ with killMove	0.9
7. forward v_{max} to backward v_{max} with killMove	1.9 - 2.4

For the purpose of testing, the joystick was configured to emit the *killMove* ALMotion command upon button press. The recorded servo accelerations when executing the *killMove* command are demonstrably higher than the maximum acceleration values attainable through the ALMotion *moveToward* command. No toppling over or unstable behavior was observed during the *killMove* maneuver, see Figure 4 and Figure 5.

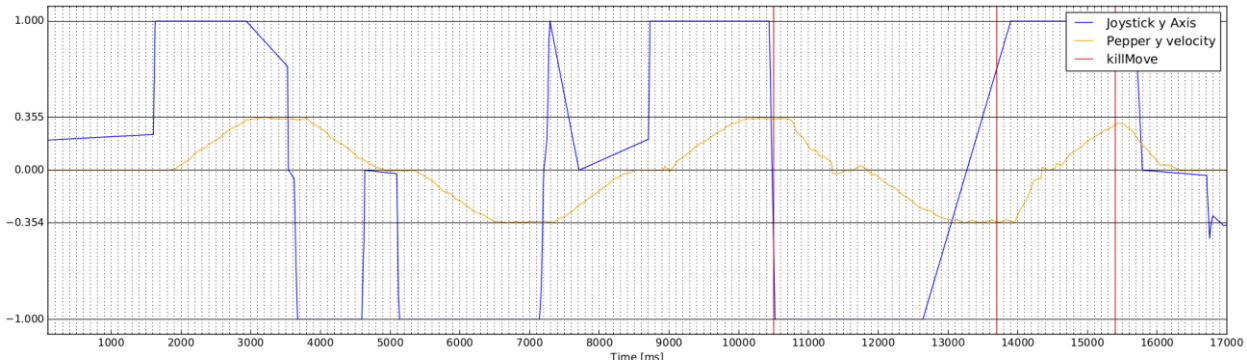


Figure 4. Comparison in velocity responses with and without using the killMove command. Velocities are in [m/s], moveToward commands are unitless [0, 1]

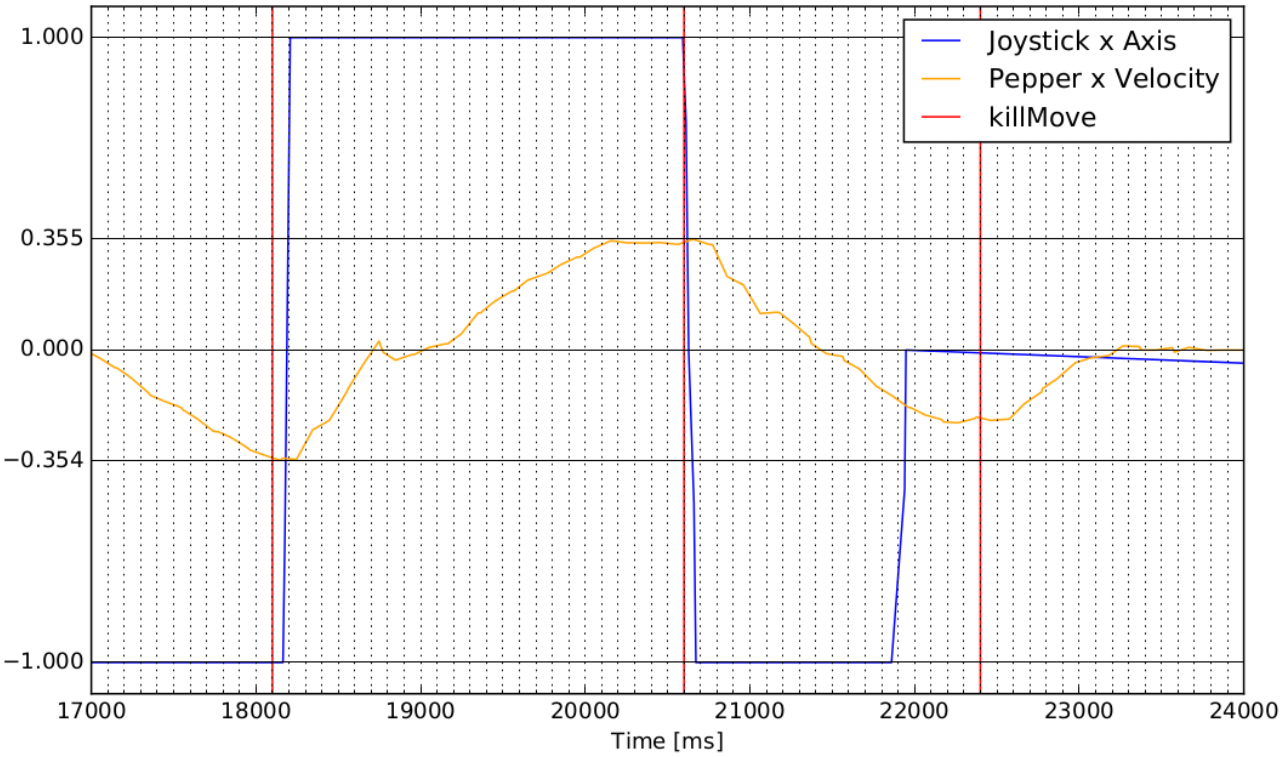
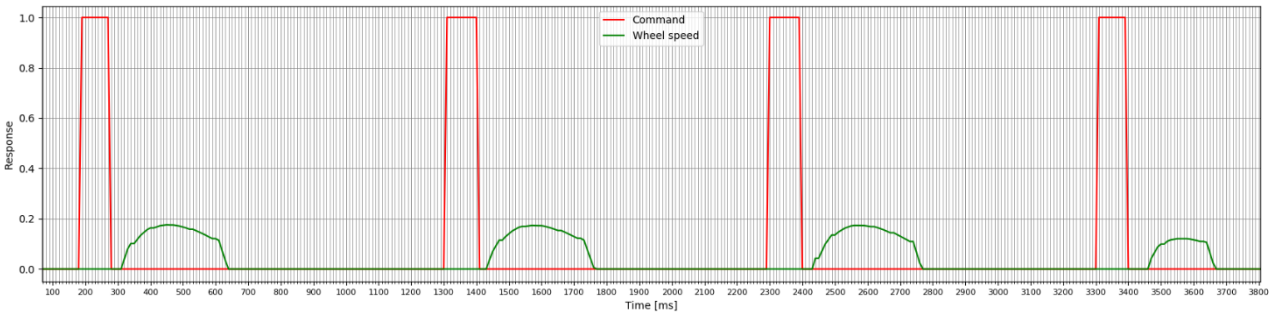


Figure 5. KillMove command applied to sideways velocities. Velocities are in [m/s], moveToward commands are unitless [0, 1]

1.1.1.2.2. Command Propagation

The time response when a command is sent to the wheelbase has also been a matter of profiling since a visual delay observed from the moment a key is pressed till the actuator moves, is considerably high. In consequence, Figure 6 shows the correspondent time responses for angular and linear velocities using the default Pepper controller. As can be observed, there is a very conservative response in terms of power delivery and a considerable delay between command sent and wheel response.



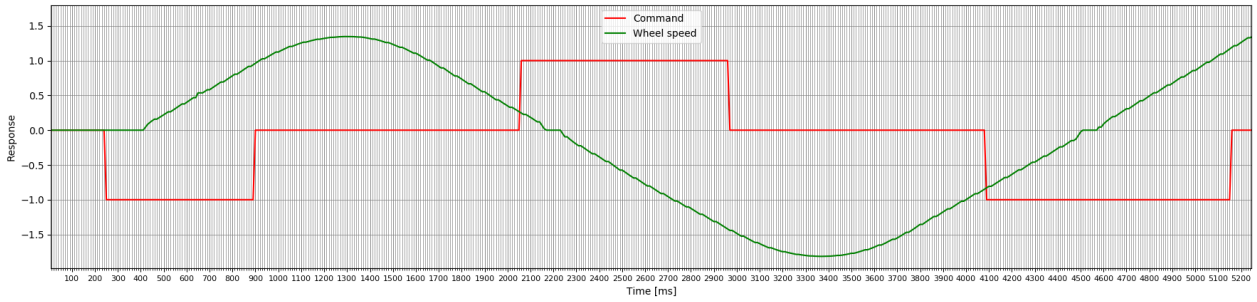


Figure 6. Vanilla controller response (green) with respect to the sent command (red) for linear (top) and angular (bottom) velocities.

The summary of the previous observations is shown in Figure 7.

From	To	Delay [ms]
Joystick actuation registration	moveToward command emission	20 ± 10
moveToward command emission	Reception NAOqi OS (LAN)	0.3 ± 0.03
Reception NAOqi OS	Servo actuation	100 ± 50

Figure 7. Command propagation delay results

1.1.1.3. Summary

The shape of the velocity measurements shows smooth changes in velocity, which would seem to indicate that the NAOqi OS wheel controller enforces linear jerk/constant snap constraints. This hypothesis is further reinforced when comparing acceleration times for i) max velocity to stop, and ii) max velocity to min velocity. The measured time for case i) takes on average 100 ms longer than that of case ii). If acceleration was simply mapped linearly to joystick axis for example, and set to 0 when the target velocity was reached, we would expect the delay in case i) to be exactly half that of case ii). If linear jerk is enforced however, the necessary reduction of acceleration when approaching the target velocity leads to additional delay in reaching the target velocity.

Based on the maximum accelerations measured using the i) *killMove* and ii) *moveToward*, it would seem that the NAOqi OS sets conservative limits on maximum acceleration during normal operation which can only be exceeded in exceptional scenarios (*killMove*). One consequence is that using maximum values for the *moveToward* Pepper control is not only safe, but almost always recommended, as its use does not approach in any way the mechanical and kinematic limits of the robot.

The possibility of using *killMove* not only for stopping, but also for changing direction as quickly as possible, was a point of interest. However, even though the command leads to higher deceleration than otherwise, there is a significant variable delay after its use before the robot starts executing movement commands again. In our tests, this delay often negated the gains achieved from using the command for this purpose. Furthermore, in our use case, predictability of command completion-time is strongly desired. As such, using *killMove* for direction changes is not currently advisable.

In conclusion, the implementation of controller that increase the reliability and performance of the wheelbase is necessary to ensure a suitable baseline for integration of navigation solutions and reactive navigation approaches that match the dynamics of a crowd.

1.1.2. Boost Controller

In order to successfully decrease the time response of the wheelbase when a command is sent from the planner till it is received in the corresponding actuator, Pepper's default vanilla controller has been replaced by a custom controller. In this way, costly operations computed before a command is sent to the wheelbase are bypassed through a lightweight motion controller that contains the minimum requirements to operate the base through the DCM module as shown in Figure 8.

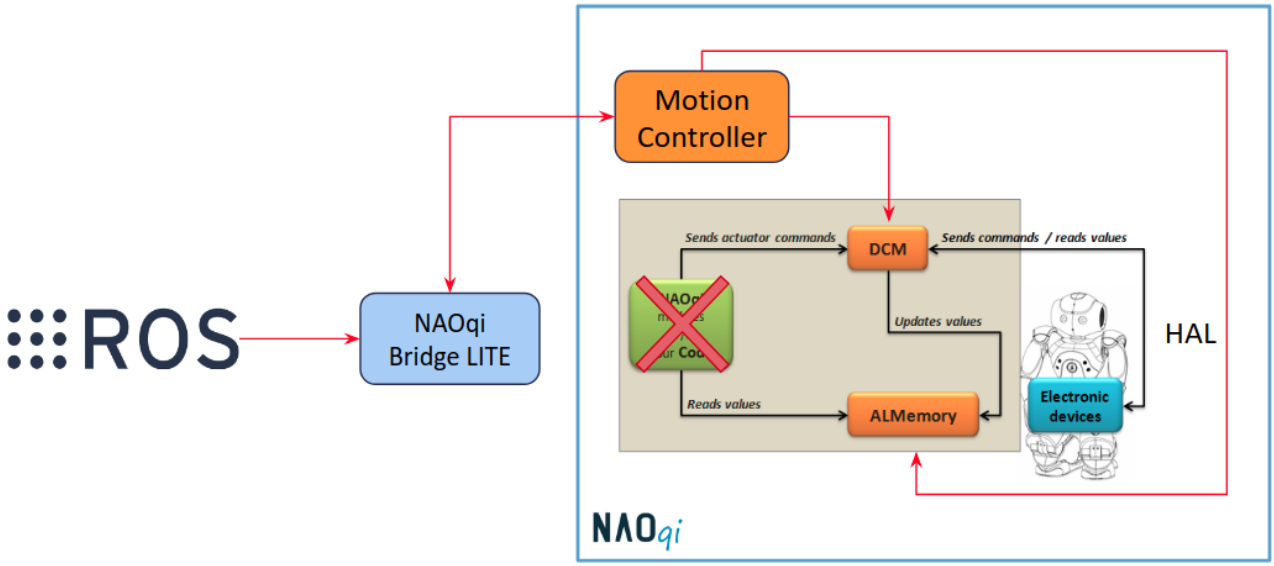


Figure 8. New architecture schema for Pepper robot

Accessing the DCM to be able to send a command to the corresponding wheel is essential for the controller. In order to do so, the name of the Actuator to use, or the name of the Sensor value to read (from ALMemory), is made of 3 parts separated by a slash:

Prefix	This prefix is not mandatory for actuators in the API. (Fixed, but also given by the getPrefix API).
Name	the name of the sensor/actuator, generally composed of different sections using "/" as a separator. All sensor are ended with "sensor" and all actuator by "actuator".
Attribute	The attribute for the actuator and sensor main value is simply "Value". This is the most useful attribute.

Let us consider one example.

The name of the electric current Sensor of the joint named headPitch is:

Device/HeadPitch/ElectricCurrent/Sensor/Value

Then, the kinematic configuration is needed. Considering Figure 9, the wheel angles can be extracted and used to compute the desired direction towards the goal g(x,y) by subtracting the angle of such goal to the angle of each wheel with respect to the center of the base.

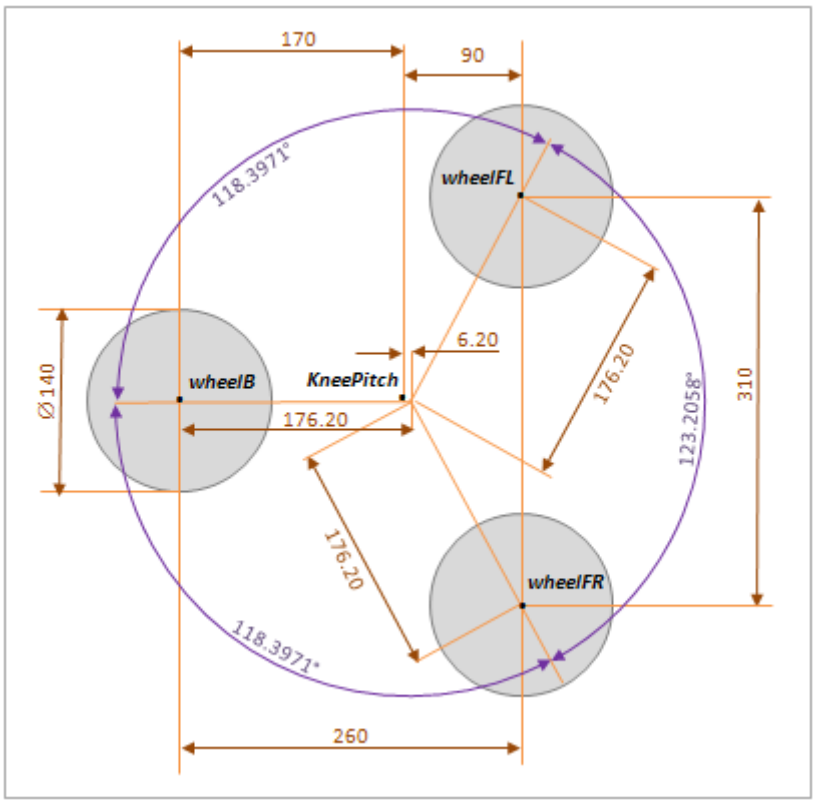


Figure 9. Sketch of the Pepper's wheelbase

Based on the wheel disposition the following operations are performed in order to obtain the resulting motion vector:

```
wheelFL += vel * std::cos(2.64 - atan2(y/x));
wheelFR += vel * std::cos(0.50 - atan2(y/x));
wheelB += vel * std::cos(4.71 - atan2(y/x));
```

Additionally, a safety feature has been added with respect to the default controller that allow us to safely stop the platform in case the network connection between the external computation unit and the robotic platform is interrupted. When this issue takes places, the default behavior is the execution of the last command indefinitely. In order to accomplish it, the time difference between sequential commands is monitored so that it does not exceed a predefined safety threshold. If it happens, a stop command is sent to the wheels to cease the displacement.

1.1.2.1. Boost controller API

The resulting API from the previous design is defined following the original NAOqi controller and it is shown below:

void MotionController::move_toward(linear_x, linear_y, angular_z)
Moves the robot base at a given velocity.

Parameters:
- *linear_x* – Velocity along the x-axis in meters per second.
- *linear_y* – Velocity along the y-axis in meters per second.
- *angular_z* – Velocity around the z-axis in radians per second.

void MotionController::set_acceleration(acc)

Sets a given acceleration.

Parameters:
- *acc* – Acceleration along the x y and wz axis in meters per second squared.

void MotionController::set_angular_velocity(angular_vel)

Sets a given angular velocity.

Parameters:
- *angular_vel* – Velocity along the wz in meters per second.

void MotionController::set_linear_velocity(linear_vel)

Sets a given linear velocity.

Parameters:
- *linear_vel* – Velocity along the x and y-axis in meters per second.

bool MotionController::is_enabled()

Returns whether this node is enabled (received move commands).

Returns:
- *linear_vel* – Velocity along the x and y-axis in meters per second.

1.1.2.2. Changes in Naoqi Bridge

As a consequence of deploying a new controller, several changes have been performed in the stock Naoqi driver, part of NAOqi Bridge ROS nodes. Firstly, in order to ensure the compatibility of the new Boost controller with ROS. Secondly, with the goal of maximizing the time responses and shrink delays between high and low level controllers:

- Modifying the current NAOqi driver calls to the vanilla motion controller, by calling the new Boost controller instead.
- Disabling unnecessary services with potentially high consumption of computational power such as RGB, depth and IR cameras, sonar, audio and IMU units, as shown in Table 2. Default NAOqi services available through NAOqi bridge and final state for performance enhancement.Table 2.

Table 2. Default NAOqi services available through NAOqi bridge and final state for performance enhancement.

Service	Default	Lite	Description
front_camera	TRUE	FALSE	Front RGB camera frames retrieved at 10 fps
bottom_camera	TRUE	FALSE	Chin RGB camera frames retrieved at 10 fps
depth_camera	TRUE	FALSE	Eye depth camera frames retrieved at 10 fps
ir_camera	TRUE	FALSE	Infrared sensor information retrieved at 10fps
info	TRUE	TRUE	Robot description and definition
logs	FALSE	FALSE	Record of communication events
diag	TRUE	TRUE	Diagnostic resources (temperature, battery, electric

			current)
imu_torso	TRUE	FALSE	Nao inertial unit located in the torso
imu_base	TRUE	TRUE	Pepper inertial unit located in the base
joint_states	TRUE	TRUE	Robot joint states
laser	TRUE	TRUE	Base laser values
sonar	TRUE	FALSE	Front and back sonar sensor values
audio	TRUE	FALSE	Mic and recording capabilities
bumper	TRUE	TRUE	Base bumper values
touch_hand	TRUE	TRUE	Hands tactile sensors
touch_head	TRUE	TRUE	Head tactile sensors
odom	TRUE	TRUE	Robot wheel odometry

1.1.2.3. Results

As can be seen in Figure 10, the response of the system after a command is received shows a substantially lower delay in comparison with Figure 6. In addition, it can be seen that a less conservative acceleration and a raise of both linear and angular velocities when a command is sent till the actuator executes it.

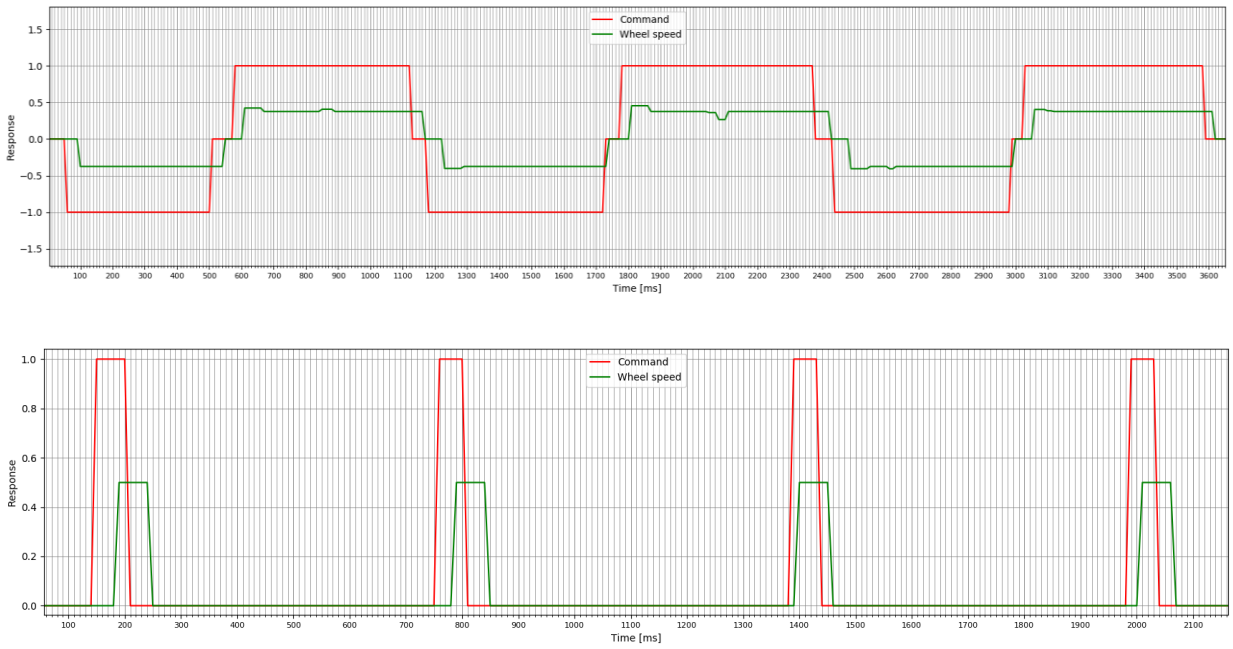


Figure 10. Boost motion controller response (green) with respect to the sent command (red) for linear (up) and angular (down) velocities

As a result, the summary of the enhancement can be seen in Table 3.

Table 3. Variables provided in NAOqi. Responses using both controllers and improvement rate.

Variable	Description	Vanilla	Boost	Improvement
MaxVelXY	Linear velocity (m/s)	0.35	0.55	x1.57

MaxVelTheta	Angular velocity (rad/s)	1	2	x2
MaxAccXY	Acceleration (m/s²)	0.3	0.55	x1.83
MaxAccTheta	Angular acceleration (rad/s²)	0.75	3	x4
TimeResp	Time passed till actuator responds (ms)	~150	~40	x3.75

1.2. Wheelchair

The smart wheelchair operates in semi-autonomous or "shared control" mode where the human operator continuously provides a joystick input, but before the corresponding control signals are sent to the wheelchair motor control unit, they may be modified to provide some level of driving assistance. The smart wheelchair itself is built upon a conventional power wheelchair with additional sensors and computational power embedded into the platform (see section 2.3 Wheelchair). As such all of our additional sensors must be available as ROS nodes, to allow easy integration with nodes and modules from other work packages.

In this section, we first describe these ROS nodes and topics, which allows interoperability between partners. We then go on to explain how we have implemented a number of example control methods to test the architecture, from a simple inverted joystick behavior, to a "virtual bumper" for safety and then an initial shared control mode, based upon the classic dynamic window approach.

1.2.1. ROS nodes and topics

Since the wheelchair platform is being developed to run on ROS, we have created a package that provides access to each of the sensing and motor control nodes in a simple launch file. Currently, the following nodes are implemented in the ROS package (Figure 11), which interface with the hardware described in section 2.3 Wheelchair:

Nodes	Topics
sonarFront	/rangeOne /rangeTwo /rangeThree /rangeFour /rangeFive /rangeSix
sonarBack	/rangeOne /rangeTwo /rangeThree /rangeFour /rangeFive /rangeSix
imu	/data /data_raw

wheel_chair_bridge	/joy /user_velocity_input /velocity_cmd
odometry	/odom
hokuyo	/scan
camera	/depth/image_rect_raw /color/image_rect_color

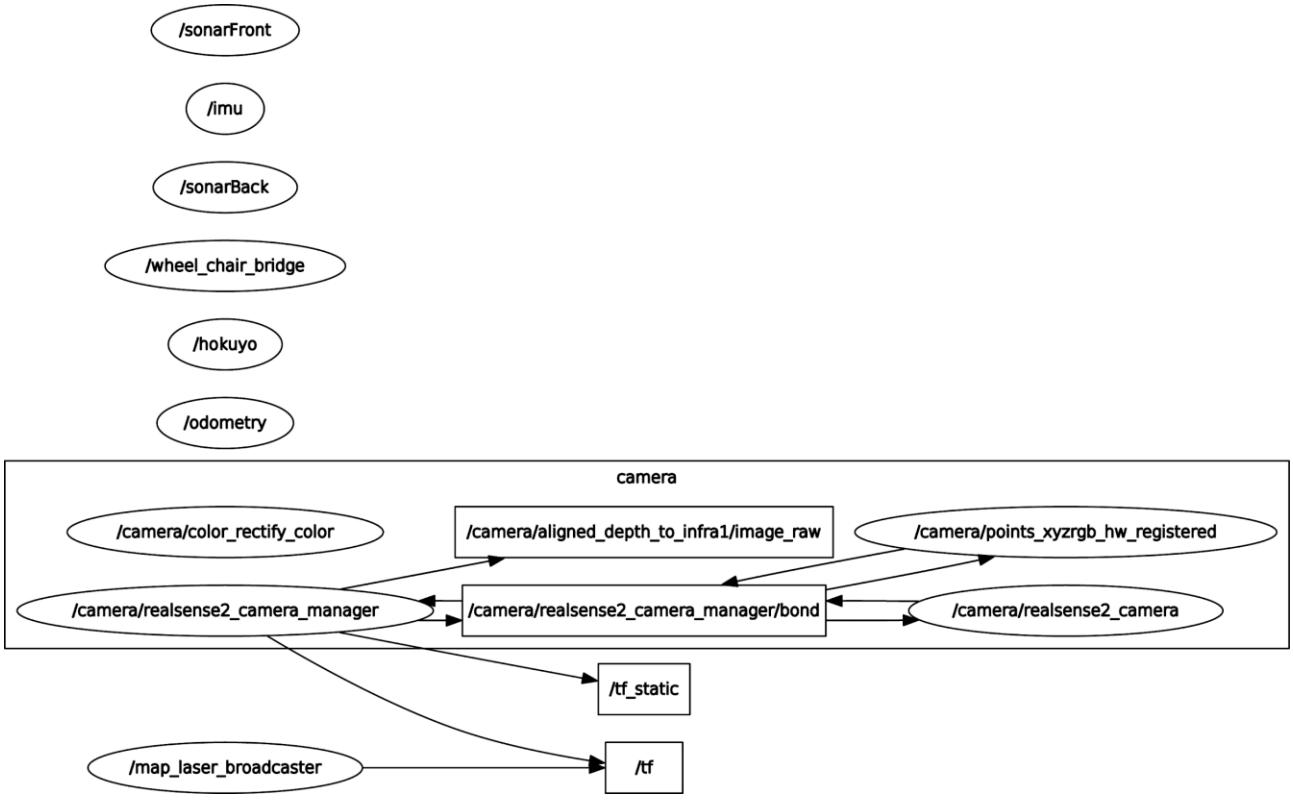


Figure 11. ROS nodes currently integrated into the smart wheelchair platform

1.2.2. Control Systems Integration

To test the functionality of the implemented nodes some test scripts were generated.

Invert Joystick

A simple test of the *wheel_chair_bridge* node that inverts the axis of the user input. The resultant behaviour has been verified, which demonstrates that the ROS script can read the joystick signals from the wheelchair's

R-Net[1] bus, modify them (in this case invert the axes), and then send the modified signals to the motor control unit.

The virtual bumper is a simple control system that uses ultrasonic sensors and wheelchair bridge node.

The user's input on the joystick is passed directly back to the motor control unit unless an ultrasonic sensor detects an obstacle in close proximity. Different directions of movement commands are inhibited depending on which sensors are in proximity to an obstacle. This test used a static distance threshold as a final safety check. However, in practice it is not that useful as a control system, since users are likely to get stuck going through narrow doors (which can oftentimes only be in the order of centimetres greater than the width of the wheelchair itself). It is, however, a good demonstration of the implementation and speed of the ultrasonic sensors, joystick and motor controller nodes.

Dynamic Window Approach

The classic dynamic window approach has also been implemented as the fundamental collision avoidance algorithm for our wheelchair. It works by computing a set of feasible velocities that can be reached in the next time step. From these velocities, a set of safe velocities that would not result in a collision is computed. Then all velocities are assigned weights that correspond to the sum of the speed, distance to the obstacle (clearance) and heading to the goal. This sum is known as the cost function. The final velocity with the maximum cost is selected.

Prior to the integration with the wheelchair itself, we first tested DWA in our simulators (gazebo and Unity3D; at this stage we used one 360 degree laser sensor in Unity3D for simplification). In gazebo, 12 sonars with 3 in each corner of the wheelchair were added to the simplified wheelchair model. Sonar readings were used to constructed a 40m*40m global map with occupancy grids. A fixed goal pose was used to test the algorithm, as shown in Figure 11Figure 12, where the simulated wheelchair is able to navigate around a narrow maze, safely without collisions.

To integrate the DWA implementation into our physical wheelchair platform, a main laptop is connected to the onboard Odroid as a ROS master and slave, respectively. At this stage, the Odroid on the wheelchair processes information from the wheel encoders and 12 sonar sensors, which are further published as ROS topics */odom*, */sonarFront* and */sonarBack* (c.f. Figure 1) at a rate of 10Hz, 5Hz and 5Hz separately. In the future, a laser node will be added. The main laptop subscribes to these topics and runs the *sonar_map* and */dwa node*. The final velocity command *velocity_cmd* is currently published back to the wheelchair at 5Hz.

[1] proprietary protocol by Curtiss-Wright, built upon the CAN standard

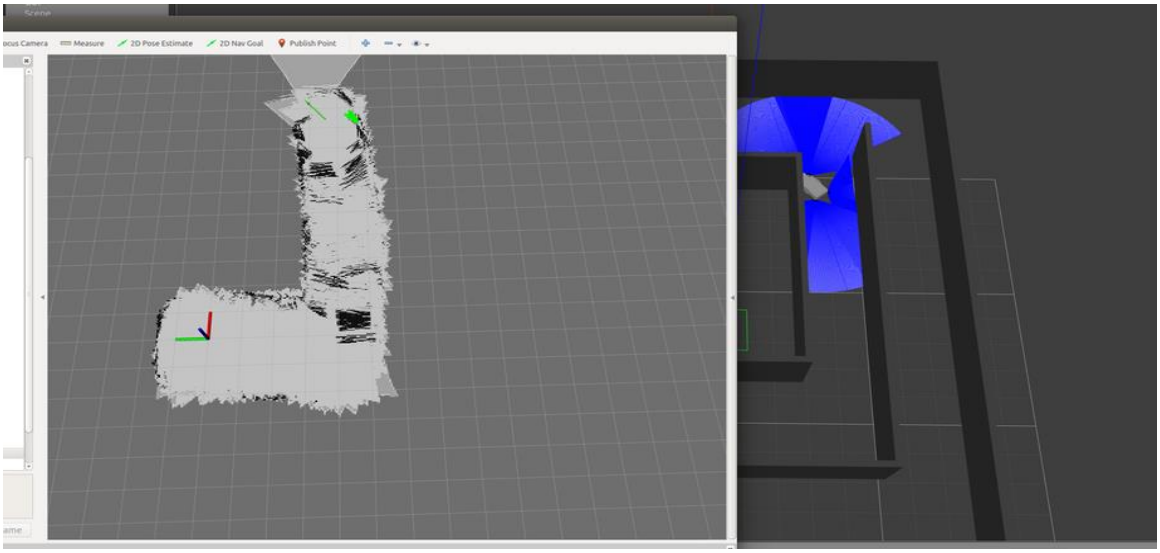


Figure 12. Gazebo simulation showing, verifying our DWA implementation, prior to integrating it into the physical wheelchair platform.

Probabilistic Shared Control

In addition to DWA, we also integrated UCL's probabilistic shared control module. When shared control is running, it subscribes to the user's joystick input */joy* from the Odroid and converts that to a standard Twist message */user_command*, as can be seen in Figure 13.

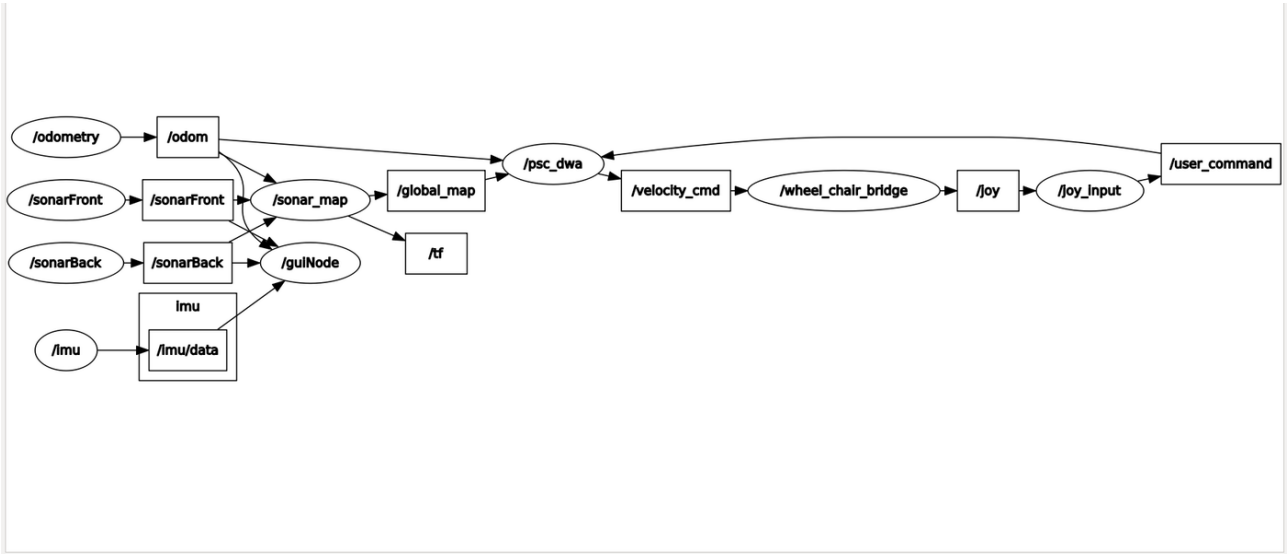


Figure 13. ROS diagram of UCL's probabilistic shared control system, currently integrated into the smart wheelchair prototype.

Velocity Obstacle

To account for dynamic obstacles, we have also implemented velocity obstacles in our simulator, allowing for dynamic safety thresholds to be set.

Similar to DWA, velocity obstacle (VO) is a velocity-based planning approach. It has been widely used in robot navigation in dynamic environments as it ensures a collision free path by removing all the velocities that may cause a collision in the near future. In VO, finding a suitable velocity mainly involves two steps. To start

with, all the obstacles are assumed as circular objects and they are mapped to the configuration space of the robot. For each obstacle, a collision cone is constructed and the relative velocity between the robot and the obstacle that falls outside of the collision cone is guaranteed to be collision free. After that, another set of admissible velocities that can be reached at the next time step was constructed by considering the robot's kinematic constraints. The final velocity is selected from the collision free and admissible velocity set which maximize a custom cost function.

We will integrate and test this planner on our wheelchair in the near future, but before we can do so, we need to be able to estimate the velocities of pedestrians. Hence the next subsection briefly discusses the integration of the RWTH people tracker module into the wheelchair platform.

People Tracker integration

The people tracker from RWTH has been integrated into the UCL wheelchair platform. The tracker detects humans and outputs their estimated velocity trajectories using RGBD camera and LiDAR sensor data, as shown in Figure 14.In future this data will be incorporated into the wheelchair's shared control planner to aid robot navigation in dense crowds.

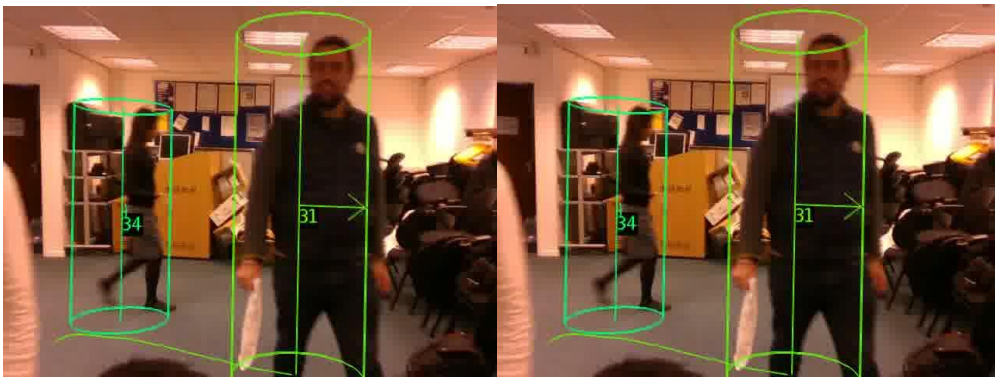


Figure 14. The RWTH people tracker is integrated into the wheelchair platform, so pedestrians can be tracked in the surroundings and their motion incorporated into the wheelchair's shared control planner.

1.2.2.1. Summary

We have seen that this framework makes it relatively easy to test different planners in our simulator and then deploy onto our actual wheelchair platform, although of course there is always need for recalibration of parameters due to the imperfect simulated model. Perhaps more importantly though, our planner can be easily replaced with others due to the modular setting. Hence our next steps will involve integrating the ETHZ local planner into the wheelchair platform.

1.3. CuyBot

The integration of CROWDBOT software on cuyBot (see Figure 15) is mainly done by exploiting the ROS framework. The PC used in the robot uses a standard Ubuntu installation. Ubuntu version 16.04 with ROS Kinetic was mostly used so far in accordance with the other project partners. The only required specific ROS package for cuyBot is the driver for the so-called SmartWheel actuators. Since this driver has very few dependencies, it can run under all recent versions of ROS. It offers the usual interfaces expected by the other CROWDBOT components, in particular topics /cmd_vel for motion commands, /odom to provide odometry data, and /tf for coordinate transforms.

Figure 15. Image of cuyBot robot

For all relevant sensors (LIDAR, RGB-D cameras) used on cuyBot, the default ROS driver packages can be used. Also coordinate transforms to specify the relative coordinates of all relevant parts of the robot are provided. Therefore integration of software requires mainly the eventual renaming of ROS topics and adjusting of configuration files. One other important aspect for integration is the required computational power of PC and graphics card. This issue will be discussed in Section 2.4.

Low-level control of the cuyBot is implemented within the SmartWheels themselves and in a ROS driver node that synchronizes all wheels and makes the bridge between ROS and the wheels. The SmartWheels use EtherCAT to communicate with the PC, a real-time capable industry bus system. By default the ROS driver runs its EtherCAT loop in a 1 kHz cycle. Therefore any ROS motion command coming from the navigation stack can be executed within milliseconds. Acceleration limits are currently rather arbitrarily set in order to avoid too sudden motions of the robot, which might surprise or discomfort people around it. Also the maximal velocity is restricted now to 1 m/s, which is about walking speed of a human. Higher velocities can be chosen if it should be desired. Therefore for CROWDBOT we don't expect issues with speed, acceleration or delays due to the robot in the upcoming tests.

For simulation of the cuyBot robot for the CROWDBOT simulator a model was provided. The model works in Unity as well as in Gazebo. Amongst others it was used in CROWDBOT for the analysis of collision points between the robot and people. This model is still being updated in order to make it more realistic and at the same time to reduce the complexity of its 3D data.

1.4. Qolo

As in all other robots in the CROWDBOT project, we have prepared the system for software integration of all partners exploiting the robot operating system (ROS). For this propose, we have developed a set of ROS packages for providing input and output communication from the robot's sensors (LIDAR, RGB-D cameras, F/T sensor), controller, and low-level interface. This architecture allows integration of software requiring mainly the renaming of ROS topics and configuration files; with only one specific driver packages used only for the Qolo robot. The *qolo_control* package provides a user intention recognition inferring the user desired

input from the embodied control system and publishes in the topic */qolo/user_input* as an array with linear an angular velocity, as well, this package provides topics with the state of the robot: velocity from shared control output *qolo/velocity*, actual wheels' velocities *qolo/wheels* and emergency switches state *qolo/emergency.*

For distributing the computational cost of sensor input processing, user intention recognition, and low-level control, we have divided each of these processes into independent computers, as shown in Figure 16. The main low-level control is performed at 500 Hz in an embedded microcontroller that communicates directly with the in-wheels control system, on top of it an UpBoard Squared embedded computer controls for the user input recognition and shared control output. Finally, the sensors are divided in RGBD being processed with a Nvidia Jetson AGX (embedded computer with 512 cores GPU), and LIDAR and F/T sensor data being processed at a second Upboard computer.

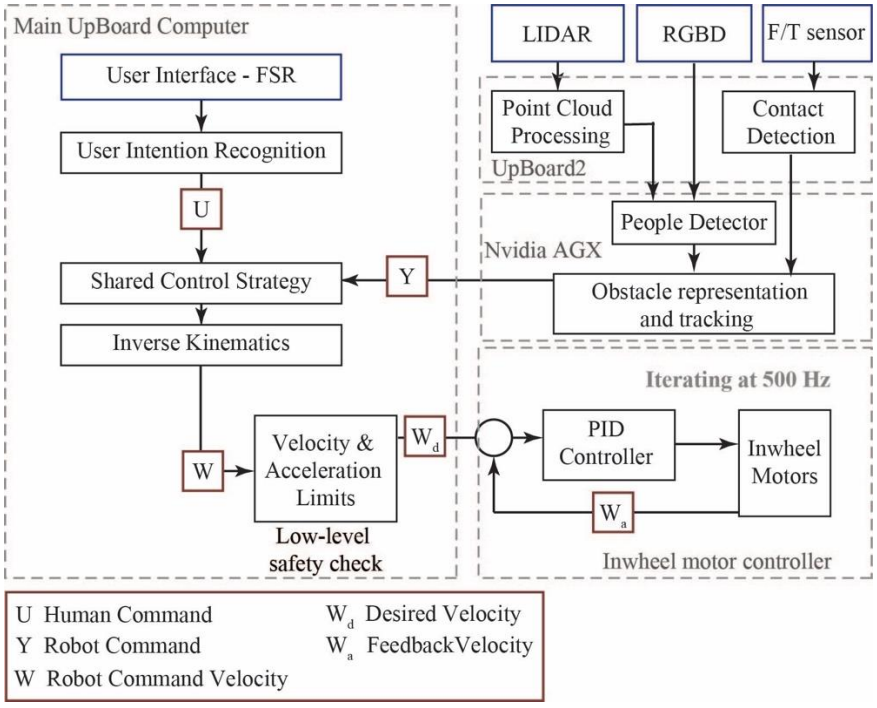


Figure 16. Qolo's controller distribution for CROWDBOT project

2. Hardware Integration

The present section describes the hardware solutions designed for the three different robot platforms: Pepper, Wheelchair and CuyBot. A review of the different approaches taken is provided; including problem definitions, design choices, technical description of the implementations and results.

2.1. Sensor choice suite

In order to provide a consolidated and unified solution for the three robots, the choice of the sensors has been agreed to be common across platforms as far as possible. Unfortunately, some sensors are very difficult to integrate or acquire due to the different nature of the platforms. In this section, the common sensors are presented in order to discuss the integration approach in the following sections. Details for the 2D LiDARs and the VI-sensor can be found in deliverable D5.2, Section 5.1.

2.1.1. RealSense D435

As described in deliverable D2.1-Sensor Specification, the Intel RealSense D435 (see Figure 17) was chosen due to the versatility and field of view (see Table 4).

Figure 17. Front, rear, up and down views of the Intel RealSense D435

Table 4. Intel RealSense D435 specifications.

Features	**Use Environment:** Indoor/Outdoor **Image Sensor Technology:** Global Shutter, 3µm x 3µm pixel size	**Maximum Range:** Approx. 10 meters. Accuracy varies depending on calibration, scene, and lighting condition.
Depth	**Depth Technology:** Active IR Stereo **Depth Field of View (FOV):** 87°±3° x 58°±1° x 95°±3°	**Minimum Depth Distance (Min-Z):** 0.105 m **Depth Output Resolution & Frame Rate:** Up to 1280 x 720 active stereo depth resolution. Up to 90 fps.
RGB	**RGB Sensor Resolution & Frame Rate:** 1920 x 1080 **RGB Frame Rate:** 30 fps	**RGB Sensor FOV (H x V x D):** 69.4° x 42.5° x 77° (+/- 3°)

Major Components	**Camera Module:** Intel RealSense Module D430 + RGB Camera	**Vision Processor Board:** Intel RealSense Vision Processor D4
Physical	**Form Factor:** Camera Peripheral **Length x Depth x Height:** 99 mm x 25 mm x 25 mm	**Connectors:** USB-C* 3.1 Gen 1* **Mounting Mechanism:** One 1/4-20 UNC thread mounting point. Two M3 thread mounting points.

2.2. Pepper robot

2.2.1. Realsense D435

The H2020 European Project MuMMER (MultiModal Mall Entertainment Robot) triggered the development of the D435 sensor integration. In consequence, the CROWDBOT project aligned with this requirement in a similar time period. Here, a sensor location feasibility study, the different iterations in design and the results, are shown.

2.2.1.1. Feasibility study

This section presents the feasibility study carried out to define the most suitable location for placing the D435 sensor considering as main requirement maximizing the human detection and tracking. Possible locations are summarized in Figure 18 and evaluated with different distances to a human target.

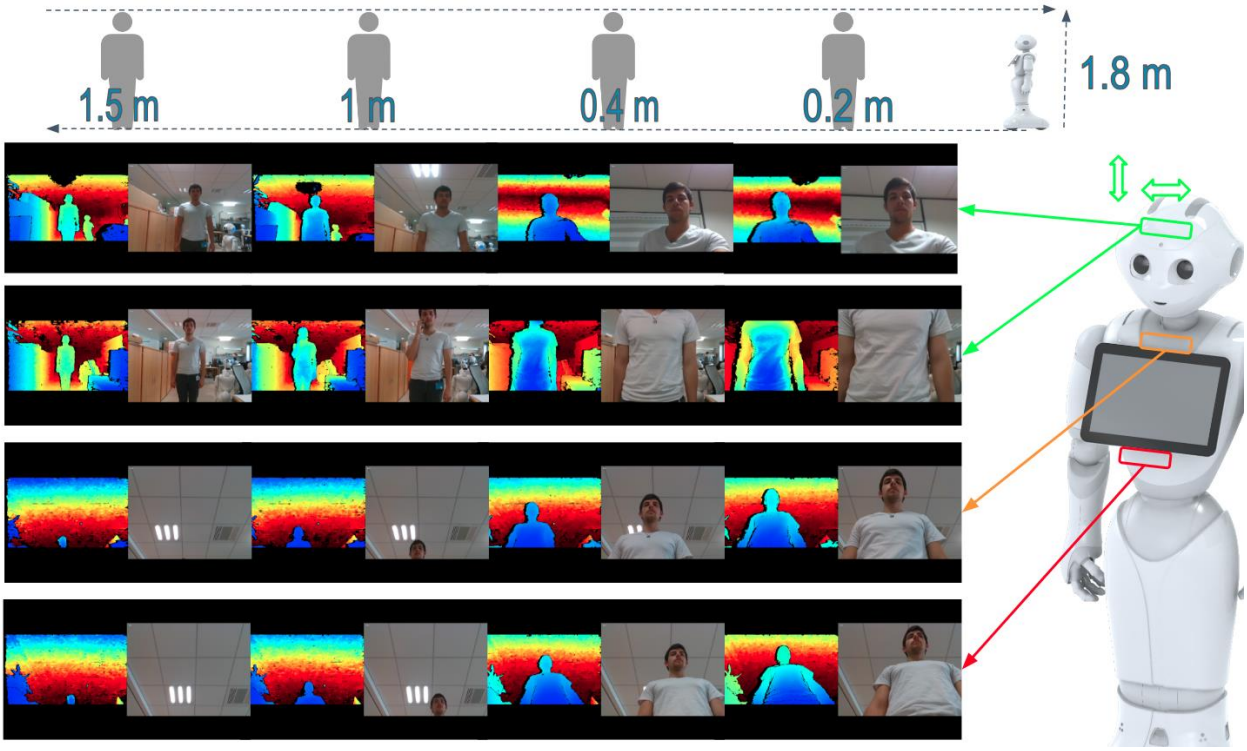


Figure 18. Visual summary of the feasibility study for placing the D435 sensor on Pepper

Despite the fact that the placement of the camera on the front head impacts heavily Pepper's design and increases the size of the top skull, it becomes the location that brings the best field of view for tracking people.

As can be seen in Figure 18, the perception of humans increases substantially in a high position, so the head was selected as the preferred location.

2.2.1.2. First prototype

The first prototype of the D435 integration was proposed with the primary objective to minimize the sensor impact in the original Pepper's head design, by making use of the empty space within the head. A 90° cut is performed in the forehead that provides enough room to place the sensor while keeping the default roundness of the surface. When the D435 is placed in the bay, 2 M3 screws that go through the forehead surface keep it fixed while resting in two small flat lifters to mitigate vibrations (see Figure 19).

Figure 19. First prototype of Pepper's redesigned scalp

Unfortunately, during the initial tests performed on the integrated D435 sensor, a substantial raise in the temperature of the sensor is detected due to the obstruction of the lower ventilation openings. Considering the sensor's passive cooling, this design led to unexpected responses such as low frame rate acquisition or quality image reduction.

2.2.1.3. Second prototype

The redesigned prototype allows the exposure of the upper and lower ventilation areas while facilitating the USB cable installation under the surface till the nape, as can be seen in Figure 20. This design also makes use of 2xM3 screws that go through the forehead surface in order to keep the sensor fixed, but also of one lifter in between the two lower ventilation areas.

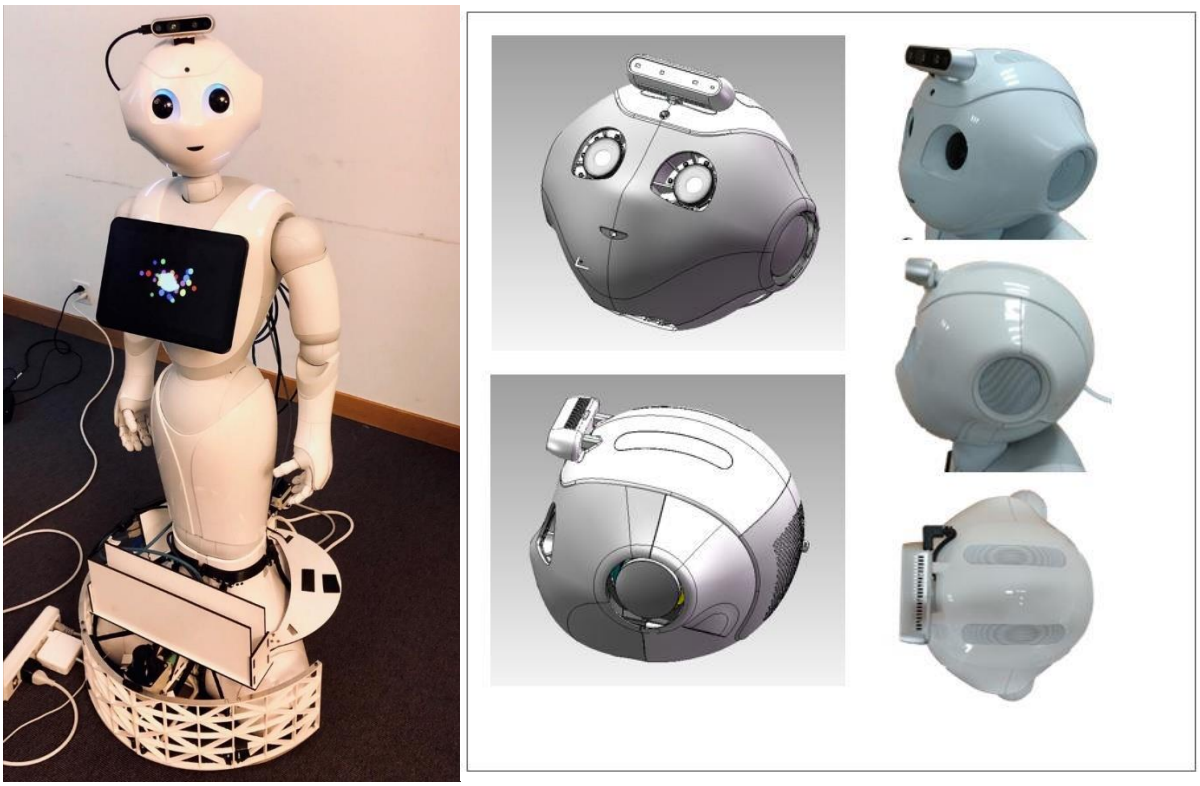


Figure 20. Second prototype of Pepper's redesigned scalp

The second prototype complies with the heat working limits of the RealSense D435. In this case the three areas monitored during testing, apart from the ambient temperature, are: the camera upper ventilation opening, the rear left and right of the minimal area in proximity with the surface of the robot.

Finally, a ROS tf frame needs to be defined within the robot *.urdf* (Unified Robot Description File) considering the position of the sensor, as follows:

```
<link name="camera_link"/>
<joint name="realsense" type="fixed">
  <parent link="Head"/>
  <child link="camera_link"/>
  <origin rpy="0 0 0" xyz="0.0715 0.0175 0.195"/>
  <axis xyz="0 0 0"/>
</joint>
```

2.2.1.3.1. Heat dissipation

The passive dissipation of the heat generated by the sensor has been target of an in-depth analysis since the design of the first scalp prototype proved to be very inefficient in this regard. During the iterative design process few heat tests were carried out in order to monitor if the sensor remained within the manufacturer's preferred working limits. In Figure 21, the test bench is set with three temperature sensors placed in the rear and top of the D435. In addition, one sensor shows the ambient temperature.

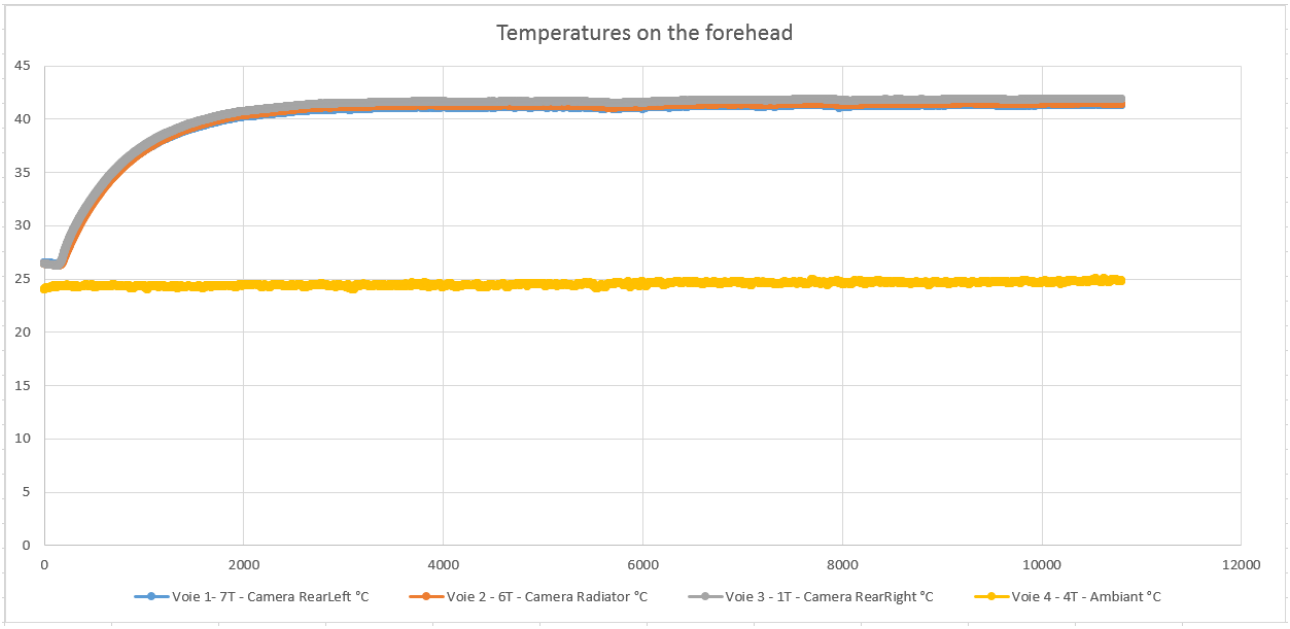


Figure 21. Readings of the three temperature sensors, in degrees celsius, placed on the camera during an exposure time of 3 hours

2.2.2. BAKI tensioner

2.2.2.1. Problem statement

An important mechanical constraint when dealing with budget robots, is the loss of motion between two gear teeth, or backlash (see Figure 22). In other words, the motor torque $\boldsymbol{\tau}_M$ is not transmitted regularly to the link, $\boldsymbol{\tau}_L$, creating a discrepancy between both. Such nonlinearity is often negligible in case of precision industrial robots but largely present in personal robots such as Pepper.

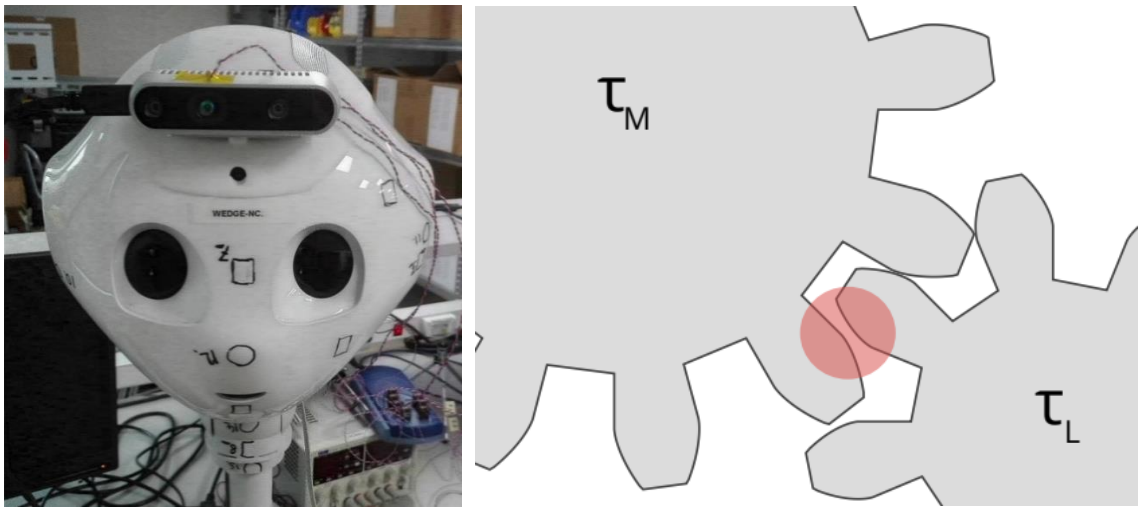


Figure 22. Representation of the backlash between motor and link gears

This mechanical property can be observed not only when modifying a joint position,but also when displacing the base; due to possible irregularities on the surface of the floor or sudden changes in acceleration. For instance, Pepper's head pitch joint backlash is quantified around 3.5°, impacting the image acquisition of the RGB and depth sensors placed on the head. However, a more significant issue to be addressed is the misalignment that takes place between two custom sensors essential for autonomous navigation as described

in *D52 (First basic integrated robot system prototype in experimental scenario*): the VI-Sensor, located in the head, and the 2D LiDAR, located in the base. In this regard, minimizing the clearance between the joints will not only benefit the mapping (see Figure 23) but also the localization. And overall provide a greater robustness and precision during motion.

2.2.2.2. Approach

To mitigate the backlash of Pepper's head pitch joint on the navigation stack, a small tensioner component that attaches to the back has been designed in an iterative manner. Such iterations start with a proof of concept and evolve till a fully working prototype in this deliverable, in order to be the best fit for the current setup.

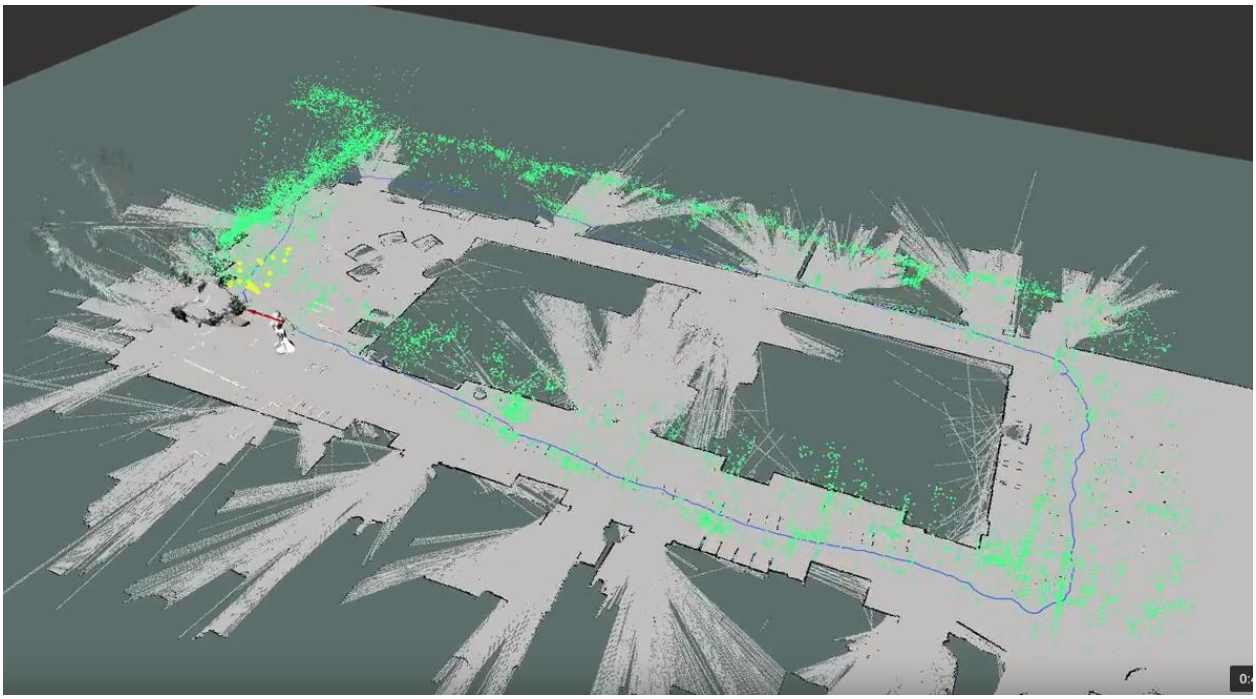


Figure 23. Combined VI-Sensor (green) and 2D LiDAR (black and gray) mapping and localization. Yellow points show matched features found using ROVIOLI. The RGB and depth data are also projected in the scene.

The principle exploited consists of increasing the torque of the joint by adding additional resistance. As a consequence, a contact between the teeth of both gears occurs, reducing the gap between them. A simplified set up to achieve the previous can be seen in Figure 24 where a semi-rigid cable is attached from the existing openings of Pepper's nape to the tensioner placed on Pepper's back.

This proof of concept was enough to see a potential benefit of this approach, although a more refined and evolved concept for a tensioner needed to be proposed in order to deal with the default booting procedure of Pepper: when the robot shutdowns the neck joint moves the head towards the torso till the chin rests on the tablet. On the contrary, when the robot is booting, the joint places the head vertically aligned with a standing body after performing a predefined movement. Therefore, an automatic plug-and-play system that could work completely unattended was required in order to plug and unplug while booting and shutting down, respectively.

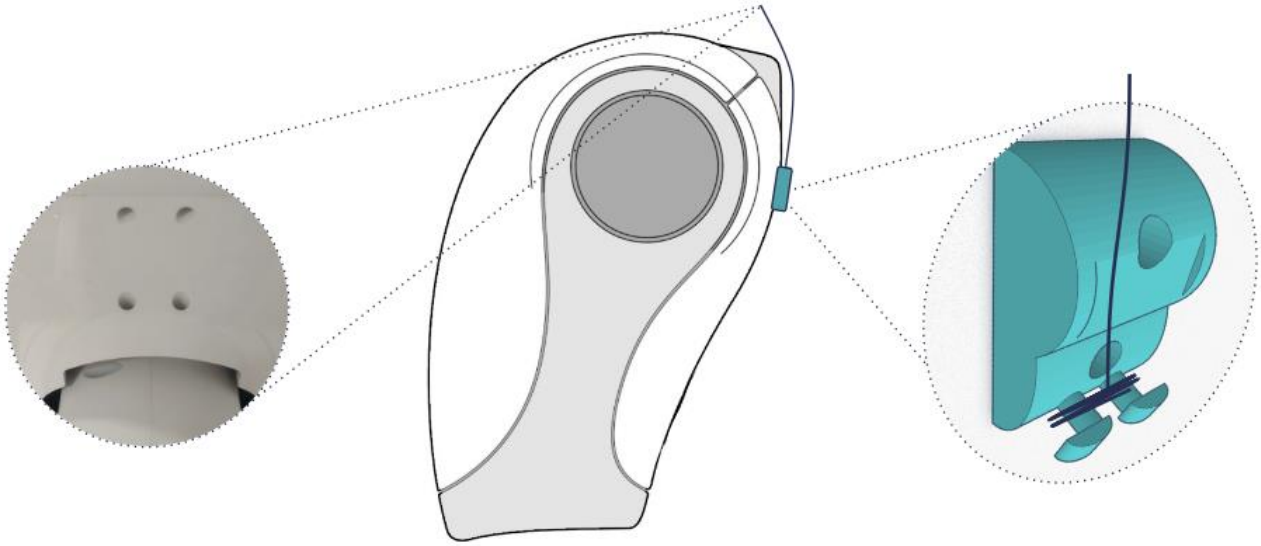


Figure 24. Components for the proof of concept of the tensioner. Attachment of the semi-rigid cable from the holes placed in the rear of the head and the bracket.

2.2.2.3. Tensioner v0.1

Once the proof of concept showed the benefits of the approach, a more suitable hardware was designed in order to satisfy the constraints of the platform during booting and shutting down. Pepper's kinematic configuration when off or idle, places the joints in a safe state; in the case of the head joint, the head ends up resting on the upper frame of the tablet. On the contrary, when the platform is booting, the robot performs a predefined movement looking at the left and right of its base while that one rotates over its z-axis. Considering the constraints mentioned, the primary goal in terms of design was to find a suitable choice of components so that the system can work completely unattended. In this regard, several choices were tested eventually converging to the following to the components shown below:

- Semi-rigid cable: The choice of this component facilitates the encounter of the two different parts that compose the tensioner. In this case, a 0.405mm thick electric cable was used.
- Spring: This component allows to mitigate the overheating generated by the joint when compensating the force exercised by the tensioner. A long list of different spring types are compatible with the setup. However, the choice based on availability was a 30 coils spring with hook ends, a length of 2cm and a wire diameter of 0.4mm.
- Holder: First part of the system that is able to hold a magnet inside and attaches to the spring.
- Magnets: Both magnets are components already used in Pepper robot to hold parts of its outer covers.
- Tensioner: Second part of the system that holds the other magnet and attaches to the back cover of Pepper using 2 x M1.5 bolts.

A mockup of the different components all together is shown in Figure 25. Unfortunately, the present design didn't account for the direction of the tension pulling the holder, which becomes 90° with respect to the magnetic force. In consequence, the magnets were sliding easily instead of popping.

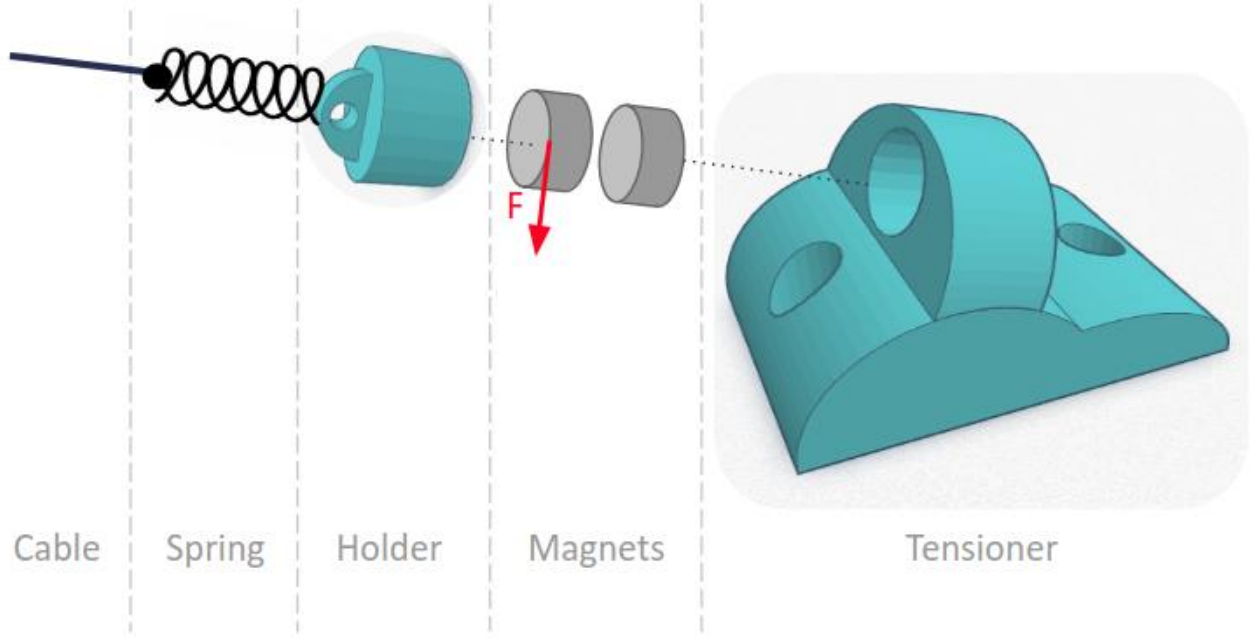


Figure 25. Components used in tensioner v0.1

2.2.2.4. Tensioner v0.2

The second prototype version of the BAKI tensioner aimed to fix the issue with the magnet adhesion by conducting the spring in a way the tension (T) coming from the head joint becomes opposite to the magnetic force (F) as seen in Figure 26. In this case, a 130° facilitated by the tensioner shape is needed to achieve that (see Figure 27). In addition, that will provide enough clearance to the semi-rigid cable to contact the robot surface.

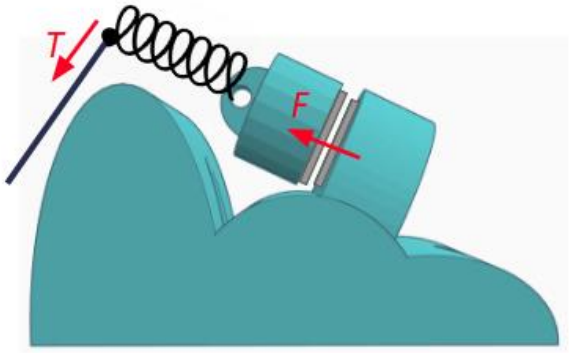


Figure 26. Tensioner v0.2 prototype with a shape that solves the magnet adhesion issue

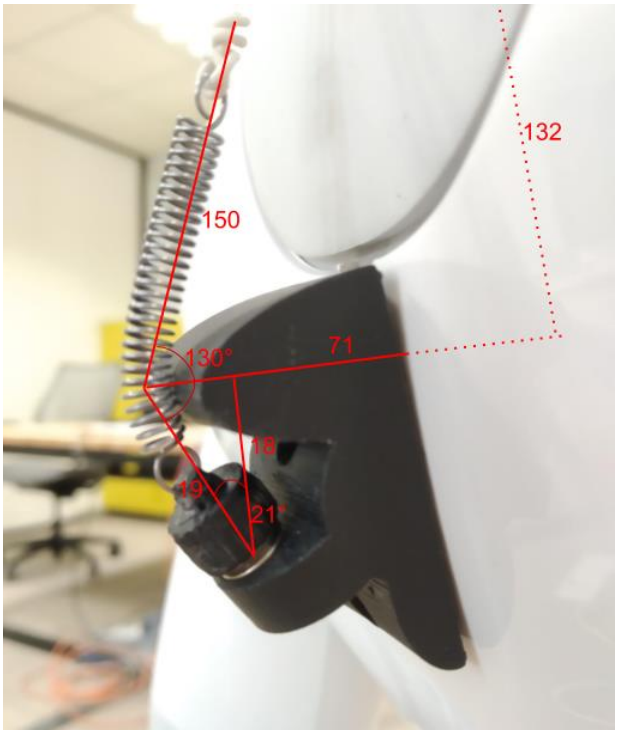


Figure 27. Tensioner v0.2 distances and angles to ensure adhesion and avoid the cable contact with the surface of the robot

2.2.2.5. Results

The results acquired testing the v0.2 are very positive in terms of backlash mitigation. Preliminary tests were carried out where the robot wheelbase was displaced at maximum speed and acceleration to break after one second, to finally perform the same movement backwards. As can be seen in Figure 28, a reduction around 81% in comparison with the default system is observed in the head pitch joint. In other words, a gap reduction from 2.30° to 0.28° (bottom right corner). In addition, despite the increment in torque values (top left corner), no issue is observed by using the proposed approach.

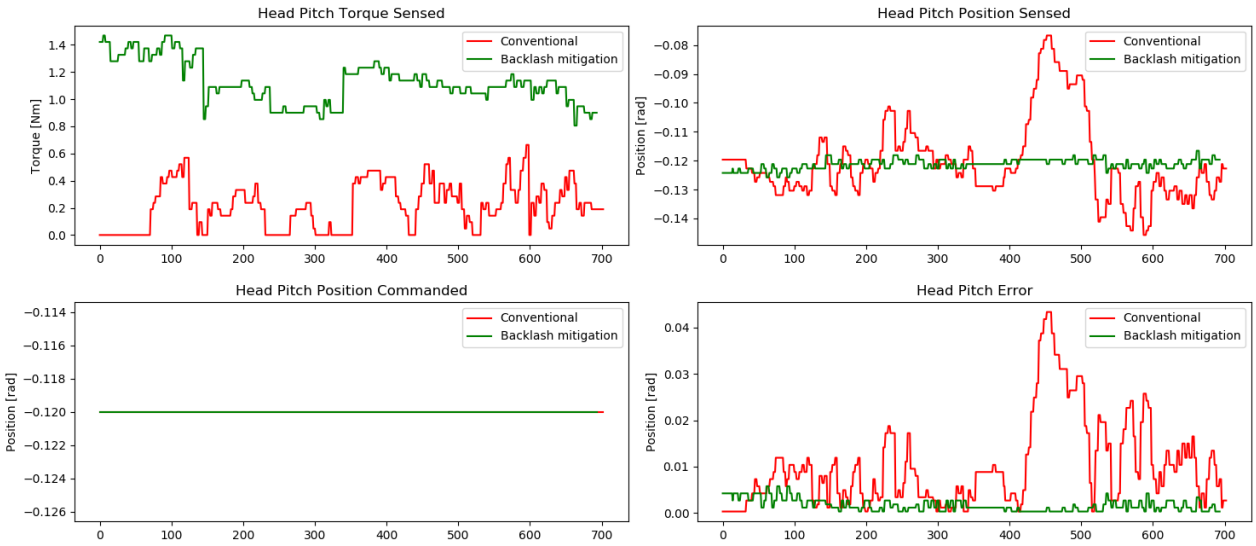


Figure 28. Backlash reduction results acquired during testing with and without the BAKI tensioner v0.2. Conventional system (red) is shown in comparison with a backlash mitigation (green).

2.2.3. 3D printed fender

Crowdbot Pepper will be tested in scenarios that involve navigating in close proximity with humans. To protect both human participants as well as the robot and its additional sensor mounts, a front guard was designed, constructed and installed on Pepper. The design includes a rigid aluminium base plate connected to a 3D printed brace that fits snug beneath Pepper's knee joint. Several trusses connect the front guard to the brace such that all compressive loads are directly transferred to the base and do not pass on to the sensors. Figure 29 shows the CAD models of the fender and brace designs. Figure 30 shows the complete setup as mounted on Pepper.

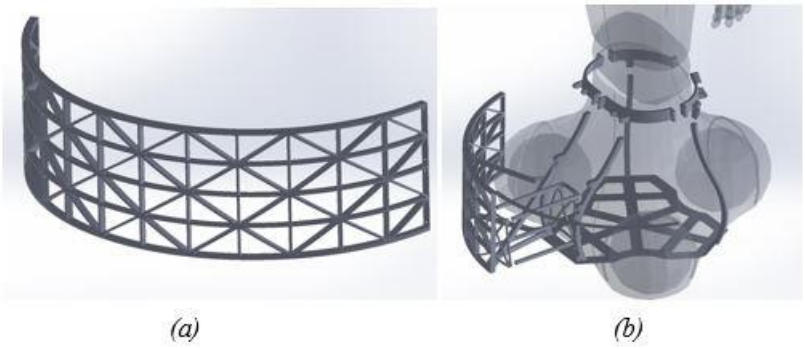


Figure 29. (a) CAD model of the front guard installed around Pepper's base. The fender was designed to be lightweight but structurally rigid. (b) Installation of Pepper's front guard. Only half of the fender is shown in to highlight the underlying structural support.

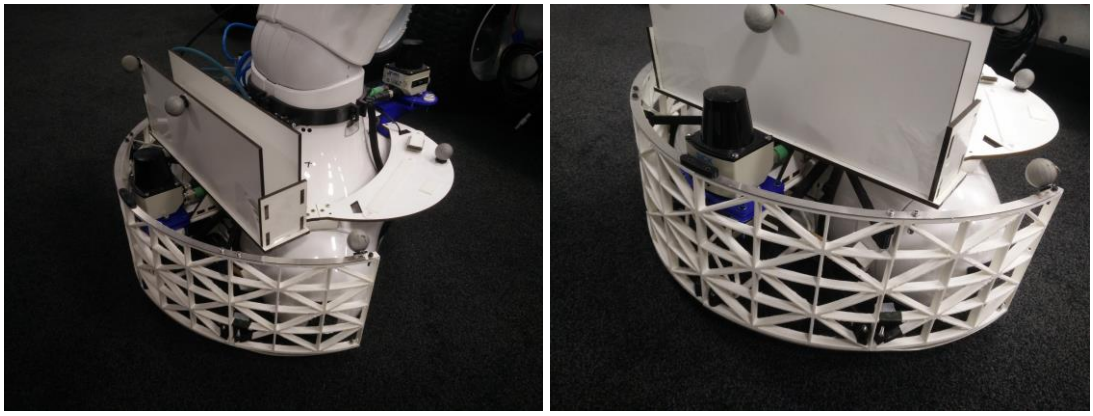


Figure 30. The full setup and installation of the fender and brace, showing isolation from the sensor mounts

2.3. Wheelchair

The wheelchair control system runs across two computers; a gaming laptop used for image processing and more CPU intensive operations and an onboard Odroid X4 which interfaces with the motor controller and low-level Arduino based sensors. The RGB-D sensor and the LiDAR both connect directly to the gaming laptop, whereas all the other sensors are connected to the Odroid, some of which are pre-processed by Arduinos. An overview of the hardware integration on the wheelchair is shown in Figure 31 and we discuss each of these components throughout this section, finishing with a discussion of the initial safety tests.

Figure 31. The original CAD drawings and the integrated robotic wheelchair platform. All the hardware (emergency stop button, ultrasonic sensor clusters, joystick controller) that was used for the initial safety tests is also shown.

2.3.1. RealSense RGDB Camera

The Intel RealSense camera captures images and depth information from the surrounding environment for use in tracking and route planning algorithms. We have constructed a custom-built frame (Figure 32), which is mounted on the rear of the wheelchair to position the RGB-D sensor above the user's head. The RGB-D sensor is connected to the gaming laptop.

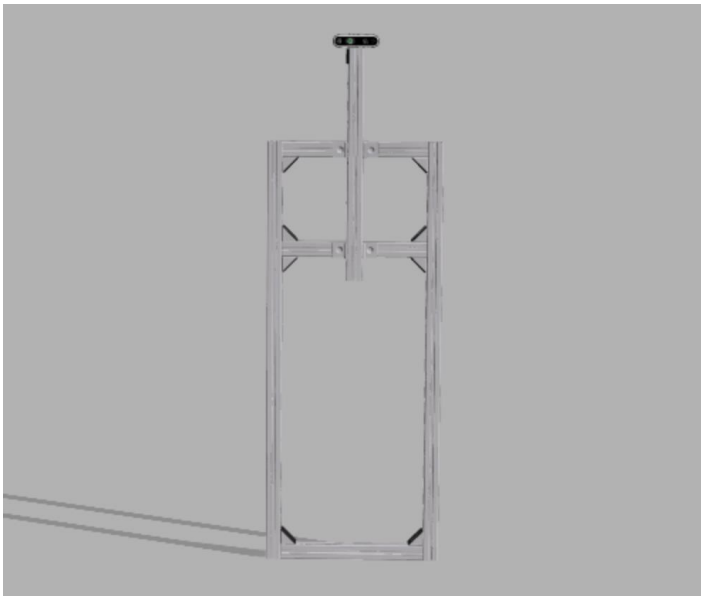


Figure 32. The Intel RealSense camera, mounted on a custom-built frame, which is then attached to the rear of the wheelchair

2.3.2. Hokuyo LiDAR

The LiDAR (Figure 33) captures range information to aid with object avoidance, tracking and route planning and is a fundamental sensor that inputs into the RWTH people tracker. The LiDAR is physically mounted on the wheelchair footrest and is connected to the gaming laptop.

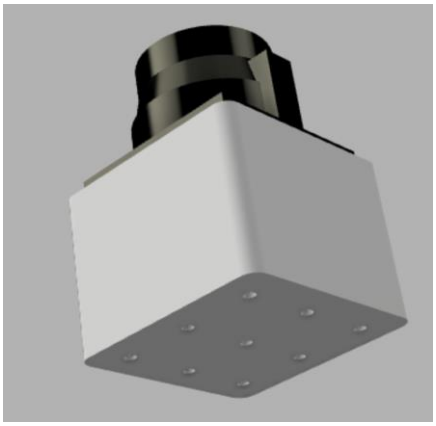


Figure 33. CAD model of the Hokuyo URG-04LX-UG01 Scanning Laser Rangefinder

2.3.3. Wheel Encoders

Both wheel encoders (see Figure 34) are connected to an Arduino Mega which estimates the odometry and publishes twist and pose ROS messages to help keep track of the speed and position of the chair. The Arduino sends ROS messages over USB serial to the Odroid.

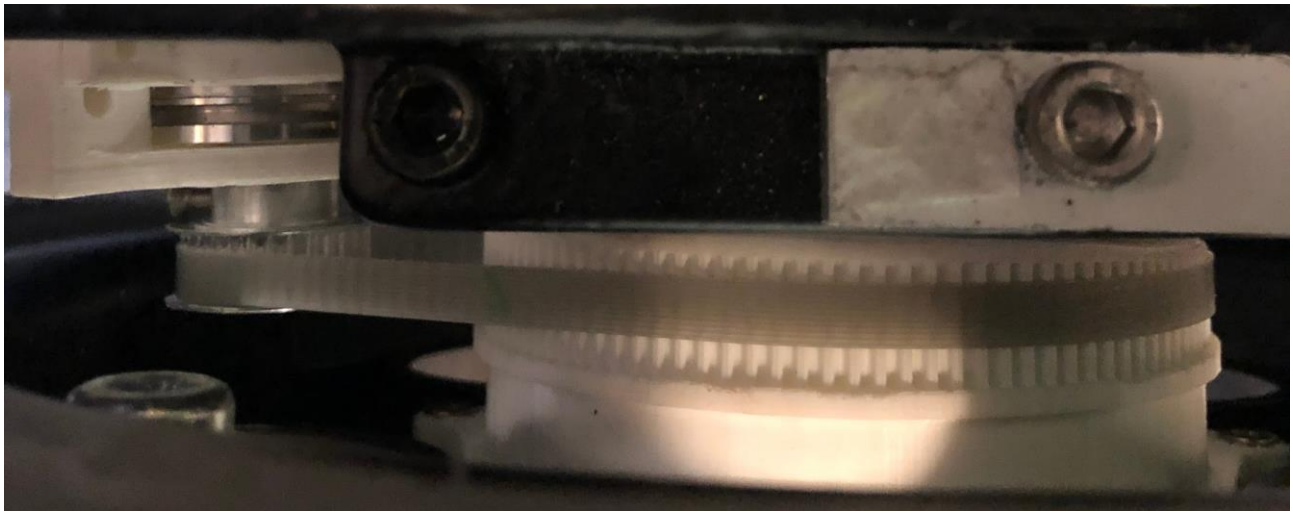


Figure 34. Custom built wheel encoders are installed on the wheelchair

The wheel encoders are mounted on the chair using a 3d printed housing, they are then connected via a pulley to a 3D printed cog (Figure 35) attached to the wheel.

Figure 35. 3D printed wheel encoder cog

2.3.4. IMU

The IMU (SparkFun 9DoF Razor IMU M0) provides angular velocity, linear acceleration and orientation data. It is used in conjunction with the wheel encoders via a Kalman filter to improve the odometry estimate and also provides the ability to sense slopes and inclines. A 3D printed casing was designed to help protect the sensor and mount it on the chair, as shown in Figure 36. The IMU connects to the Odroid via USB serial.

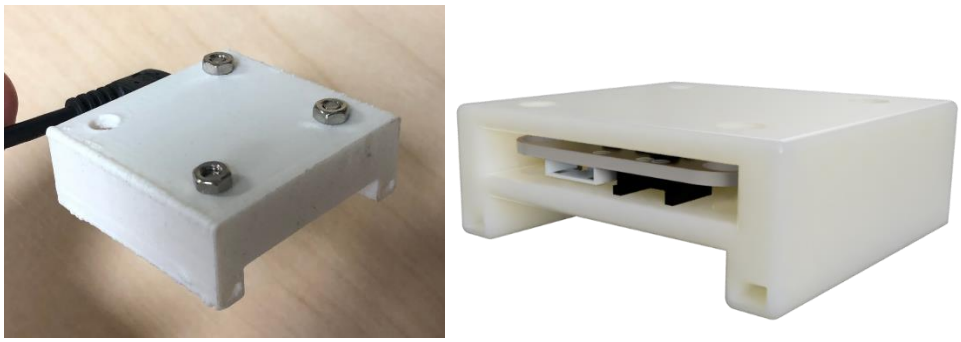


Figure 36. IMU in custom 3D printed housing, to mount securely on the wheelchair chassis

2.3.5. Sonar clusters

The ultrasonic sensors are used to detect proximity to nearby obstacles to help avoid collisions. The sensors are arranged in clusters of 3 at each corner of the wheelchair (see Figure 37). Arduinos are used to read data from the sensors and publish readings as ROS messages. One Arduino reads the front two clusters and one the rear two. Each Arduino publishes a separate ROS node, sonarFront and sonarBack.

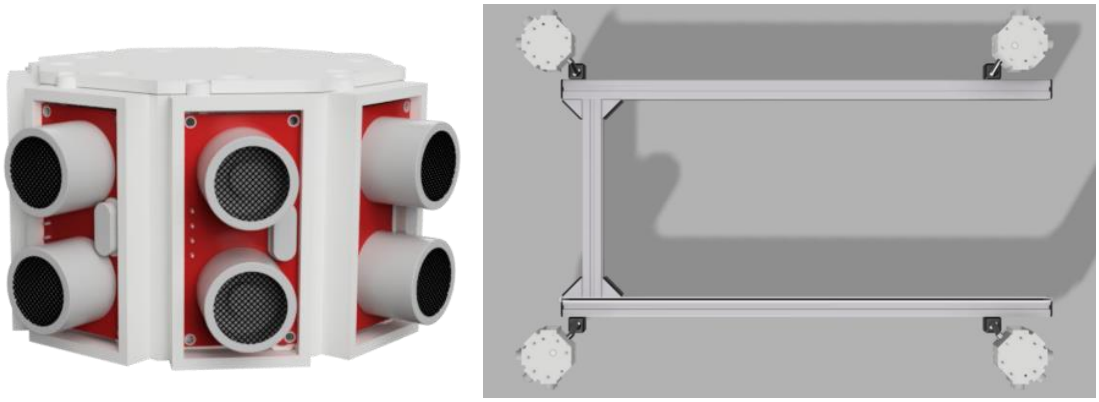


Figure 37. Left – CAD rendering of our ultrasonic sensor cluster. Right – CAD rendering of four sensor clusters, mounted at the four corners of the sensor frame, which in turn is affixed to the wheelchair chassis.

2.3.6. R-Net joystick integration

The main drive interface of the Salsa M^2 quickie wheelchair is an R-Net joystick controller. The joystick interfaces with the motor controller which in turn sends drive commands to the motors propelling the drive wheels. To drive, the wheelchair user must provide a continuous joystick input, letting go of the joystick brings the wheelchair to a complete stop. To drive the wheelchair using shared control the user must still provide a continuous input to the virtual joystick controller. When no input is provided the wheelchair stops moving.

This virtual joystick controller was programmed to function in two distinct modes namely the crowdbot shared control mode and the manual manufacturers mode. The manual mode comes with four different profiles each with five different velocity acceleration selections. The minimum and maximum forward driving speeds for each profile in manual mode are summarised in Table 5 below. Note that profile 3 is reserved for crowdbot shared control mode.

Table 5. Forward driving speeds for each mode of the R-Net joystick controller

	Crowdbot	Profile 1	Profile 2	Profile 4	Profile 5
Forward Max	1.4 mph	1.4 mph	1.6 mph	3.3 mph	4 mph
Forward min	0.8 mph	0.8 mph	0.6 mph	1.0 mph	1.2 mph

The virtual joystick provides an R-Net interface allowing the wheelchair to be driven via ROS velocity commands. The architecture of this virtual joystick interface implemented via a Tiva microcontroller which interfaces with the drive control unit built into the wheelchair (via the R-net bus) and connects to the Odroid via USB. Essentially it forms a proprietary interface that is used to read the joystick messages from the R-Net

bus and place motor velocity messages back on to the R-Net bus, which in turn are fed to the motor controller unit.

It is important to note that in order to send a velocity command to the motors a continuous input must be provided by the user via the joystick interface. This is indeed the case in the shared control semi-autonomy navigation paradigm.

2.3.7. Hardware and software safety tests

We conducted three levels of safety tests for the robotic wheelchair namely manufacturer's, mechanical, and software safety checks.

The manufacturer's safety test considered the performance of the joystick in terms of stopping time and distance (Table 6). First the normal drive stop was tested (here the wheelchair user simply releases the joystick while driving to bring it to a stop). Next the joystick emergency stop was tested (here the wheelchair user pulls back hard on the joystick to bring it to a swift halt).

The mechanical safety test evaluated the response time and distance travelled when the emergency stop button was activated (which we added to the wheelchair). Pressing the emergency stop button sets the mode of the motor control unit into emergency drive inhibit, as per the manufacturer's guidelines, releasing the button returns control to normal mode.

For the software safety tests we implemented a virtual bumper using the array of ultrasonic sensors mounted on the wheelchair base (see Figure 30) which causes the wheelchair to come to a complete stop, if an obstacle is detected within a manually set threshold detected by the sensor.

Table 6. Stopping distance and time for crowdbot mode

	Joy	Joy emergency	Emergency button
Distance (cm)	40	30	37
Time (s)	1.41	0.6	0.63

Safety test results for all categories are shown in the figures below.

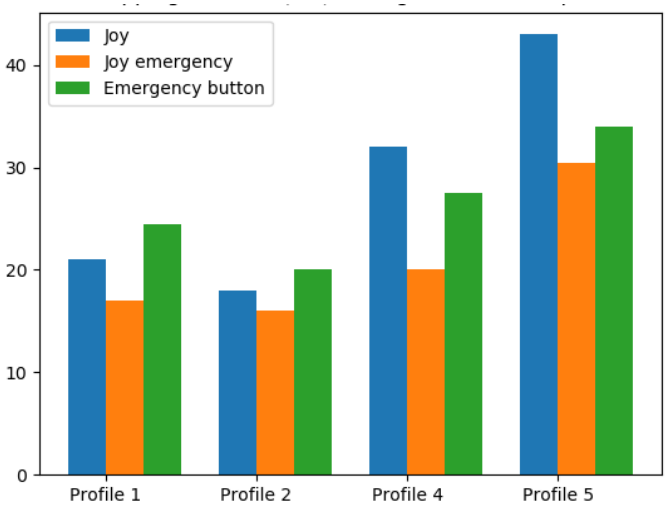


Figure 38. Stopping distance (cm) driving at minimum speed

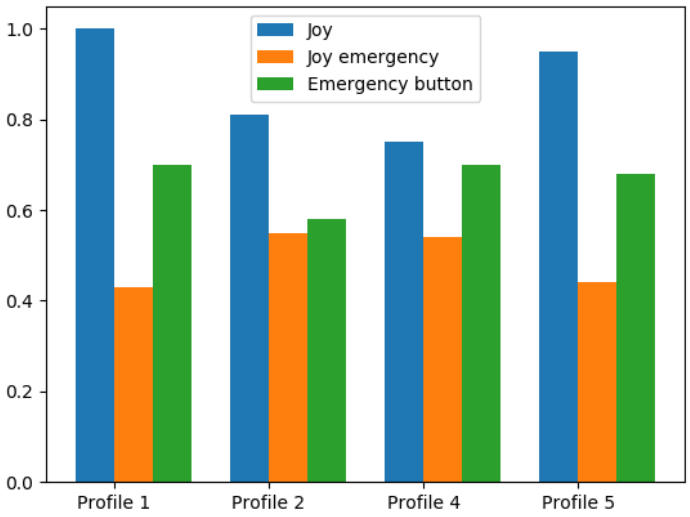


Figure 39. Stopping time (s) driving at minimum speed

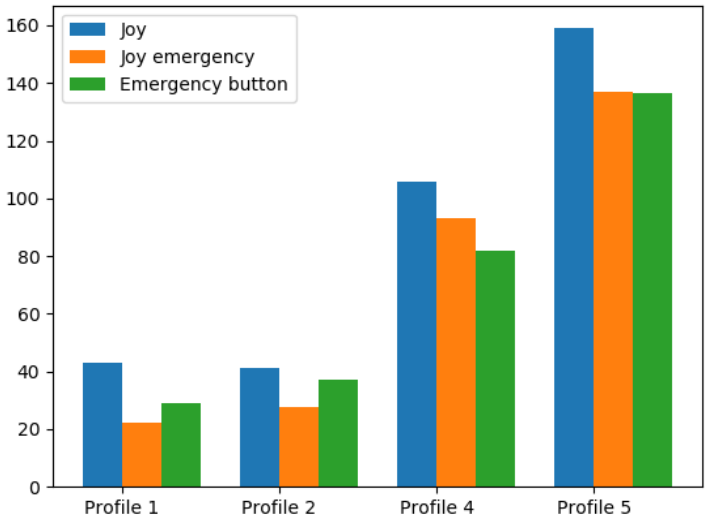


Figure 40. Stopping distance (cm) driving at maximum speed

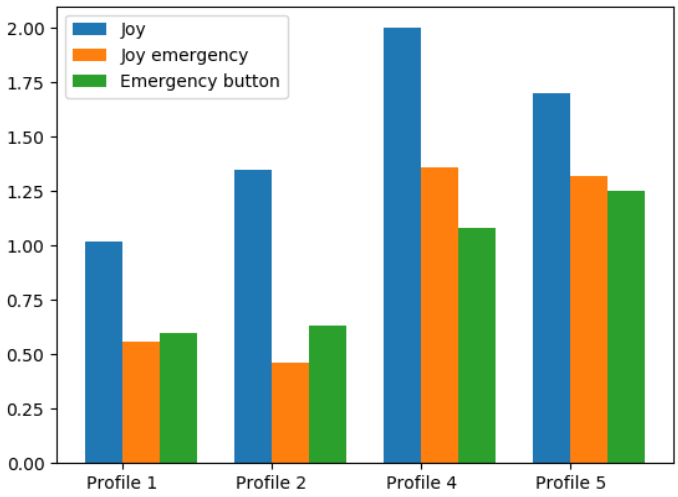


Figure 41. Stopping time (s) driving at maximum speed

Both the stopping distance and stopping time, when operating using the Crowdbot controller, are comparable with profile 1 of the manufacturer's default configuration and indeed are lower than that of profile 5. Therefore, we are confident that the responsiveness of the Crowdbot mode does meet baseline safety requirements.

2.4. CuyBot

The cuyBot robot (see Figure 42) used in the project follows the sensor recommendations given in CROWDBOT Deliverable D6.2 *Robot Design Recommendations*. The basic hardware setup as described in D6.2 consists of the following parts:

- Robot base with:
  - 4 SmartWheels,
  - 1 or 2 LiDARs,
  - Electronics,
  - Hull with force sensors.
- Trunk with head:
  - 2 RGB-D cameras,
  - Touchscreen,
  - Loudspeaker.

Figure 42. CuyBot robot

In general the robot can be equipped with

- One or two 2D LiDARs at the bottom front and back,
- One or two Intel RealSense D435 cameras at the top front and back.

Following the recommendations in D6.2, two Intel RealSense D435 as well as two LiDAR sensors are installed by default. Since there is a lot of open space on top of the robot additional sensors could be added, as discussed already in Deliverable D6.2, in order to further support people tracking or localization in CROWDBOT.

It should be noted that the definition of front and back of the robot is somewhat arbitrary and can be chosen according to the application. The outer shape of the robot's base is symmetrical, and the trunk with the head can be mounted on either side. Therefore in CROWDBOT we will try out both combinations, the trunk either at the front or at the back. The main difference, except for the appearance, is that the field of view of the RGB-D cameras will be slightly different.

2.4.1. RealSense D435

Two Intel RealSense D435 cameras are located at the top of the trunk below the head of the robot, facing in opposite directions, at heights of 96 cm resp 100 cm above ground. The position of the sensors remains constant relative to the robot's base, except from possible vibrations during driving (see Figure 43Figure 43).

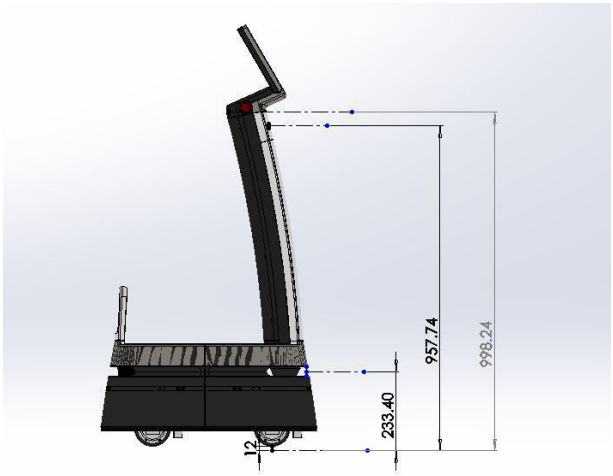


Figure 43. Heights of LiDAR (bottom) and both RealSense D435 (top) sensors

If the trunk is placed at the front side of the robot's base, the forward-facing camera looks ahead of the robot, see Figure 44. Objects or persons very close to the robot might not be in the field of view of the camera. If instead the trunk is placed on the back side, the forward-facing camera looks over the robots base. Therefore it covers a bit more space on the sides. Also the minimal distance of 10.5 cm below which no distance data is received does not play a role anymore. Any person standing close in front of the robot will be farther away from that camera than 10.5 cm, as long as the person does not lean over the robot.

Figure 44. Field of view of front-facing camera when trunk is placed at the front (left image) or at the back (right image) of the robot's base.

Therefore for the detection of people it seems reasonable to put the trunk at the back, making the camera in the robot's default motion direction cover more space close to the robot and increasing the view for people detection. On the other hand it might seem more natural to put the trunk (with the touchscreen) at the front, so it will be in front when the robot drives around. The eventual effects in practice of this choice will be evaluated during the project.

2.4.2. 2D LiDARs

The cuyBot has two spaces for LiDARs in its base, one bigger and one smaller one. The bigger space can hold up to the size of a Sick LMS100. The smaller one is suitable for sizes like that of a Hokuyo UST-10/20LX. By default a Sick LMS100 is placed at the side facing forward, and a Hokuyo URG-04LX-UG01 at the back. That way the more powerful Sick sensor is the main sensor during navigation as the robot is supposed to mostly drive forward. When two LiDARs are installed, they cover the whole (2D) space around the robot. If only the mentioned Sick sensor is installed, only an area of 270° around the robot is covered.

The option of one or two LiDARs goes well together with the possibility of the new person tracking component provided by project partner RWTH Aachen detecting the legs of people. That component is designed to work with either only one LiDAR sensor, or also with multiple sensors, whatever is available.

One particular aspect for the people tracking is the height of the sensor. This is important, since the motion patterns of legs look differently on different heights and might require relearning of the detector. For cuyBot the scan line of the LiDAR is about 23 cm above ground, which is lower than the 37 cm used when learning the detector.

2.4.3. PC and graphics card

Concerning the PC and graphics card used in the robot there are several options. Initially Locomotec used a PC-board in Mini-ITX format with an Intel Celeron processor. This was a tradeoff made concerning computation speed and very low power requirements. Also it provided a PCI-E interface to which a custom graphics card could be connected

Tests with the navigation software of CROWDBOT indicated that this board will not be sufficient concerning computation power. The alternative will be an Intel NUC with an Intel I7 processor. This models offer much higher computation power than the Celeron in a compact format and also with comparably low power requirements. Unfortunately it costs much more and is more restrictive about which graphics cards can be connected.

The people tracking software on cuyBot was first running with a GeForce GTX 1050 Ti graphics card. This card managed only moderate frame rates and in particular would not be suited for multiple cameras. In CROWDBOT Deliverable D2.2 *Local Sensing First Prototype* the performances of multiple graphics cards with the latest state of the software are presented. Considering these evaluations the NVIDIA Jetson AGX Xavier seems to be a good solution. Compared to custom powerful graphics cards like GeForce 1080 TI it has much lower power consumption. Unfortunately it is also much more expensive than those other cards. Therefore in the project the AGX Xavier will be used, but the best tradeoff between computation power, power consumption and price still needs to be determined.

2.5. Qolo

The Qolo robot as shown in Figure 45, for the project follows the sensor recommendations given in CROWDBOT Deliverable D6.2 *Robot Design Recommendations*.

The main robot setup is as follows:

- Robot base with:
    - Mobile wheelchair system of Qolo.
    - 2 x 3D LiDARs: frontal and rear
    - 3 x Realsense RGBD cameras D435.

- Frontal Force-Torque Sensor

- Person Carrier:
  - Embedded Pressure Control Interface
  - Optional Joystick Controller
  - Emergency switch onboard and remote.

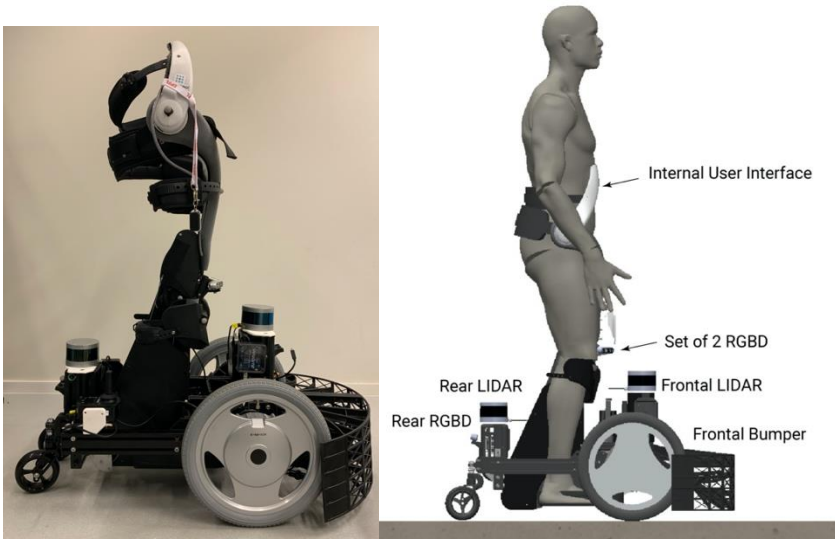


Figure 45. Qolo robot used in CrowdBot project

Following the recommendations in D6.2, three Intel RealSense D435, as well as, two 3D LiDAR sensors Velodyne VLP16 are installed in the robot. Location of the RGBD sensors could be changed accordingly to the desired testing scenario. Currently, two cameras are set forward and one heading rear of the robot.

Moreover, we have included a frontal FT sensor for contact detection attached to a bumper, which could be used for detecting small contact with feet or other surfaces, and potentially controlling for compliance in such scenarios. As a passenger carrier robot Qolo's control interface is made mostly for forward motion, therefore, the sensor is placed in front of the robot.

2.5.1. RealSense D435

Two Intel RealSense D435 cameras are located at the front of the robot facing directions at 114 degrees of one another, at 65 cm above ground. The combined field of view (FoV) of the sensors' RGB (cameras) would cover an approximate 138 degrees of the frontal area, as depicted in Figure 46, left side (blue FoV). While point cloud information from infrared sensors could provide up to 170 degrees of frontal FoV, as shown in the right side (red FoV) in Figure 46.

As well a rear camera set at 35 cm above the ground can be used for detecting incoming people or for using the point cloud proximity data directly behind the robot, with 68 degrees for camera FoV, and 89 degrees for proximity data.

The position of the sensors remains constant relative to the robot's base, except from possible vibrations during driving.

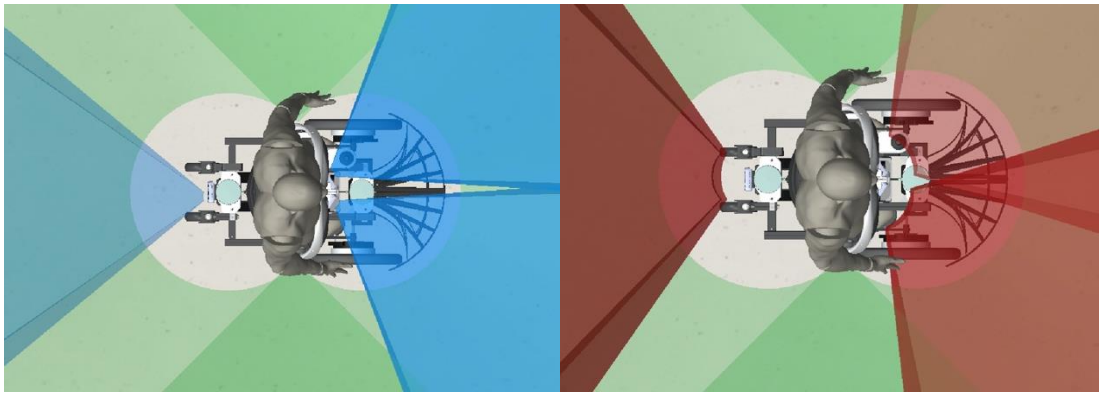


Figure 46. Sensors location in the robot Qolo, with frontal and read RealSense D435 (blue and red) field of view, and front and rear LiDAR in green

The sensor's location was moved further up than initially proposed in D6.2, for increasing the visibility of the upper body of a person in-front of the robot. It is estimated that at 1 meter ahead of the robot any person below 170 cm of height will be in full view of the camera. Thus, enabling closer distance for people detection with the proposed tracker from RWTH.

2.5.2. 3D LiDARs

Qolo, as a standing mobility device for spinal cord injury, carries a person in the middle of the robot, therefore, it makes necessary to use 2 LiDARs mounted in front and rear of the robot, as proposed in D6.2, for gaining full 360 view of the surroundings.

Frontal LiDAR is located at 476mm from the ground, while the rear LiDAR is at 361mm, as shown in Figure 47. The difference in height is due to the need of allowing a user to sit down in a chair, thus the rear LiDAR requires a lower position for permitting docking to a standard chair.

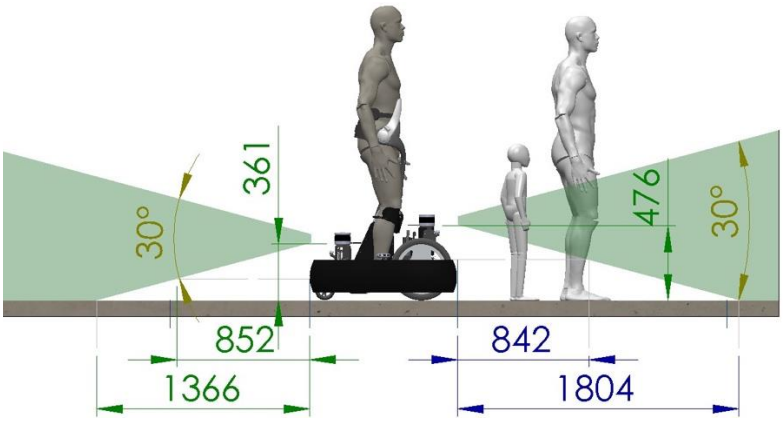


Figure 47. Frontal and rear 3D LiDAR view

An important aspect of the LiDARs' location is the minimal distance at which it would be able to perceive a person. As presented in Figure 47, the current setup gives visibility at the height of the robot's most forward component at 842 mm, and in the rear at 852 mm.

2.5.3. PC and graphics card

The Qolo robot is currently controlled from an embedded computer, UpBoard Squared with a CPU intel Celeron 2.4Ghz, which handles low-level control and user interface inputs. For the integration of the LiDARs

we have set a second UpBoard computer which can run independently of the low-level controller for a distributed control architecture. This second computer is embedded in the LiDAR's base for a compact configuration.

Tests with the navigation software of CROWDBOT indicated that this board will not be sufficient in computational power for running the people tracker algorithm. Therefore, based on CROWDBOT Deliverable D2.2 *Local Sensing First Prototype*, where the performances of multiple graphics cards were presented, we chose a Nvidia Jetson AGX Xavier which seems a good solution with low power consumption. The Nvidia Jetson AGX can be installed in the rear of the robot with its own compact power source, and up to 2 cameras could be used in serial for tracking accordingly to the test performed by the team in RWTH.

3. Experimental Scenario

An initial system test was carried out at ETHZ in June, 2019. The Pepper robot was tasked to navigate in a corridor of roughly 30m x 5m. During navigation, 10-15 crowd participants from the Crowdbot team interacted with the robot in several 1D and 2D flow scenarios (co- (Figure 48), counter- (Figure 49) and cross-current flows (Figure 50)). Localisation, pedestrian detection and motion planning relied only on the LiDAR sensors and was controlled using the software packages described in D3.1. Sequences from the tests are shown below.

During interaction, participants purposefully walked in front of Pepper to qualitatively assess its response time and reactive behaviour. In general, despite Pepper's top speed of approximately 0.3m/s, the navigation system is able to process the LiDAR data and command the base to avoid sudden obstacles in its path without freezing. However, a couple of close proximity human-robot interactions during the tests highlighted the need for multimodal sensing to improve the robustness of person detection. Including the RGBD person detection in the perception pipeline will mitigate this problem. In addition, we also observed that it would be beneficial for the Crowdbot to have responsive backwards movement as part of its avoidance manoeuvres to cater for instances where other crowd participants display more aggressive motion towards the robot. These observations have guided the development of the low latency local motion planner described in D3.1.

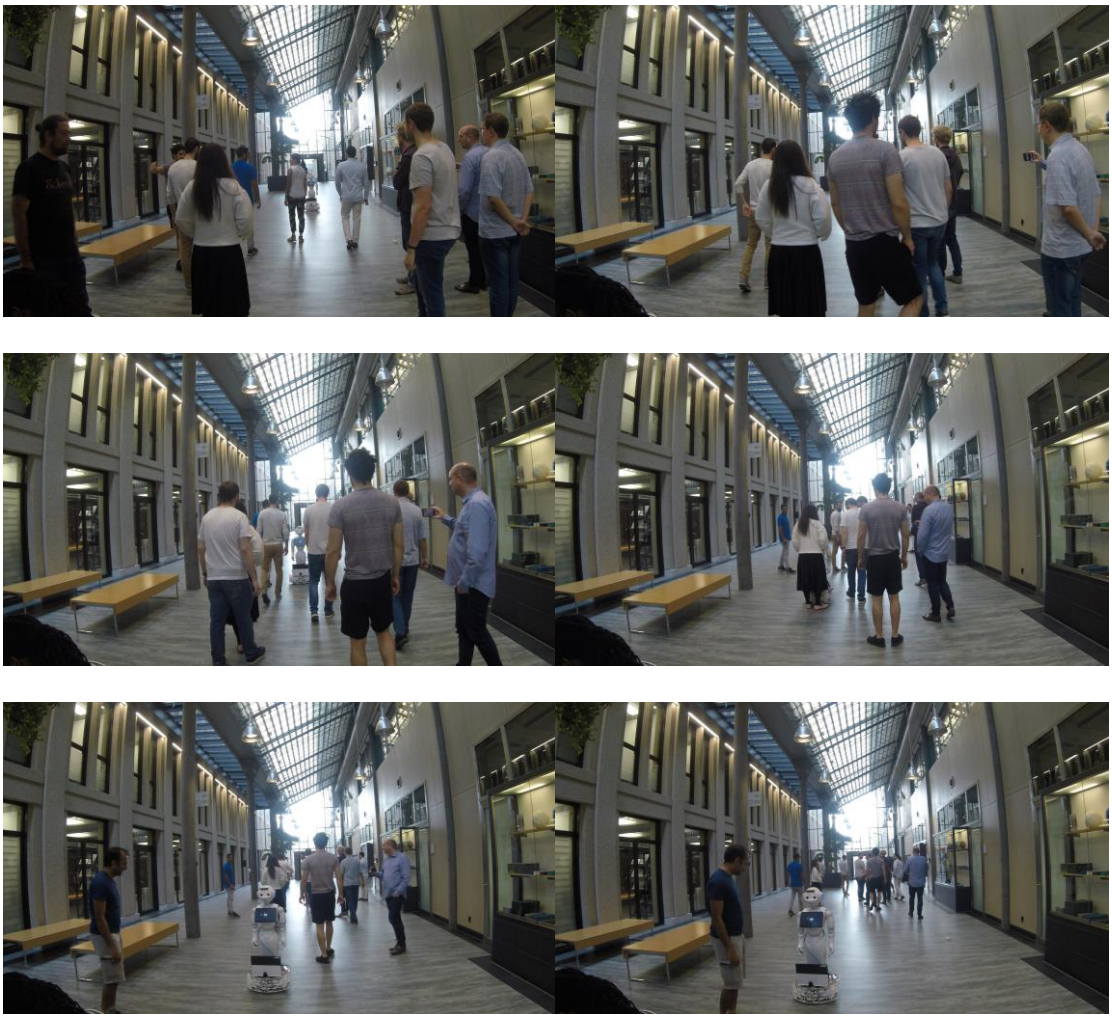


Figure 48. Example of navigation in countercurrent flow

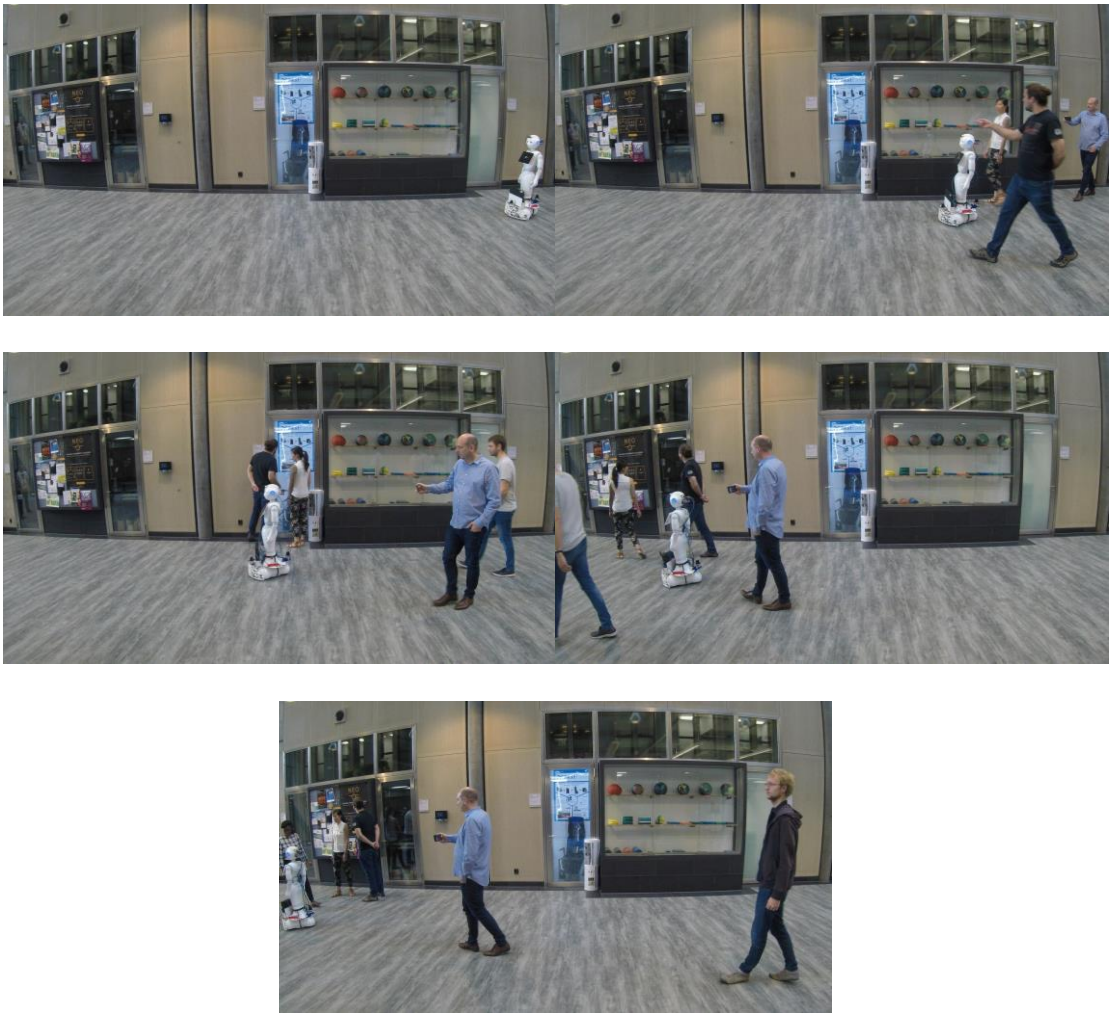


Figure 49. Example of cocurrent flow. Note that Pepper's top speed is approximately 0.3m/s, which is slower than an average human's walking pace

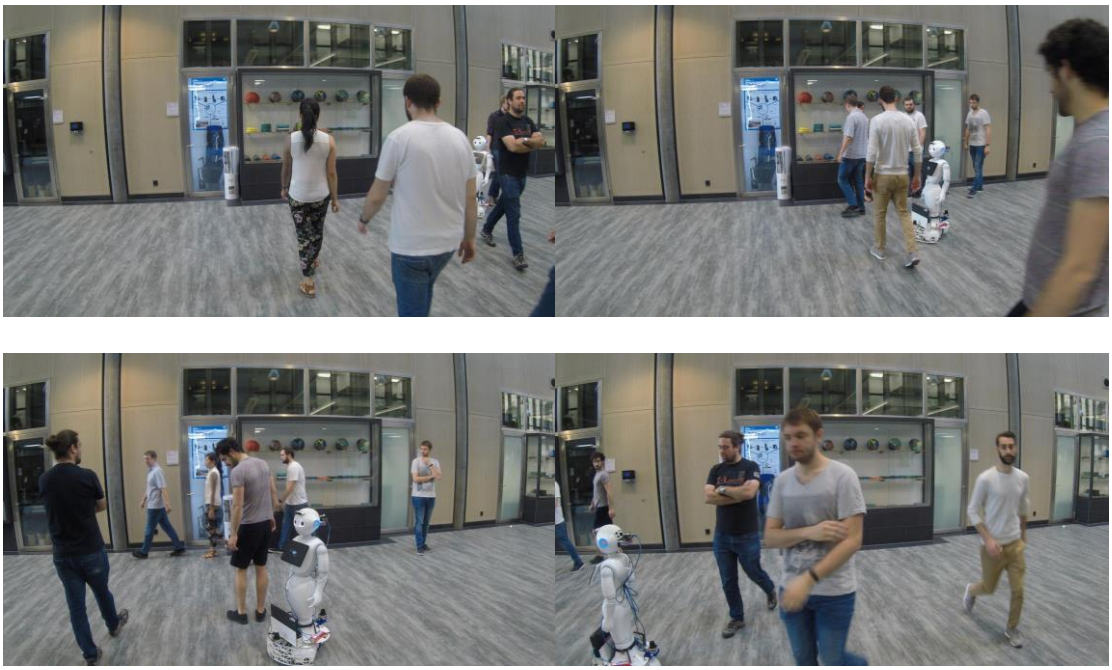


Figure 50. Example of cross-current flow

4. Conclusions

In this work package was described the integration of the different work packages across all three robotic platforms (Pepper, Wheelchair and CuyBot).

In section 1 was presented the software integration. For the Pepper robot, the vanilla controller was described with its current limitations, for which was needed to boost the controller, its implementation can increase the reliability and performance of the wheelbase, which is necessary to ensure a suitable baseline for integration of navigation solutions and reactive navigation approaches that match the dynamics of a crowd. Concerning the Wheelchair, it was presented the ROS nodes and topics used as well as the control system integration. This framework makes it relatively easy to test different planners in our simulator, next steps will involve integrating the ETHZ local planner into the wheelchair platform. The integration of CROWDBOT software on cuyBot is mainly done by exploiting the ROS framework. For simulation of the cuyBot robot a model was provided, this model is still being updated in order to make it more realistic and at the same time to reduce the complexity of its 3D data.

The Hardware integration was presented in section 2. For the Pepper robot, we presented the design of 3 prototypes. The first one, is a design of a scalp on the head of Pepper for positioning the D345 sensor (developed in the context of the H2020 European Project MuMMER). The second, is a small tensioner component that attaches to the back of the robot, designed to mitigate the backlash of Pepper's head pitch joint on the navigation stack. The third one is a front guard on the base of Pepper, the fender was designed to be lightweight but structurally rigid in order to protect both human participants as well as the robot and its additional sensor mounts. For the wheelchair, several components were designed and integrated on the system, including a custom-built frame for the intel real sense camera, the Hokuyo LiDAR, custom-built wheel encoders, 3D printed housing for the IMU, sonar clusters and the R-NET joystick. Three levels of safety tests were conducted for the robotic wheelchair namely manufacturer's, mechanical, and software safety checks. The cuyBot robot follows the sensor recommendations given in Deliverable D6.2 *Robot Design Recommendations.*

Finally, section 3 presents an initial system test, which was carried out at ETHZ in June, 2019. In this experimental scenario, the Pepper robot was tasked to navigate in a corridor of roughly 30m x 5m. In general, despite Pepper's top speed of approximately 0.3m/s, the navigation system is able to process the LiDAR data and command the base to avoid sudden obstacles in its path without freezing. Nevertheless, the observation lead to ideas to improve the navigation system.