Grant agreement No: 101017008

Harmony

Assistive robots for healthcare

Enhancing Healthcare with Assistive Robotic Mobile Manipulation

(HARMONY) | H2020-ICT-2018-20| RIA

Start of the project: 01.01.2021
Duration: 42 months

Deliverable Number	D2.2
Deliverable Name	First integrated hardware/software system
WP Number	2
Lead Beneficiary	ETH
Dissemination Level	Public
Internal Reviewer	ABB
Due Date	31.12.2022
Date of Submission	
Version	1.0

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101017008

Revision History

Version	Date	Author(s)	Comments
1.0	09.12.2022	Paula Wulkop, Giulio Schiavi, Francesco Milano, Lionel Ott	First version
2.0	12.12.2022	Pietro Falco	Reviewed

Contents

Summary

The objective of this document is to describe how the software and hardware components from the different partners are integrated with each other. This document is a direct update of the deliverable D2.1, which included a description of all the individual software modules. In this document, we describe how these modules have been integrated and tested together for a demo during the integration week at ABB in October 2022. Furthermore, we describe general guidelines on communication interfaces and software setup such that it is easy to deploy new code on the existing devices.

Introduction

This document consists of three chapters. The first Chapter addresses general rules for the system architecture and software deployment. The second Chapter describes the software and hardware integration of multiple modules tested during the integration week. The last Chapter is an updated list of modules, based on Deliverable 2.1.

System Architecture and Software Deployment

The purpose of this Section is to describe how the proposed modules are integrated into one system, focusing on three separate aspects:

- Communication between the modules
- Coordination between the modules
- Deployment of software components to a shared compute platform

In doing so, it will discuss the difficulties encountered in the integration week in October 2022 and outline solutions for them.

Communication

Communication between the modules is handled via the Robot Operating System (ROS) framework, which is the standard choice for modular software development in research settings. At the time of writing, two major versions are available, ROS and ROS2. Official support for all ROS releases will be terminated as of 2024, while the most recent ROS2 release ("Humble") will be supported until 2027. Communication between ROS modules and ROS2 modules was found to be inefficient and unreliable. For these reasons, after October 2022, ROS2 "Humble" will be used exclusively. This will increase the lifespan of the software components and reduce the effort needed for integration. ROS2 "Humble" can be run natively on Ubuntu 22.04.

Coordination

A behaviour engine is used to coordinate the interaction between the modules required for a demo. In practice, this is implemented as an additional module, using the FlexBe software [8]. This provides a graphical user interface to design a state machine, which handles the transitions between the different modules. The state machine triggers the required node to start its process and waits for the node to return a boolean success flag. Based on this returned value, the state machine either triggers the next node or executes some fail-safe behaviour. More details are available in the dedicated GitHub repository, found at [10].

Software deployment

At the October 2022 integration week, deployment of all modules on the shared compute platform was found to lead to conflicts between different modules. For example, two modules required different versions of the OpenCV library to be installed in the same location. In order to avoid this issue, after October 2022 all the software modules (excluding ABB hardware-related modules and sensor drivers) will be deployed using OS-level virtualization. OS-level virtualization allows the existence of multiple isolated filesystems, called containers, on a shared host compute platform. Access to shared resources can be restricted, as programs running inside a container can only access the resources that have been assigned to it [14]. A container setup therefore reduces the risk of conflicts during

integration by isolating each module in a separate filesystem. Containers are also highly portable (*i.e.,* they can be run on an arbitrary host platform), bringing down significantly the effort needed for each integration. The portability of containers additionally allows external researchers to easily use HARMONY software, increasing the impact of the project.

Several solutions for software virtualization are currently available. For HARMONY, we chose to adopt Apptainer [7] due to its ease of use, compatibility with other implementations, and overall good support. For more information, readers are directed to the Apptainer documentation found online at [9]. Taking into account the requirements outlined in the Communication section, all the HARMONY containers will be based on an Ubuntu 22.04 filesystem running ROS2 Humble.

Integration week Demo

Software Modules

During the integration week at ABB in October 2022, a demo was set up to test the capabilities required for the USZ and KUH use case. Specifically, the goal of the demo was that the robot autonomously opens the grey box from the Karolinska hospital, whereas the box position is only roughly known a priori. This demo requires the robot to first navigate in a known environment towards a given goal. When the goal position is reached, the robot needs to locate the box, estimate its pose and perform the manipulation task, i.e. opening the box. Figure 1 shows an outline of the demo, as a flow diagram. The following modules are used in the demo:

- Localization_MCL: Localises the robot in the map
- Local_Motion_Planning: Plans the robot path to the goal position
- Perception_Panoptic_Segmentation: Performs segmentation of the box
- Perception_Object_Pose_Estimation: Estimates the 6D pose of the box
- Arm_Left/Right_Impedance_Controller: Controls the opening of the box, given its pose
- People_and_Moving_Object_Tracking: Tracks moving obstacles

The FlexBe State Machine is used to coordinate the modules (see System Architecture and Module Integration -> Coordination).

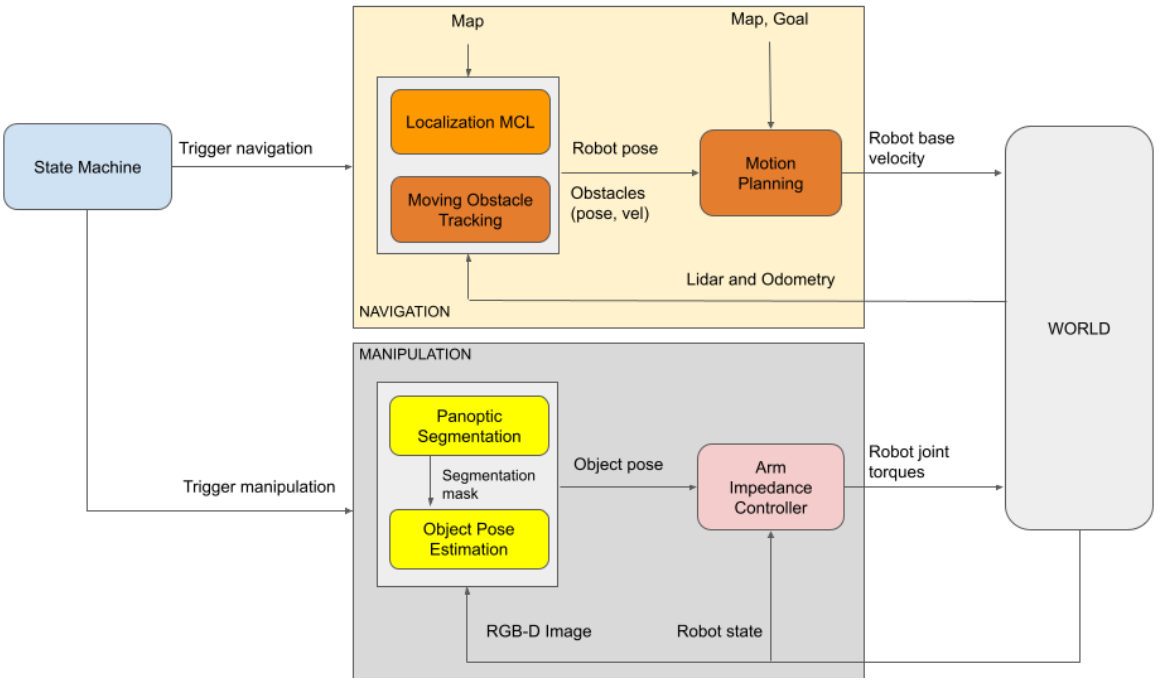

Figure 1: Flow diagram of the integration week demo. Each node in the Figure represents one module (e.g. State Machine)

Hardware Integration

The demo was tested on the ABB mobile manipulation platform. In order to provide the input to the perception software modules, several cameras were mounted on the robot platform using a 3D-printed structure (the specifications of the cameras are described in Deliverable 3.2). For each camera, the intrinsics are calibrated using the open-source ASL Kalibr toolbox, which can be obtained from [11]. Several procedures were tested for the

calibration of the camera extrinsics (position and orientation relative to a fixed point on the robot). The final procedure is described on the harmony github repository [12].

In order to provide the computational resources necessary to run all of the software modules, ABB purchased a Zotac compute platform (model MAGNUS EN173080C) and integrated it into the mobile manipulation platform. The full specification of the Zotac platform can be found at this link [13]. At the demo, we were able to show that the computer has sufficient compute power to run the modules described above (“Software Modules”). Further evaluation of the compute power will have to be performed once all the modules are tested together.

Updated list of modules from Deliverable 2.1

This Section is a direct update of the one from Deliverable 2.1 and contains the list of all the modules identified in this phase of the project. An overview and their interconnection are given in Figure 2. We direct readers to the online version of the system architecture overview at [15] where they can navigate through the system architecture details in an interactive fashion.

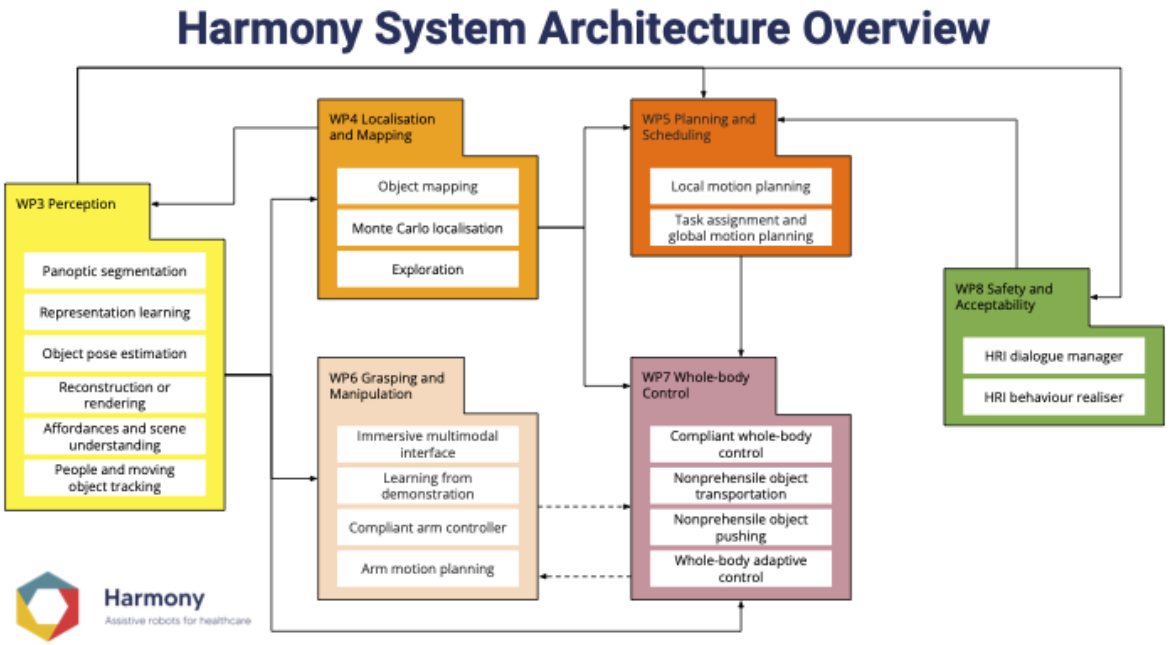

Figure 2: Overview of the modules. The reader is directed to this website [15] to access the interactive system architecture overview.

Perception

The following modules are necessary to implement the perception robot capabilities:

- Perception_Panoptic_Segmentation
- Perception_Representation_Learning
- Perception_Object_Pose_Estimation
- Perception_Reconstruction_or_Rendering
- Perception_Affordances_and_Scene_Understanding
- Perception_People_and_Moving_Object_Tracking

Perception_Panoptic_Segmentation

Partner: BONN
Interface:

- Input Data:
  - RGB-D Data (messages via RosNode at 15Hz)
- Output Data:
  - Pixel-wise semantic labels as an image where one channel corresponds to the class and the other to the instance id (end-to-end model prediction without postprocessing for instance segmentation: 10Hz).

Parameters:

- Number of network parameters (and in general network architecture)

Functionality Description:
Given input data, i.e., an RGB-D image, the semantic labels and instances are inferred using a CNN using a shared encoder and decoders that provide semantic and instance segmentations, similar to [1] building on recent advances in the backbone design (EfficientNetV2, etc.).

Data size and quantitative information:

- Size of the model for semantic segmentation: 300 MB.
- Size of the model for panoptic segmentation: 1 GB.
- Size of images: 20-300 KB.
- Actual image size used for training: RGB 640x480x3, depth 640x480.
- GPU memory requirements: about 5 GB for loading the panoptic segmentation model with weights, loading RGB and depth images and making inference. Smaller models (RGB-only, semantic segmentation-only) lead to smaller memory requirements.

Perception_Representation_Learning

Partner: ETHZ
Interface:

- Input Data (offline):
  - RGB-D data
  - Per-frame camera pose
  - Per-frame segmentation mask
- Output Data:

- 3-D representation of the object, mapping points in space to colour, surface density, and features that can be used for object pose estimation (cf. `Perception_Object_Pose_Estimation`). This representation is in the form of a multilayer perceptron network.

Parameters:

- Number of network parameters (and in general network architecture)
- Number of camera viewpoints

Functionality Description:
This module builds object representations to be included in the *object database* (cf. Task 3.3). These representations provide dense mapping from coordinates in an object frame to: colour, surface density, and features that can be used for object-pose estimation. In particular, given as input RGB-D images, object segmentation masks, and camera poses, a *neural field* (i.e., a multi-layer perceptron that takes coordinates as input) [4] is trained to fit the above-mentioned mappings in an offline step. The camera poses can optionally be refined using Structure-from-Motion. In this offline step for the construction of the object database, the object observations should be taken from several different viewpoints (e.g., up to 100) all around the object.

Perception_Object_Pose_Estimation

Partner: ETHZ
Interface:

- Input Data (single frame):
  - RGB-D Data
  - Camera pose in map frame
  - Per-frame segmentation mask and class label
  - Pre-trained representations for known objects whose pose should be estimated (cf. `Perception_Representation_Learning`)
- Output Data:
  - 6-D object pose in the camera frame

Parameters:

- None.

Functionality Description:
The module is used to return an estimate of the 6-D pose of a known object from the Harmony database (cf. `Perception_Representation_Learning`) with respect to the camera frame. The obtained pose complements the object representations formed as part of Task 3.3 and is necessary to perform manipulation tasks (cf. Task 7.3). A single RGB-D observation of the object of which the pose should be estimated is required, as well as the camera pose with respect to the map frame. Additionally, for each viewpoint the object mask, along with the class label, is assumed to be provided (cf. `Perception_Panoptic_Segmentation`).

Perception_Reconstruction_or_Rendering

Partner: ETHZ
Interface:

- Input Data:
  - Object database from `Perception_Representation_Learning`

    - Desired object class to render or reconstruct
    - If rendering, pose from which to render (up to 5 Hz)
- Output Data:
    - Rendering (e.g., colour or depth) of the object from the given viewpoint (5 Hz).
    - Alternatively, reconstruction (mesh or point cloud) of the given object. A point cloud can be obtained by querying the network (cf. `Perception_Representation_Learning`) at a set of points either uniformly distributed across the scene, or belonging to a specific set of interest, depending on the applications. A reconstruction algorithm like Marching Cubes can be applied to obtain a mesh from the point cloud.

Parameters:

- Rendering/reconstruction resolution.

Functionality Description:

Given the object representations provided by the module `Perception_Representation_Learning`, which encode the object properties, this module allows to extract these properties in a format that can be used for subsequent tasks such as manipulation and control. In particular, any of these representations can be returned:

- A point cloud containing a sample of surface points from an object of interest, where each point is associated with its properties (e.g., colour, or probability of occupancy);
- A mesh, fitted from the above point cloud, that reconstructs the object surface;
- A rendered view of the object, with each pixel associated with the properties (e.g., colour) of the corresponding point in the scene. This can be obtained via *neural rendering* [4].

Rendering and/or reconstruction might be required by tasks involving manipulation (cf. Task 7.3).

Perception_Affordances_and_Scene_Understanding

Partner: ETHZ

Interface:

- Input Data:
    - RGB-D Data (>=5 Hz)
    - Pose and map from `Localization_MCL` module and `Localization_Object_Mapping` (>=5 Hz)
    - Panoptic Segmentation from `Perception_Panoptic_Segmentation` module (>=5 Hz)
- Output Data:
    - Object-centric, scene-level representation containing information about attributes, dynamic state, affordances and relationships between objects. The data structure could be a scene graph where each detected object is represented through a node and has a list of attributes associated with it. Each relationship between the objects is represented by an edge in the graph. (5 Hz)
    - In particular, object dynamicity will be a key property captured by the scene graph and dynamic objects will also include velocity information. (5 Hz)

Parameters:

- Types of affordances to consider

Functionality Description:
The goal of this module is to enable a higher-level scene understanding by getting a notion of the objects in the scene, as required by Task 3.4. The robot will learn object properties and affordances, where affordances mean the possible interactions of the objects with the robot and with each other. A scene representation will be built up which includes attributes (eg. material, texture) and dynamic states of the objects (eg. open/closed) as well as relations between the objects (eg. standing on, inside of) and affordances. The affordances could be task-specific (eg. test tubes can be placed in the rack, which can be placed in a box, which can be placed on the cart). A possible structure to represent this information are 3D Scene Graphs, as used for example in [3], [6]. The information will be obtained from a combination of sources: Some properties will be derived directly from the geometry, some will be learned using an object database and scene graph datasets and affordances could be learned from human demonstrations. The scene graph will be built up online in an incremental fashion: As more areas of the scene are discovered by the robot and more objects appear in the segmentation map, additional nodes and relationships will be added. The existing ones can also be updated if the semantic labels change due to additional perspective views.

Perception_People_and_Moving_Object_Tracking

Partner: ETHZ
Interface:

- Input Data:
  - RGB images from the 3 FLIRs and the front-facing Kinect (15 Hz).
  - 2D LiDAR scan, front & back (40 Hz)
  - Robot pose (10Hz)
- Output Data:
  - Per-frame estimation of position and extent of humans and moving objects in the 360° scene (5 Hz).
  - Tracked movement of the humans and moving objects, including per-frame velocity estimates (5 Hz).

Functionality Description:
This module uses learning methods to detect humans and other moving objects, such as wheelchairs or carts, from RGB images and LiDAR scans. Additionally, it matches the detections to the past frames, such that it can consistently track the movement of the humans and moving objects, providing a velocity estimate. Some elements of the module are based on the perception stack of the Crowdbot EU-Project. The output of this module can be used for local motion planning (cf. Task 5.1).

Localization and Mapping

The following modules are necessary to implement the localization and mapping robot capabilities:

- Localization_Object_Mapping
- Localization_MCL
- Localization_ModelAcquisition

Localization_Object_Mapping

Partner: BONN

Interface:

- Input Data:
  - Panoptic segmentation from `Perception_Panoptic_Segmentation`
  - Pose information from the localization module
  - 2D grid map build from LiDAR using existing SLAM system (e.g., GMapping)
- Output Data:
  - Object instances added to the environment model
  - Map representation that can be used for localisation
  - Possibly masking/removing dynamic parts of the environments
  - Refined poses by exploiting the information above

Functionality Description:

Given the semantic scene interpretation of the `Perception_Panoptic_Segmentation` and the pose/2D grid map information, we want to build an enriched map representation that includes explicit object instances, such that these can be later used to improve localization. The module will provide a representation that can be used in the MCL module to perform global localization. The resulting localization map will be 2D or 2.5D to account for the indoor environment and the provided pose that is (x, y, theta) instead of a full 6D pose.

Localization_MCL

Partner: BONN

Interface:

- Input Data:
  - Semantic information from Perception_Panoptic_Segmentation
  - Maps from the module `Localization_Object_Mapping`
  - Odometry information from visual odometry & wheel encoders (50 HZ)
  - 2D LiDAR (40 Hz)
  - RGB images from the 3 FLIRs and the front-facing Kinect (15 Hz)
- Output Data:
  - Global pose in relation to the map (10 Hz)

Parameters:

- Number of particles

Functionality Description:

Given the maps provided by the module `Localization_Object_Mapping`, which encode the scene properties, this module provides a localization using a Monte Carlo Localization (MCL) using a particle filter to model the multi-modality of the localization problem, which will be

needed to handle uncertainty in a large-scale environment with many rooms, sharing a similar setup as commonly found in a hospital. A major point will be the integration of the object knowledge provided by the `Perception_Panoptic_Segmentation` in the observation model of the MCL approach that could help to distinguish different rooms.

Localization_Model_Acquisition

Partner: BONN

Interface:

- Input Data:
  - Maps from the module `Localization_Object_Mapping`.
  - Pose from `Localization_MCL`
- Output Data:
  - Pose of next viewpoint to capture

Parameters:

- Number of considered views for exploration

Functionality Description:

Given the currently acquired maps, we want to generate viewpoint poses that can be used to record a more complete map or more precise information about objects with only minimal human intervention. Here, we want to determine the next best views that improve the map information, but at the same time optimise the path planning such that nearby poses are first explored.

Planning and Scheduling

The following modules are necessary to implement the planning and scheduling robot capabilities:

- Local_Motion_Planning
- Task_Assignment_and_Global_Motion_Planning

Local_Motion_Planning

Partner: TUD
Interface:

- Input Data:
  - 2D grid map (3D for manipulation or if robot bigger than base) with segmented static and dynamic obstacles (dynamic obstacles at a distance between 0.1m and 10m from the border of the robot), possibly including affordances
  - Global pose in relation to map (min 10 Hz, preferred 20 Hz)
  - Position, velocity, orientation, footprint of dynamic obstacles from 'Perception_People_and_Moving_Object_tracking' (min 10 Hz, preferred 20 Hz), possibly including semantic information
  - Dynamics of robots
  - Goal locations from `Task_Assignment_and_Global_Motion_Planning` (whenever new goal location appears)
- Output Data:
  - Collision-free motion plans (20 Hz)

Parameters:

- Weights of the cost function
- Neural network layers and design
- Robot model and size parameters

Functionality Description:
Given information on the environment, this module generates collision-free trajectories (Task 5.1) for the robot in dynamic and obstacle-rich environments shared with humans. Moreover, it enables the agent to track the provided trajectories. The aim is to produce more intuitive motion plans (for the humans and other robots) that avoid congestion and that are safe (Task 5.2).

Task_Assignment_and_Global_Motion_Planning

Partner: TUD
Interface:

- Input Data:
  - Defined tasks (pickup and drop-off location, object size/weight, time constraints, possibly priority or urgency, cooperation-requirements)
  - Static map of corridors and rooms
  - Physical constraints of robots (e.g. battery, speed)
  - Possibly also history of tasks to predict high demand areas
- Output Data:

- Schedule of goal poses for each robot

Parameters:

- Number of robots
- Capacity of robots
- Available communication between robots (e.g. number of considered neighbours)

Functionality Description:

This module performs online (on demand) assignment of tasks to a group of robots (Task 5.3). For the assignment a distributed method is used in which robots will agree on tasks via communication with neighbours. Capacity constraints of robots, time constraints and priorities are considered. For multi-robot manipulation, e.g. multiple robots carrying large objects, additional constraints for joint manipulation/execution are considered. When human co-workers are in the loop, the team's behaviour can be adapted.

Grasping and Manipulation

The following modules are necessary to implement the grasping and manipulation robot capabilities:

- Immersive_Multimodal_Interface
- Grasp_Pose_Prediction
- Learning_from_Demonstration
- Compliant_Arm_Controller
    - Arm_Left/Right_Joint_Controller
    - Arm_Left/Right_Impedance_Controller
    - Hand_Left/Right_Controller
    - Hand_Left/Right_Current_Limit_Controller
- Arm_Motion_Planning

Immersive_Multimodal_Interface

Partner: UEDIN

Interface:

- Input Data:
    - Pose commands of human demonstrations
- Output Data:
    - Retargeted end-effectors' poses
    - Joint torque commands
    - Human performance metrics

Parameters:

- VR head-mounted display specs (FOV, pixel resolution, frame-rate e.g. 90Hz)
- Leap Motion Hand Controller specs (FOV, frame-rate e.g. 120Hz)
- Physics simulation frequency: 1000Hz
- Haptic device sampling rate: 1kHz-2kHz
- Robot control frequency: 1kHz

Functionality Description:

The `Immersive_Multimodal_Interface` module provides a natural and intuitive way for medical staff to deliver new manipulation capabilities at aided, semi-, and autonomous levels. The clinically proven haptic devices will be used to demonstrate a wide range of daily operations following specific safety measures, using multi-sensory gesture recognition, motion retargeting, and adaptation for dexterous and precise manipulation. The human performance will be quantified by the success rate of repeated manipulation and grasping tasks within the defined use case scenarios to get the qualified demonstration data used to control policies for grasping and manipulation.

Grasp_Pose_Prediction

Partner: UEDIN

Interface:

- Input Data:
    - RGB-D Data (depth image should be aligned with RGB image)

- tf data, from robot base link (“myumi_005_base_link”) to camera link (“kinect_subordinate_rgb_camera_link”)

- Output Data:
  - Detected object instances
    - for each detected object instance
      - class label
      - parameters of 2D bounding box
      - instance mask
      - grasp configurations in image plane (x, y, w, h, rotation)
      - grasp configurations in robot base link (position: x,y,z; rotation: w,x,y,z)

Parameters:

- Crop area (SSGROS.crop_area): Crop the image and only detect object in cropped area.

Functionality Description:
This module is used to synthesize instance-wise grasp configuration with RGB-D input and convert the grasp configurations from image plane to grasp pose (position, orientation) in robot base frame.

Learning_from_Demonstration

Partner: UEDIN
Interface:

- Input Data:
  - Human demonstrations of retrieving vials from the table and placing them on the trays
  - Data of the provided vials of different sizes and materials
  - Robot proprioceptive feedback (joint positions/velocities, end-effectors’ torques, etc)
- Output Data:
  - Learned control policies for grasping and manipulation

Parameters:

- Network hyperparameters
- Robot’s proprioceptive feedback sampling rate: 250Hz - 1kHz
- Human commands sampling rate: 1kHz

Functionality Description:
The `Learning_from_Demonstration` module enables the transfer of manipulation, grasping, object handling, and carrying skills from trained and verified demonstrations to different levels of autonomy for operating the wide range of instruments in use case 2. The module is developed within the tasks T6.1 and T6.3.

Compliant_Arm_Controller

Arm_Left/Right_Joint_Controller

Partner: CREATE

Interface:

- Input Data:
  - Joint name, positions, velocities, accelerations
  - Duration time from start
  - Type of trajectory (trapezoidal, cubic polynomial, quantic polynomial)
- Output Data:
  - Joint position, time, result (succeed or failed)

Parameters:

- Position representation (radian, degree)+
- Velocity representation (radian/second, degree/second)
- Via points

Functionality Description:

Moves the selected joints according to the sequence of positions that the left/right arm joints have to reach in given time intervals. Each trajectory point specifies [positions, Velocities, accelerations] for a trajectory to be executed. Additionally, the duration time is used to guarantee the execution time for the trajectory. The frequency of the module is 250Hz.

Arm_Left/Right_Joint_Controller_SafeCommand

Partner: CREATE

Interface:

- Input Data:
  - Joint name, positions, velocities, accelerations
  - force
  - Duration time from start
  - RGB-D data
  - LIDAR data
- Output Data:
  - Joint position, time, result (succeed or failed)

Parameters:

- Position representation (radian, degree)
- Velocity representation (radian/second, degree/second)

Functionality Description:

Moves the selected joints according to sequence of positions that the left/right arm joints have to reach in given time intervals if and only if it does not lead to a self-collision or object collision.

Arm_Left/Right_Impedance_Controller

Partner: CREATE

Interface:

- Input Data:

    - Desired cartesian positions, velocities, accelerations (250 Hz)
    - Desired inertia, damping, stiffness (constant at the moment, in theory up to 250 Hz)
- Output Data:
    - Position error, rate of position error (250Hz)

Parameters:

- Position and orientation representation (Euler angles, quaternion angles)

Functionality Description:
This node is used to tune the impedance parameters of the left/right arm according to the desired parameters.

Hand_Left/Right_Controller

Partner: CREATE
Interface:

- Input Data:
    - Joint name, positions, velocities, accelerations,
    - Effort
    - Duration time from start
- Output Data:
    - Position error, rate of position error, result (succeed or failed)

Parameters:

- Position and orientation representation (Euler angles, quaternion angles)

Functionality Description:
Moves the selected joints according to the sequence of positions that the left/right hand joints have to reach in given time intervals. Each trajectory point specifies [positions, velocities, accelerations] or [positions, efforts] for trajectory to be executed. Additionally, the duration time is used to guarantee the execution time for the trajectory. The frequency of this module is considered 250Hz. For sake of exploration, the use of a robotic hand will be investigated as a possible alternative to the gripper.

Hand_Left/Right_Current_Limit_Controller

Partner: CREATE
Interface:

- Input Data:
    - Actuator name
    - Current limit
- Output Data:
    - Joint position

Functionality Description:
Set maximum allowed current for each actuator of the left hand specified as a factor in [0 1] of the actuator’s maximum current. For example, the number 0.5 would set an actuator’s current limit to half its maximum value.

Arm_Motion_Planning

Partner: CREATE

Interface:

- Input Data:
  - ARM (Arm_Left, Arm_Right, Arm_Left_Torso, Arm_Right_Torso, Dual_Arm_Torso)
  - Desired position (with respect to base frame)
  - Type of planners (geometric planners, control-based planners)
  - Duration time from start
  - Maximum time decision
- Output Data:
  - E-E current position, time, result (succeed or failed)

Parameters:

- Position representation (metre)
- Velocity representation (radian/second, degree/second)

Functionality Description:

This module moves the selected arms (Single arm or dual-arms, with or without torso) according to input data. The types of planners are selected by the users. This module covers the requirements in Tasks 7.2, 7.3. The Frequency of this module is considered 250Hz.

Whole-body Control

The following modules are necessary to implement the whole-body control robot capabilities:

- Compliant_Whole-body_Control
- Nonprehensile_Object_Transportation
- Nonprehensile_Object_Pushing
- Whole-body_Adaptive_Control

Compliant_Whole-body_control

Partner: CREATE
Interface:

- Input Data:
  - Robot state (torque/force sensor at the wheels, joint positions, joint velocities) (250Hz)
- Output Data:
  - Joint torques (250Hz)
  - Joint velocities (250 Hz)
  - Joint positions (250 Hz)
  - Wheel velocities (250 Hz)

Parameters:

- Robot dynamic parameters
- Compliance parameters

Functionality Description:
Provide a compliant behaviour of the robot to the external environment (interactions, collisions, obstacles). The frequency of all messages is 250Hz.

Nonprehensile_Object_Transportation

Partner: CREATE
Interface:

- Input Data:
  - Robot state (end-effector pose) (250Hz)
  - Transported object state (position, orientation, linear and angular velocities) (30Hz)
  - End-effector force/torque measurements or estimates (250Hz)
- Output Data:
  - Desired task trajectories (250Hz)

Parameters:

- Controller gains
- Object dynamic parameters

Functionality Description:
Transport an object in a nonprehensile way (i.e., on a tray) with the mobile robot. A controller that outputs the next target end-effector state (250 Hz) will be developed. The controller will need the robot state, in terms of end-effector pose (250 Hz), and the

transported object state, in terms of object pose and velocity (30 Hz). End-effector force/torque measurements or estimates (250 Hz) will be needed as well. The controller outputs the desired task trajectory (250 Hz), in terms of next end-effector pose, velocity, acceleration. This will be used by the whole body controller to calculate the motor commands.

This module is developed as part of Task 7.3.

Nonprehensile_Object_Pushing

Partner: CREATE, UEDIN
Interface:

- Input Data:
  - Robot state (end-effector pose) (250Hz)
  - Environmental reconstruction of the objects on the desk/shelf (position, orientation, linear and angular velocities for each of the objects in the scene) (30Hz)
  - End-effector force/torque sensor measurements or estimates (250Hz)
- Output Data:
  - Desired task trajectories (250Hz)

Parameters:

- Controller gains
- Objects dynamic parameters

Functionality Description:
Push objects away on a cluttered desk/shelf to grasp the desired one, using a Model Predictive Controller (MPC) to track the desired object position and orientation (25Hz) with command of the computed interaction forces through admittance control.

This module is developed as part of Task 7.3.

Whole-body_Adaptive_Control

Partner: ABB
Interface:

- Input Data:
  - Robot state (configuration and velocities) (250Hz)
  - Desired task trajectories (position, velocity and acceleration) and priority levels (250Hz)
- Output Data:
  - Motor torque commands (250Hz)

Parameters:

- Controller gains
- Kinematic parameters
- Estimated dynamic parameters

Functionality Description:
Whole-body control methods use the model of the robot to realise a compliant behaviour and safely interact with humans and the environment. In the presence of variable loads on

the base and tools at the end-effectors, the dynamic parameters of the model will change, and their online adaptation has the benefit of enhancing tracking performance. Given the current state of the robot, the measured force/torques and the desired trajectory, the controller will update the internal estimate of the dynamic parameters and compute based on that the required motor torques to perform the task. The module will include also an impedance self-tuning feature to set automatically the impedance control parameters.

This module will also handle the case of pure navigation commands, i.e., when the arms of the robot are not being involved in a given manipulation task and therefore are not being used.

This module is developed as part of Task 7.5.

Safety and Acceptability

The following modules are necessary for the whole-body control skills:

- HRI_Behaviour_Generator (Formerly “HRI_Dialogue_Manager”)
- HRI_Behaviour_Realiser

HRI_Behaviour_Generator

Partner: UT
Interface:

- Input Data:
  - Functional Behaviour requests
- Output Data
  - XML template that describes a certain robot action. These actions will need to be translated into platform-specific motion primitives and speech (10-20 Hz)

Parameters:

- Parameters on beliefs about the world (e.g. number of interlocutors, task progression, current task).
- Refresh rate of template checking (up to ~100 Hz)

Functionality Description:
(Formerly “HRI_Dialogue_Manager”). The behaviour generator builds a plan for the multimodal social behaviours to be executed by the *HRI_Behavior_Realizer*. It also keeps track of ongoing behaviour execution of the Realizer, and thus knows to predict completion times, knows how to gracefully stop ongoing execution (if plan changes).
Multimodal behaviours are described in templates using the Behavior Markup Language (BML) and represent various functional behaviours. There would be templates for “Greet Passerby”, or “Indicate Desired Driving Direction”. The HRI_Behavior_Generator should offer a clear interface with the FlexBE Behavior Engine for integration in the overall robot behaviour. We are evaluating if composing of HRI behaviours can be done completely in FlexBE, by providing more powerful, parameterized FlexBE states for different behaviours, including more complex execution feedback, possibly through the ROS2 action service architecture. This module will be developed as part of Tasks **8.2-8.5.**

HRI_Behaviour_Realiser

Partner: UT
Interface:

- Input Data
  - Behaviour Plans for robot behaviour specified in BML.
- Output Data
  - Control Primitives for other robot modules (LEDs, Screen, Navigation/Movement) per modality. (latency time ~2 to 10 ms, may depend on middleware)

Parameters:

- Current robot platform
- BML-specific parameters
- Schedule for action execution

Functionality Description:
Once a behaviour plan is generated by the *HRI_Behavior_Generator*,, the robot behaviour specification is sent to the behaviour realiser AsapRealizer [2], [5]. The AsapRealizer is responsible for synchronising, queuing, and monitoring the execution of the robot behaviour. It typically has a latency of about 2-10 ms. This is dependent on the complexity of the planned robot actions, and will require testing once we have this system integrated. A behaviour realiser engine takes the behaviours specifications sent by Flipper as input and can translate these to control primitives for a specific robot platform. For example, the behaviour plan may specify that the robot should smile for two seconds, which is then converted by Asap into platform-specific control primitives that correspond to a smile for the current robot platform. AsapRealizer offers various means of interfacing other components through a flexible middleware implementation. This module will be developed as part of Task **8.2-8.5.**

Conclusions

A formal specification of all the modules within the system architecture designed for HARMONY, along with their interfaces, has been described. The modules have been organised according to the functionality that they implement and, for each of them, interface, parameters and functionality description have been provided.

During the integration week in October 2022, the interaction between some core modules from different partners was already successfully shown. In future integration weeks, the remaining modules will be tested on the robot and the software architecture for the final demo will be implemented.

References

[1] Milioto, A., Behley, J., McCool, C., and Stachniss, C. (2020). "LiDAR Panoptic Segmentation for Autonomous Driving". In IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).

[2] Reidsma, D., and van Welbergen, H. (2013). “AsapRealizer in practice – A modular and extensible architecture for a BML Realizer”. Entertainment Computing, 4(3), 157–169.

[3] Rosinol, A., Gupta, A., Abate, M., Shi, J., and Carlone, L. (2020). "3D Dynamic Scene Graphs: Actionable Spatial Perception with Places, Objects, and Humans". In Robotics: Science and Systems (RSS).

[4] Tewari, A., Fried, O., Thies, J., Sitzmann, V., Lombardi, S., Sunkavalli, K., Martin-Brualla, R., Simon, T., Saragih, J., Nießner, M., Pandey, R., Fanello, S., Wetzstein, G., Zhu, J.-Y., Theobalt, C., Agrawala, M., Shechtman, E., Goldman, D. B, and Zollhöfer, M. (2020). "State of the Art on Neural Rendering". In EUROGRAPHICS, 39(2).

[5] van Welbergen, H., Reidsma, D., Ruttkay, Z. M., and Zwiers, J. (2009). Elckerlyc. Journal on Multimodal User Interfaces, 3(4), 271-284.

[6] Wu, S.-C., Wald, J., Tateno, K., Navab, N., and Tombari, F. (2021). "SceneGraphFusion: Incremental 3D Scene Graph Prediction from RGB-D Sequences". In IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

[7] Apptainer. Online: https://apptainer.org/.

[8] FlexBe. Online: http://flexbe.github.io.

[9] Apptainer User manual. Online: https://apptainer.org/docs/user/main/.

[10] Harmony behaviour engine. Online: https://github.com/harmony-eu/harmony_behavior_engine.

[11] Kalibr. Online: https://github.com/ethz-asl/kalibr.

[12] Harmony sensor calibration. Online: https://github.com/harmony-eu/harmony_sensor_calibration.

[13] Zotac specifications. Online: https://www.zotac.com/us/product/mini_pcs/magnus-en173080c-barebone#spec.

[14] OS-level virtualization. Online: https://en.wikipedia.org/wiki/OS-level_virtualization.

[15] Interactive Harmony System Architecture. Online: https://docs.google.com/presentation/d/14uL7ydY5oRlRcbHjJUULF0PuRqaFBYR_r95-eHxev78/present#slide=id.g1301d03ed53_0_147.