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	D6
Deliverable Name	Formal Description of System Architecture
WP Number	2
Lead Beneficiary	ABB
Dissemination Level	Public
Internal Reviewer	ETH
Due Date	30 June
Date of Submission	30.06.2021
Version	1

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
0.1	16/06/2021	Gianluca Garofalo Pietro Falco	
1	30/06/2021	Gianluca Garofalo Pietro Falco	

Contents

Summary

During the System requirements and evaluation phase, several features have been identified for the correct execution of the tasks that the future generation of robots developed within the HARMONY projects will need to execute. The different robot capabilities that will be implemented on the robots, will need to rely on a modular architecture with appropriate interfaces and information exchanges. The objective of this document is to describe the modular system architecture, with a formal specification of modules and interfaces.

The modules are organized according to the corresponding robot capability that they are involved to implement. Each module is identified by its name and for each one, the corresponding interface and parameters are provided. A Functionality Description is also included, which provides additional information on the goals and purpose of the module.

List of modules

This section contains the list of all the modules identified in this phase of the project. An overview and their interconnection are given in Figure 1.

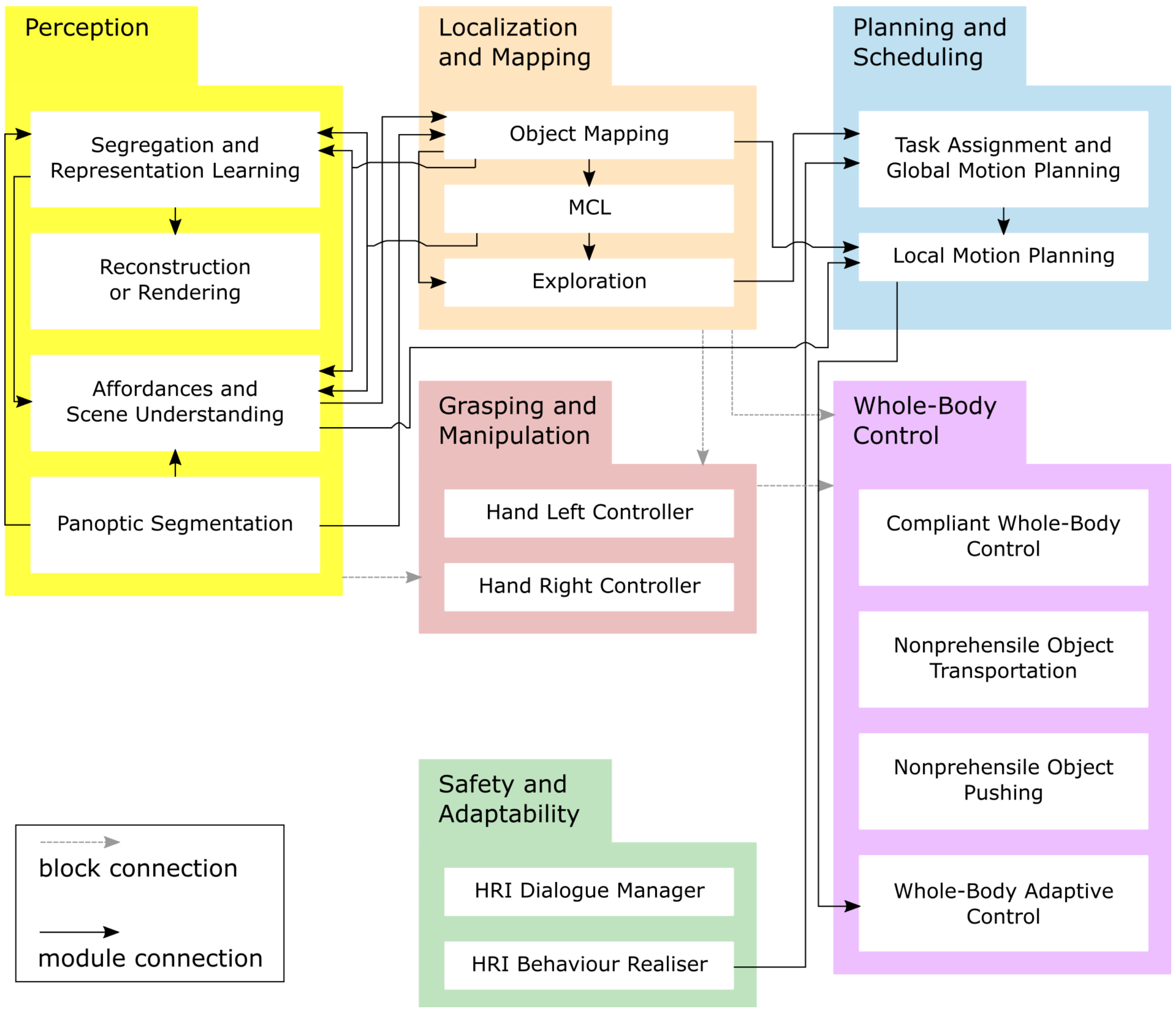
Perception
Segregation and Representation Learning
Reconstruction or Rendering
Affordances and Scene Understanding
Panoptic Segmentation
Localization and Mapping
Object Mapping
MCL
Exploration
Planning and Scheduling
Task Assignment and Global Motion Planning
Local Motion Planning
Grasping and Manipulation
Hand Left Controller
Hand Right Controller
Whole-Body Control
Compliant Whole-Body Control
Nonprehensile Object Transportation
Nonprehensile Object Pushing
Whole-Body Adaptive Control
Safety and Adaptability
HRI Dialogue Manager
HRI Behaviour Realiser
block connection
module connection

Figure 1: Overview of the modules

Perception

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

- Perception_Segregation_and_Representation_Learning
- Perception_Reconstruction_or_Rendering
- Perception_Affordances_and_Scene_Understanding
- Perception_Panoptic_Segmentation
- Perception_Object_Pose_Estimation_Point_Cloud

Perception_Segregation_and_Representation_Learning

Partner: ETHZ

Interface:

- Input Data
    - RGB-D Data
    - Pose & Map from `Localization_MCL` module and `Localization_Object_Mapping`
    - [To be evaluated: Semantic instance segmentation]
- Output Data
    - Set of submaps at different layers of abstraction, each represented by a neural network that encodes properties of points in the scene (color, occupancy). These properties are therefore fully encoded in the neural-network weights; the corresponding value for a point in the scene can be retrieved by querying the network at that point.

Parameters:

- Number of layers of abstraction
- Number of models/slots per layer (cf. Functionality Description)
- Number of network parameters (and in general network architecture)

Functionality Description: Given input data extracted from a scene, builds submaps where entities are segregated at different levels of hierarchy and representations for them are learned. Here we use *segregation* and *representation learning* in the sense from [Greff et al., 2020, Fig. 2]. The idea is that the module receives posed RGB-D images as inputs, and possibly associated 2D semantic labeling, and learns to fit a map of the scene at different levels of abstraction. The scene maps are based on *neural implicit representations* (e.g., [Sitzmann et al., 2019]), i.e., neural networks which learn to fit (mathematical) functions from 3D points in space to: color information or occupancy. We define here two forms of *segregation*:

- Horizontal segregation: At each level of abstraction, the map can be decomposed into different *slots*, where each slot contains elements that are related to one another, either semantically, functionally, spatially, or a combination of these. This segregation should possibly be learned beforehand in an self-supervised/not-fully-supervised way, likely requiring extensive training on large datasets.
- Vertical segregation: At each level of abstraction, the slots can each be considered as single submaps, and can themselves be further decomposed into smaller units.
  In the highest level of abstraction, the maps could for instance represent the whole scene; in a subsequent hierarchical layer, semantically-related parts of the scene could be assigned to a

same slot. Going further down, slots could be decomposed into semantic classes and instances.

While the segregation is carried out based on modules learned beforehand (cf. “Horizontal segregation” above), the map fitting is meant to be performed in an online fashion, by integrating the input data received from the sensors, possibly in real time (or close to it).

The actual labelling of objects or learning of relationships between entities in submaps could be implemented in a separate perception module, potentially based on the module `Perception_Affordances_and_Scene_Understanding`.

·In the future, learning of “generic” features, tailored to a downstream task, can be evaluated.

Perception_Reconstruction_or_Rendering

Partner: ETHZ

Interface:

- Input Data
  - Maps from the module `Perception_Segregation_and_Representation_Learning`.
- Output Data
  - Reconstruction (mesh or point cloud) of a given submap of the scene. A point cloud can be obtained by querying the map network (cf. `Perception_Segregation_and_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.
  - A possibility is also to obtain images containing the color/occupancy properties of the scene when observed from a specific viewpoint; this is done by querying the network at multiple specific points in space through *neural rendering* (e.g., [Tewari et al., 2020]).

Parameters:

- Submap to reconstruct, or
- Pose from which to render the scene properties.

Functionality Description: Given the maps provided by the module `Perception_Segregation_and_Representation_Learning`, which encode the scene 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 the points within a submap of interest, where each point is associated with its properties (e.g., color, or probability of occupancy);
- A mesh, fitted from the above point cloud, that reconstructs the parts of the scene contained within a submap;
- A rendered view of the scene, or of a single submap, with each pixel associated with the properties (e.g., color) of the corresponding point in the scene.

One relevant use case would be for instance reconstructing specific parts of the workspace according to a user input or to a known task.

Perception_Affordances_and_Scene_Understanding

Partner: ETHZ

Interface:

- Input Data
    - RGB-D Data
    - Pose & Map from Localization_MCL module and Localization_Object_Mapping
    - Panoptic Segmentation from Perception_Panoptic_Segmentation module
- Output Data
    - Scene Graph with information about the objects in the scene: attributes, dynamic state, affordances and relationships between objects. The data structure will be a 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.
    - In particular, object dynamicity will be a key property captured by the scene graph and dynamic objects will also include velocity information.

Parameters:

- Current Task of the robot
- Dataset of objects needed for the use case

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. 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 [Wu et al., 2021], [Rosinol et al., 2020]. 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_Panoptic_Segmentation

Partner: BONN

Interface:

- Input Data
    - RGB-D Data
- Output Data
    - Pixel-wise semantic labels as an image where one channel corresponds to the class and the other to the instance id
    - Pixel-wise instance segmentation of unknown objects

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 [Milioto et al., 2020] building on recent advances in the backbone design

(EfficientNetV2, etc.). In the long term, we want to extend the approach such that unseen instances of unknown object classes can be identified by using depth information to segment objects.

Perception_Object_Pose_Estimation_Point_Cloud

Partner: UEDIN

Interface:

- Input Data
    - RGB-D Data
- Output Data
    - Pose of the object, including position and rotation
    - The noise in standard deviation (SD) of the actual and predicted pose (in mm/cm/m and degrees for position and orientation respectively)
    - the colour of the object for later colour segmentation (Point clouds surrounding the object sufficient for pose estimation as described above but also, RGB data is needed for semantics)
    - Bounding box of the object, including the vertices positions e.g. [Tremblay et al., 2018]

Parameters:

- Labelled type of object in the current task (With sub-primitive: from simple (E.g. sphere, cube) to complex (Eg. a toy, bottle etc)
- Camera specifications (FOV, pixel resolution and noises, frame-rate e.g. 25Hz or 30Hz)
- Number of network parameters (and in general network architecture)

Functionality Description: Given the input data extracted from a scene, calculate:

- The predicted pose of the object (position and orientation) [high importance]
- Generate point clouds for grasping [average importance]
- High-confidence grasping areas.

Ideally, we would like to grasp different types of objects via immersive multimodal for imitation learning. This could go beyond primitive shapes (e.g. a cube/sphere or a cylinder) and extend towards grasping bottles, toys or other complex shaped rigid objects. While we can imitate the behaviour, we would ideally like to have point clouds of different objects (from a database) to determine the optimal grasping.
As far as pose extraction goes, this is more important, as the Sim2Real gap would be closed as much as possible by imitating the real noise (in SD) of the camera/approach in our simulation environment in Unity3D. Consequently, having the SD, calculated via two subsets of the “predicted” and “actual” position (Vector) and orientation (Quaternion/Euler) of the object would be very helpful. The noise of the depth image would also be helpful.
The variety of objects can be discretional, ideally primitive shapes should be included, with the

addition of more complex shaped ones. Similar to the DexNet 2.0 paper test objects [Mahler et al., 2017].

Localization and Mapping

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

- Localization_Object_Mapping
- Localization_MCL
- Localization_Exploration

Localization_Object_Mapping

Partner: BONN

Interface:

- Input Data
  - Panoptic Segmentation from `Perception_Panoptic_Segmentation`
  - Pose information from the localization module
- Output Data
  - Reconstruction with object instances (known and unknown)
  - Map representation that can be used for localization
  - Possibly dynamic parts of the environments (poses, class) or changed parts
    - Refined pose that accounts for changes in the environment(?)

Parameters:

- Resolution of reconstruction

Functionality Description: Given the semantic scene interpretation of the `Perception_Panoptic_segmentation` providing pixel-wise semantics and the pose from the localization module, we want to build a map representation that includes explicit object instances, such that these can be later used to update movable objects in the map. The module will provide a representation that can be used in the MCL module to perform global localization and update the pose in respect to the map. The localization map will be either 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. To account for changes, the map representation can also include information about changes that happened by revisiting already mapped areas by identifying objects that got relocated.

Localization_MCL

Partner: BONN

Interface:

- Input Data
  - Maps from the module `Localization_Object_Mapping`.

  - Odometry information from Visual Odometry & wheel encoders
  - Refined pose information from the mapping module
  - 2D LiDAR & RGB-D data
- Output Data
  - Global pose in relation to the map

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_Exploration

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 poses that can be used to record a more complete map with only minimal human intervention. Here, we want to determine the next best views that improve the map information, but at the same time optimize 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 (dyn. obs. within distance of 0.1m from border of robot and 10m), possibly including affordances
  - Position, velocity, orientation, footprint of dynamic obstacles from Perception_Affordances_and_Scene_Understanding
  - Dynamics of robots
  - Goal locations from Task_Assignment_and_Global_Motion_Planning
- Output data:
  - Collision-free motion plans

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 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_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 neighbors)

Functionality Description: This module performs online (on demand) assignment of tasks to a group of robots. 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

Arm_Left_Joint_Controller

Partner: CREATE

Interface:

- Input data:
    - Joint name, Positions, Velocities, Accelerations,
    - Force,
    - 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 arm joints have to reach in given time intervals. Each trajectory point specifies [positions, Velocities, accelerations] or [positions, force] for a trajectory to be executed. Additionally, the duration time is used to guarantee the execution time for the trajectory.

Arm_Left_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 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_Impedance_Controller

Partner: CREATE

Interface:

- Input data:
  - Desired cartesian positions, Velocities, Accelerations,
  - Desired Inertia, Damping, Stiffness
  - Desired Force, Measured Force
- Output Data
  - Position error, Rate of position error, Result (Succeed or Failed)

Parameters:

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

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

Arm_Right_Joint_Controller

Interface:

- Input data:
  - Joint name, Positions, Velocities, Accelerations,
  - Force,
  - 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 right arm joints have to reach in given time intervals. Each trajectory point specifies [positions, Velocities, accelerations] or [positions, force] for a trajectory to be executed. Additionally, the duration time is used to guarantee the execution time for the trajectory.

Arm_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 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_Right_Impedance_Controller

Partner: CREATE

Interface:

- Input data:
    - Desired cartesian positions, Velocities, Accelerations,
    - Desired Inertia, Damping, Stiffness
    - Desired Force, Measured Force
- Output Data
    - Position error, Rate of position error, Result (Succeed or Failed)

Parameters:

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

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

Hand_Left_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 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.

Hand_Left_Current_Limit_Controller

Interface:

- Input data:
  - Actuator name, Current limit
- Output Data
  - Joint Position

Parameters:

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.

Hand_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 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.

Hand_Right_Current_Limit_Controller

Partner: CREATE

Interface:

- Input data:
  - Actuator name, Current limit
- Output Data
  - Joint Position

Parameters:

Functionality Description: Set maximum allowed current for each actuator of the right 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.

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)
- Output Data:
    - Joint torques
    - Joint velocities
    - Joint positions,
    - Wheel velocities

Parameters:

- Robot dynamic parameters
- Compliance parameters

Functionality Description: Provide a compliant behavior of the robot to the external environment (interactions, collisions, obstacles).

Nonprehensile_Object_Transportation

Partner: CREATE

Interface:

- Input Data:
  - Robot state (torque/force sensor at the wheels, joint positions, joint velocities)
  - Transported object state (position, orientation, linear and angular velocities)
- Output Data:
  - Joint torques
  - Joint velocities
  - Joint positions
  - Wheel velocities

Parameters:

- Robot dynamic parameters
- Object dynamic parameters

Functionality Description: Transport an object in a nonprehensile way (i.e., on a tray) with the mobile robot.

Nonprehensile_Object_Pushing

Partner: CREATE

Interface:

- Input Data:
  - Robot state (torque/force sensor at the wheels, joint positions, joint velocities)
  - Environmental reconstruction of the objects on the desk/shelf (position, orientation, linear and angular velocities for each of the object in the scene)
- Output Data:
  - Joint torques
  - Joint velocities
  - Joint positions

Parameters:

- Robot dynamic parameters
- Objects dynamic parameters

Functionality Description: Push objects away on a cluttered desk/shelf to grasp the desired one.

Whole-body_Adaptive_Control

Partner: ABB

Interface:

- Input Data
  - Robot state (configuration and velocities).
  - Force/torque sensors measurements.
  - Desired task trajectories (position, velocity and acceleration) and priority levels.
- Output Data
  - Motor torques commands.

Parameters:

- Controller gains
- Kinematic parameters
- Estimated dynamic parameters

Functionality Description: Whole-Body control methods use the model of the robot to realize 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.

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.

Safety and Acceptability

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

- HRI_Dialogue_Manager
- (Possibly) HRI_Behaviour_Realiser

HRI_Dialogue_Manager

Partner: UT

Interface:

- Input Data:

    - *Processed* perception data (e.g. location of task-relevant materials in the scene, location of the interlocutor, the location of the interlocutor's face, or any communication from an interlocutor).
- Output Data
    - XML template that describes a certain robot action. These actions will need to be translated into platform-specific motion primitives and speech.

Parameters:

- Parameters on beliefs about the world (e.g. location of current interlocutor, number of interlocutors, task progression, current task).

Functionality Description: The dialogue manager responds to user events and generates appropriate behaviors. For this we use the dialogue manager Flipper 2.0 [van Waterschoot et al., 2018], which is an information state system. Flipper’s information state is a representation of Flipper’s beliefs about the world and is used to trigger certain behaviours. These behaviours are specified in XML templates, written in BML (Behavioural Markup Language). Each template specifies a certain robot behaviour specification, the preconditions that must be met in order for the template to be executed, and the effect of the execution on the information state.

(Possibly) HRI_Behaviour_Realiser

Partner: UT

Interface:

- Input Data
    - XML file on a certain robot behaviour specified in BML.
- Output Data
    - Well timed API call to execute a certain robot behaviour (e.g. look at interlocutor), possibly including a (JSON) file with platform-specific motion primitives.

Parameters:

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

Functionality Description: Once a template is executed in Flipper (HRI_Dialogue_Manager), the robot behaviour specification is sent to the behaviour realiser AsapRealizer [van Welbergen et al., 2009; Reidsma & van Welbergen, 2013]. The AsapRealizer is responsible for synchronising, queuing, and monitoring the execution of the robot behaviour. Furthermore, this is a BML behaviour realiser engine that takes the behaviours specifications sent by Flipper as input and can translate these to control primitives for a specific robot platform. For example, the BML sent by Flipper 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. This requires platform specific control primitives for each behaviour can also be contained in the AsapRealizer. Alternatively, AsapRealizer can simply send an API call to a different module that has the control primitives of the robot, and which then executes the robot behaviour.

Conclusions

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

A proper design of the system architecture is key to the efficient implementation of the different robot capabilities and therefore to the advanced capabilities required to fulfil the different tasks that robots have to perform. Building on the functionalities provided by each module, information can be exchanged and more complex skills can be implemented.

References

Greff, K., van Steenkiste, S., and Schmidhuber, J. (2020). “On the Binding Problem in Artificial Neural Networks”. CoRR abs/2012.05208.

Mahler, J., Liang, J., Niyaz, S., Laskey, L., Doan, R., Liu, X., Ojea, J, and Goldberg, K. (2017). "Dex-Net 2.0: Deep Learning to Plan Robust Grasps with Synthetic Point Clouds and Analytic Grasp Metrics". In Robotics: Science and Systems (RSS).

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).

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.

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).

Sitzmann, V., Martel, J. N. P., Bergman, A. W., Lindell, D. B., and Wetzstein, G. (2019). "Implicit Neural Representations with Periodic Activation Functions". In Conference on Neural Information Processing Systems (NeurIPS).

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).

Tremblay, J., To, T., Sundaralingam, B., Xiang, Y., Fox, D., and Birchfield, S. (2018). "Deep Object Pose Estimation for Semantic Robotic Grasping of Household Objects". https://arxiv.org/pdf/1809.10790.pdf

van Waterschoot, J., Bruijnes, M., Flokstra, J., Reidsma, D., Davison, D., Theune, M., and Heylen, D. (2018). “Flipper 2.0: A Pragmatic Dialogue Engine for Embodied Conversational Agents”. In Proceedings of the 18th International Conference on Intelligent Virtual Agents (IVA '18). Association for Computing Machinery, New York, NY, USA, 43–50. DOI:https://doi.org/10.1145/3267851.3267882

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

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).