PointIt: A ROS Toolkit for Interacting with Co-located Robots using Pointing Gestures

Gabriele Abbate, Alessandro Giusti, Antonio Paolillo, Boris Gromov,
Luca Gambardella, Andrea Emilio Rizzoli, Jérôme Guzzi
Dalle Molle Institute for Artificial Intelligence, *USI-SUPSI*, Lugano, Switzerland
gabriele.abbate@idsia.ch

***Abstract*—We introduce PointIt, a toolkit for the Robot Operating System (ROS2) to build human-robot interfaces based on pointing gestures sensed by a wrist-worn Inertial Measurement Unit, such as a smartwatch. We release the software as open-source with MIT license; docker images and exhaustive instructions simplify its usage in simulated and real-world deployments.**

***Index Terms*—pointing gesture, human robot interaction, ambient intelligence, ROS**

I. Introduction

Pointing gestures (i.e., using an extended arm to indicate something in the environment) are widely used by humans when interacting with other humans sharing the same space; they represent an intuitive and efficient way to communicate an approximate nearby position ("wait *there*, please") or to select an entity that is visible to both parties ("pick *this* package and bring it to *that* person") [1]. In most cases, except when extreme accuracy is required, alternative ways to express the same concepts are much less efficient in terms of communication speed and cognitive load, for both parties.

Pointing gestures are therefore an attractive communication mechanism for humans interacting with robots located in the same environment. For example, they can be used to intuitively select [2] and control [3], [4] mobile robots, e.g. to indicate objects on which the robot should act [5] or positions that it should reach; similar applications are also possible in domotic and ambient intelligence scenarios, e.g. indicating a light or an appliance to turn it on, or a landmark to query information about it.

Gestures can be sensed by sensors placed on the robot [6], in the environment [7], or worn by the user [8]–[10]. In this work, we focus on the third case and assume that the operator wears a wristband or smartwatch, equipped with an Inertial Measurement Unit (IMU) whose readings are streamed in real-time to the system. This sensing setup does not enforce any requirement on the robot's sensing suite and processing power, and makes the system inherently robust to visibility, lighting, or occlusion issues.

In this context, our **main contribution** is PointIt, a novel open-source, modular, ROS2-compliant software toolkit that can be used to implement Pointing-based User Interfaces (PUIs). It is composed of two main parts.

This work is supported by the European Commission through the Horizon 2020 project 1-SWARM, grant ID 871743.

The first part comprises task-agnostic code with three main functionalities.

(i) Using IMU readings to reconstruct the 3D *pointing ray* along which lies the object or location being pointed at. The pointing ray is expressed in a human-centered frame of reference.
(ii) Applying a relative localization [11] algorithm that, given a set of pointing rays and the corresponding known locations in a fixed world frame, returns an estimate of the relative pose of the operator in the environment. This functionality is only needed when the operator's location is not otherwise known.
(iii) Intersecting pointing rays (expressed in the world frame) with a map of the environment to compute the pointed location or object.

The toolkit makes it easy to implement systems such as the one proposed by Gromov et al. [12], [13], in which a user indicates a point on the ground while a small quadrotor continuously maneuvers in order to hover 20 cm above the pointed location (Figure 1 left); by relying on real-time feedback provided by the robot's own position, the user can achieve high control accuracy, even when the reconstructed pointing rays are affected by significant systematic errors.

The second part, which relies on the functionality implemented in the first, includes code tailored to applications in which: pointing gestures are used to select discrete objects placed at known locations in space; fixed LED strips are used to provide real-time feedback to the user during the process (e.g. by displaying a cursor updated in real-time, indicating the position currently being pointed at). One such application consists in selecting packages on a moving conveyor belt by pointing at them (Figure 1 right).

In Section II we describe the foundations of the pointing-based user interface. Then, in Section III we describe the ROS2-compliant implementation of the toolkit, and in Section IV we discuss how the toolkit is used to build user-interfaces in concrete scenarios. Finally, in Section V we describe the actual software release with installation and usage instructions.

II. Pointing-based interaction

In this section, we briefly summarize the main building blocks used to develop the contributed pointing-based user interface toolkit as illustrated in Figure 2.

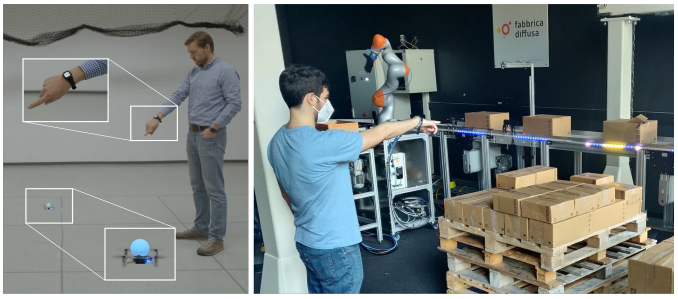

Fig. 1. Examples of deployment of IMU-based pointing interfaces built on top of PointIt. Left: a user controlling a tiny drone. Right: a user selecting packages on an industrial conveyor belt.

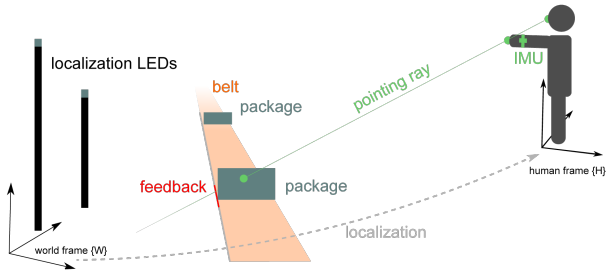

Fig. 2. Pointing-based interaction. A user, whose pointing ray is reconstructed in the $\{H\}$ frame, is localized with respect to the $\{W\}$ frame. Thus, they are able to indicate objects in $\{W\}$. In this example, using the pointing gesture, the user can select packages moving on a belt, visualizing a pointing cursor on the LED strips, which provides the user with a feedback.

A. Pointing Model

Let $\{H\}$ be the user reference frame, placed at the center of the user's feet with the z-axis pointing upwards; we implement a simplified version of the eye-finger ray cast method [14] to reconstruct the pointing ray in frame $\{H\}$: the ray originates between the user's eyes and passes through their index fingertip. We compute the user's pointing ray measuring the arm orientation with the wrist-worn IMU; we assume that the user points while keeping the arm straight and that the user measurements are approximately known (height of shoulder and eyes; arm length).

B. Relative Localization

If the user is not already localized with respect to a fixed world frame (e.g. because its position while interacting is constrained, or because it is tracked by an external component), we provide a pointing-based localization procedure [11] to estimate the pose of the user in a fixed world frame $\{W\}$. To this end, the user is asked to point at several (at least two) known world locations. This can be implemented by pointing: (i) at a robot or drone while it moves along a known path; or (ii) at two (or more) static targets with known locations, e.g. fixed LEDs turned on in a predetermined sequence and timing. During this procedure, we record the user's arm orientation, reconstruct the pointing ray in $\{H\}$, and then compute the rigid transform between $\{H\}$ and $\{W\}$ that minimizes the average angle between the measured pointing rays and the known target locations [11].

C. Feedback

The reconstructed pointing ray is not very accurate due to factors such as: the simplified human pointing model; inaccurate assumptions about the body pose (users may not point with their arm straight); sensing errors (e.g., IMU drift and noise); and inaccurate relative localization of the user in the world. Thus, to implement a convenient pointing-based user interface, we need to provide enough visual feedback about the estimated pointed location so that users can compensate for inaccuracies, adapting and correcting their pointing to achieve higher levels of accuracy. Such a feedback can be provided in different ways. For instance, while controlling fast moving drones, the position of the robot itself provides an immediate visualization of the user's pointing. With slow moving ground robots or in scenarios where a suitable infrastructure is not available, a laser pointer or a spotlight can be used to provide real-time feedback about the pointed locations. As an alternative, LED strips or matrices can display a pointing cursor.

D. Pointing-based User Interface

A pointing-based user interface allows a user to interact with a system and perform different tasks using pointing gestures. It needs three inputs: (i) the user's position in $\{W\}$, (ii) the user's pointing ray in $\{H\}$, and (iii) a map of the environment in $\{W\}$. Using (i), the PUI intersects (ii) with (iii) to compute the pointing cursor, which is a 3D point in $\{W\}$. To compute it, we need to intersect a pointing ray with the environment. For example, to provide navigation goals to a ground robot or a drone flying at a fixed altitude, we intersect the ray with a horizontal plane.

The pointing cursor is then used to provide a closed-loop visual feedback to the user. Depending on the actual task, the PUI defines specific actions that can be performed (e.g., hover to select an object), similar to how GUIs are programmed.

In general, interaction with a PUI occurs as follows:

1) the system is idle;
2) the user triggers the interaction (e.g., by pressing a button);
3) if needed, the user performs a localization procedure to compute their pose in $\{W\}$;
4) the user points at something: the system reconstructs the pointing cursor and visualizes a continuously-updated feedback about its location; a task-specific component uses the pointing cursor to control the system (e.g., to indicate a navigation goal to a robot or to select an object to be manipulated);
5) the user stops interacting (e.g., by pressing a button).

III. Software Architecture and Implementation

Our system is composed of different ROS2 nodes (see Fig. 2), which allow code modularity and reuse. Here we describe each of them, along with their expected inputs and outputs, marking them either as *task-agnostic* or *task-specific*

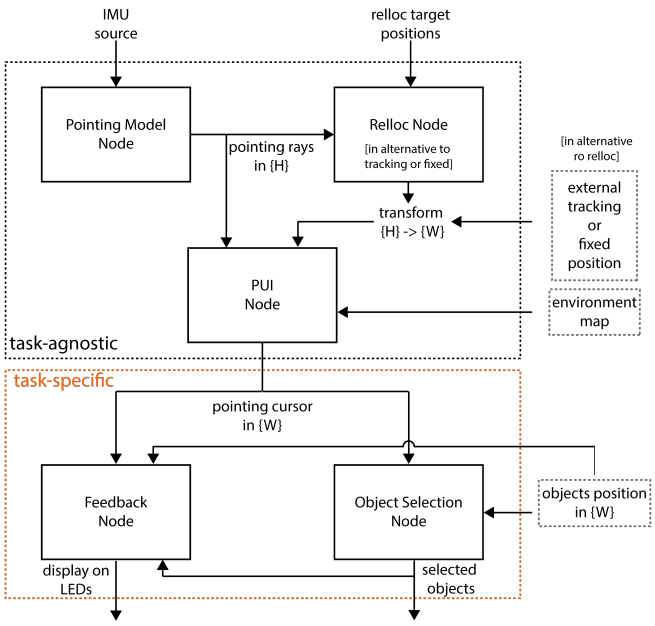

Fig. 3. Software architecture overview. Each solid black rectangle represents a ROS2 node. The arrows denote the input and output messages exchanged by the nodes. Grey-dashed rectangles on the right represent information coming from external nodes.

(for the nodes implementing the object selection functionality). We also describe the specific task used to test our interface, which is part of the code release.

A. Pointing Model Node (task-agnostic)

This node runs the pointing model described in Section II-A. Given the user's body measures, it uses the orientation returned by wrist IMU to reconstruct the pointing ray in $\{H\}$.

B. Relative Localization Node (task-agnostic)

This node is in charge of publishing the coordinates transform between $\{W\}$ and $\{H\}$, computed with the relative localization approach described in Sec. II-B. This node can be optionally removed provided that the same transform is published by an external tracking system or is manually fixed. For instance, our release contains the concrete implementation of a relative localization procedure that uses two LED lights whose positions are known in $\{W\}$: 1) both LEDs are turned off; 2) the user triggers the beginning of the procedure; 3) the first LED turns on and the user points at it; 4) the first LED turns off, the second LED turns on and the user points at it; 5) the transform is computed and published to ROS2 `tf`.

C. PUI Node (task-agnostic)

This node receives the pointing rays from the *pointing model node*, the user's position in $\{W\}$ from the *relative localization node*, and a map of the environment. All those inputs are used to output the *pointing cursor*.

In the toolkit, we implement the following approach to intersect, with a tolerance, pointing rays with objects stored in the map of the environment. We discretize objects in sets of 3D points that we store in a k-d tree. We also discretize the pointing ray to query the k-d tree for the nearest neighbor: we place the pointing cursor on the nearest neighbor if it is within the desired tolerance.

D. Object Selection Node (task-specific)

This node is specific for the task of selecting an object. It requires as input (i) the pointing cursor from the *PUI node* and (ii) a list of objects and their positions in $\{W\}$. The output is a list of selected objects.

We provide two methods for triggering the object selection/deselection. In the first method, if for all the timesteps in a time window the pointing cursor is overlaying the same object, then the object state is toggled; this method is not robust to small movements: if the cursor "flickers", the selection timer resets. The second method compensates for this problem by requiring that the object is overlaid only for a fraction of timesteps in the time window.

E. LED Strip Feedback Node (task-specific)

This node is specific to provide feedback on LED strips. In particular, knowing the LED strips locations within $\{W\}$, it displays different feedback, based on the inputs received:

- the *pointing cursor* position;
- the position of objects;
- the state of each object (whether the object is currently selected; whether the object is being hovered by the cursor).

IV. Applications

A. An interface to select packages on a conveyor belt

We used our toolkit to implement a PUI that allows users to select packages (e.g. faulty or suspicious) on an industrial conveyor belt by pointing. Selected and non-selected packages are then dispatched in different unloading bays. In this context, LED strips placed along the belts provide feedback for the pointing cursor, tracked position of each package, and state of each package (non-selected, hovered, selected). Relative localization of the user is also based on pointing and relies on two independent wall-mounted LED lights.

Our release includes code to reproduce this task both in simulation and in a real deployment. This includes ROS2 drivers to reproduce our hardware setup, nodes to control and simulate the conveyor system (e.g. packages feeding, tracking, and dispatching), nodes to produce all the available feedback, and CoppeliaSim [15] scenes to simulate the interaction. Both setups are depicted in Figure 4.

B. Other Tasks

The proposed toolkit is general and can tackle a variety of tasks. For example: using pointing to precisely control a fast quadrotor in real time [12]; using pointing to program a trajectory for a slow ground robot, using a robot-mounted laser turret to shine a pointing cursor on the floor, for feedback [3]. Additional target applications include using pointing for location or object selection for a robotic manipulator, e.g. for pick and place tasks on large workspaces.

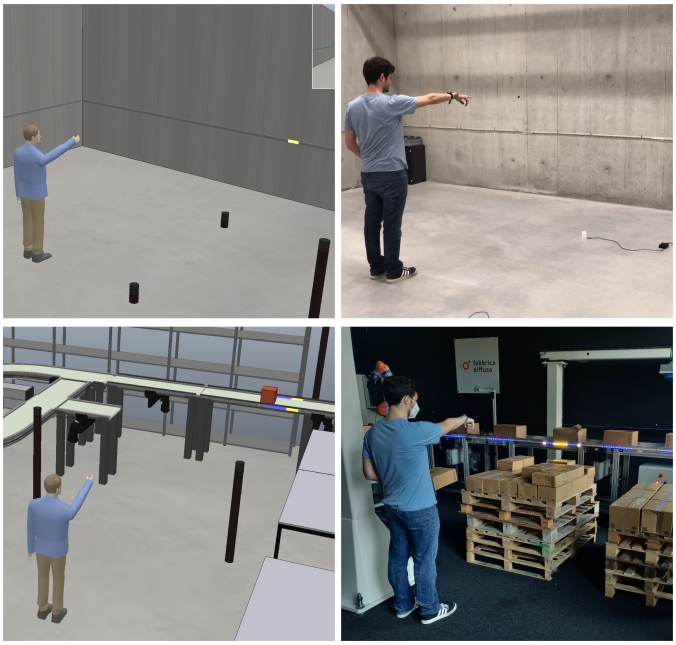

Fig. 4. Testing scenarios. Left: simulation. Right: real world. Top: users controlling a pointing cursor on the LED strips. Bottom: users selecting packages on a conveyor belt with LED feedback.

V. Software Release

Our software is available at the following link

https://github.com/Gabry993/pointing-user-interface-hri

In this repository, together with the entire source code, we provide a thorough `README` file that explains how to easily install and replicate our system using `docker` containers. This also gives the possibility to test our software in simulated scenarios, removing any hardware requirement, as explained in Section V-A. Finally, we documented all the parameters exposed by the PUI ROS2 launch files, which are also summarized in Sec. V-B. This, with the well-defined interfaces to our system, makes the components of the toolkit re-usable and suitable to be integrated into different systems.

A. Testing Our Software

We release docker containers to easily set up and run our system, with `docker` and `docker-compose` as the only software prerequisite. We provide the code to test our interface in 4 different scenarios, which can be run either in simulation or in the real world:

- `scenario_1`: the user points at LED lights to change their color;
- `scenario_2`: the user points along LED strips to move a cursor over them (Figure 4, top);
- `scenario_3`: the user points to select colored moving segments drawn on LED strips;
- `scenario_4`: the user points to select packages moving on a conveyor belt, with LED strips feedback (Figure 4, bottom).

To run the simulated scenarios, we provide a docker container containing `CoppeliaSim` and scenes recreating the environment of our laboratory. Here, a simulated user will perform the interaction. For the real-world scenarios, we provide all the information and code to replicate also our hardware setup.

The following summarizes the steps to run a scenario in docker:

1) (for simulation only) start the *simulation* container, open the desired scenario and start the simulation;
2) (for real-world only) start the *real_world* container, which will set up all the hardware related components (i.e. IMU and LEDs);
3) start the container related to the desired scenario. This will start all the PUI-related nodes and the supervisor in charge of controlling the demo state.

B. Launch Files

Another `README` file is available in the repository,[1] which describes the two main ROS2 launch files that expose all the parameters to set up the PUI:

- `user.launch` starts all user-specific nodes to interact with the system, which includes the node running the pointing model, the relative localization node, and a node supervising the state of the interaction. Multiple users need to launch one copy each with different namespaces.
- `scenario_1_2_3_4.launch`, which starts the node implementing the PUI; it requires parameters such as the path of the environment map, the time of overlay required to select an object when pointing and the pointing error tolerance to accept when computing the pointing cursor.

VI. Conclusion

We described PointIt, a novel open-source, modular, ROS2-compliant software toolkit that can be used to implement Pointing-based User Interfaces. The toolkit includes application-agnostic functionality that is widely applicable to human-robot interaction tasks involving pointing gestures sensed with wearable bracelets. Furthermore, high-level functions are provided to deal with problems involving the selection of objects when using LED strips for feedback. The code can be freely downloaded from github and includes ready-to-run examples.

References

[1] G. Butterworth, "Pointing is the royal road to language for babies," *Pointing: Where Language, Culture, and Cognition Meet*, 01 2003.

[2] J. Nagi, A. Giusti, L. M. Gambardella, and G. A. Di Caro, "Human-swarm interaction using spatial gestures," in *IEEE International Conference on Intelligent Robots and Systems*, 2014, pp. 3834–3841.

[3] B. Gromov, G. Abbate, L. Gambardella, and A. Giusti, "Proximity human-robot interaction using pointing gestures and a wrist-mounted IMU," in *2019 IEEE International Conference on Robotics and Automation (ICRA)*, May 2019, pp. 8084–8091.

[4] M. Tölgyessy, M. Dekan, F. Duchoň, J. Rodina, P. Hubinský, and L. Chovanec, "Foundations of Visual Linear Human–Robot Interaction via Pointing Gesture Navigation," *International Journal of Social Robotics*, vol. 9, no. 4, pp. 509–523, 2017.

[1] At the link https://github.com/Gabry993/pointing-user-interface-hri/tree/main/docker/pointing-user-interface/code.

[5] B. Großmann, M. R. Pedersen, J. Klonovs, D. Herzog, L. Nalpantidis, and V. Krüger, "Communicating Unknown Objects to Robots through Pointing Gestures," in *Annual Conference on Advances in Autonomous Robotic Systems (TAROS)*. Springer, 2014, pp. 209–220.

[6] S. Pourmehr, V. Monajjemi, J. Wawerla, R. Vaughan, and G. Mori, "A robust integrated system for selecting and commanding multiple mobile robots," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 2874–2879, 2013.

[7] Z. Zivkovic, V. Kliger, R. Kleihorst, A. Danilin, B. Schueler, G. Arturi, C.-C. Chang, and H. Aghajan, "Toward low latency gesture control using smart camera network," *2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, vol. 0, pp. 1–8, 06 2008.

[8] J. Sugiyama and J. Miura, "A wearable visuo-inertial interface for humanoid robot control," in *2013 8th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 2013, pp. 235–236.

[9] M. T. Wolf, C. Assad, M. T. Vernacchia, J. Fromm, and H. L. Jethani, "Gesture-based robot control with variable autonomy from the JPL BioSleeve," *IEEE International Conference on Robotics and Automation*, pp. 1160–1165, 2013.

[10] B. Gromov, L. M. Gambardella, and G. A. Di Caro, "Wearable multi-modal interface for human multi-robot interaction," *2016 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pp. 240–245, Oct 2016.

[11] B. Gromov, L. Gambardella, and A. Giusti, "Robot identification and localization with pointing gestures," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2018, pp. 3921–3928.

[12] ——, "Guiding quadrotor landing with pointing gestures," in *12th International Workshop on Human Friendly Robotics*. Springer, Oct 2019.

[13] B. Gromov, J. Guzzi, G. Abbate, L. Gambardella, and A. Giusti, "Video: Pointing gestures for proximity interaction," in *HRI '19: 2019 ACM/IEEE International Conference on Human-Robot Interaction, March 11–14, 2019, Daegu, Rep. of Korea*, Mar. 2019.

[14] S. Mayer, V. Schwind, R. Schweigert, and N. Henze, "The Effect of Offset Correction and Cursor on Mid-Air Pointing in Real and Virtual Environments," *Proc. of the 2018 CHI*, 2018.

[15] E. Rohmer, S. P. N. Singh, and M. Freese, "Coppeliasim (formerly v-rep): a versatile and scalable robot simulation framework," in *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*, 2013.