Intelligent User Interface for Roombots

Ayberk Özgür
ayberk.ozgur@epfl.ch

Abstract—Roombots are self-reconfigurable modular robots designed to study robotic reconfiguration and adaptive locomotion. One of the main goals of this platform is to be used as adaptive furniture inside human living spaces such as homes or offices. For this reason, instead of the classical Graphical User Interface methods used up to now, we propose a novel and more natural way of interaction with the Roombots modules on a Roombots grid, called the Intelligent Roombots User Interface. In our method, the user commands the Roombots modules using pointing gestures and desired target locations. The user's body is tracked using multiple Kinects. The user is also given real-time visual feedback of their physical actions and the state of the system via LED illumination electronics installed on both Roombots modules and the grid. We demonstrate how to use our interface and conclude by discussing future extensions and ideas.

I. INTRODUCTION & MOTIVATION

The Roombots (RB) platform developed at the Biorobotics Laboratory is a modular robot platform that is designed to study both robotic reconfiguration and reconfigurable locomotion [23]. Each RB module is composed of two connected “sphere-like” structures. It has 3 rotational Degrees of Freedom (DOF): One is located in between the sphere-like structures and two are located along each sphere-like structure’s diameter, as seen in Figure 1a. Active Connection Mechanisms (ACMs) can be installed on each face, letting the module connect to structured grid tiles and to other modules. The structured grid is designed such that one sphere-like structure of a module fits exactly on a grid tile. A single module can move anywhere on this grid by moving one tile at a time, latching onto the next tile before releasing the previous one using ACMs. However, free locomotion and structured motion using connected structures with more than one module are also possible. One of the major goals of the platform that incorporates all of these concepts is to autonomously create furniture, where modules move both inside and outside the structured grid environment, as depicted in Figure 1b.

Up to now, RB modules were mainly controlled by sending hardcoded motion command sequences, using a classical Graphical User Interface (GUI), or through an augmented display on a mobile tablet PC. The first is restricted to developers only. The second requires the user to focus on a computer, most often using a monitor and mouse/keyboard pair, to command the RB modules that are spatially located elsewhere. The last method requires the user to carry an additional device. To construct a User Interface (UI) that controls robots coexisting with humans, a more natural interface paradigm must be considered. An instance of such an interface would be where the user controls the robots using physical gestures and receives sensory feedback in the very same space that the robots are located without carrying additional devices.

Arguably the most natural way of controlling RB so far has been designed by Bonardi et al. [9] using a multitouch tablet PC touchscreen as input device in an augmented reality environment to place furniture made of RB modules in desired locations inside a room. Magnússon et al. used Fable, a modular robotic system, to study interaction with the user whose posture and face are detected in near real-time using a depth sensor [18]. A study by Hornecker and Buur investigates the human interaction with augmented environments [12]. They emphasize significant concepts such as the use of the user’s body as an input device exploiting the richness of bodily movement and physical objects embodying aspects of the system state so that it is legible for users. We are strongly motivated that using these concepts would increase the controllability of RB modules in a more natural way.

The idea of physical embodiment of the system state and giving sensory feedback to the user is applied by Bellmore et al. [6] in their interactive display where the user is informed of the results of their actions by playing animations on the target area of the action. A more related example is the study made by McLurkin et al. [20] where individual swarm robots are equipped with LEDs to inform the user of the robot’s state. These methods are good candidates for giving feedback to the user and enhancing system usability in our application.

Also closely related to our topic, natural multi-agent robot control interfaces are studied by Lichtenstern et al. where a subset of multiple quadrotors can be selected by pointing gestures and moved in 3D space via the user’s hand position, detected by a depth sensor [16]. Couture-Beil et al. implemented a natural interface where the user selects and commands one of...
multiple wheeled robots by face engagement (gaze) and simple hand gestures detected by an RGB camera [11]. Klompmaker et al. developed a natural interaction framework where tangible interactions with tabletop objects are detected via hand and arm tracking with a depth sensor [14]. Studies of Jojic et al. [13] and Kortenkamp et al. [15] utilize pointing gestures and discuss methods for their recognition. These studies contain useful methods for our application such as agent selection in a multi-agent setting and tracking the user’s body and gestures with depth sensors.

Motivated by the existing natural control interface studies and the lack of their application to modular reconfigurable robots, we implemented and studied such an interface for RB. For simplicity, we limited our study to the structured grid. The user is able to select individual RB modules and move them to target locations by pointing at the modules and at the targets. The pointing gestures and the grid state are recognized by a dual depth sensor setup. The user’s pointing gestures cause the emission of visual feedback on LED setups on both the grid tiles and the RB modules, indicating where the user is pointing and which object is selected. In addition to this, the system state is also exhibited on the same feedback setup, further enhancing the user interface experience. This paper is organized as follows: Hardware and software design of all components are described in detail in Section II, experiments and results are examined in Section III and in Section IV, our outlook on the study along with future extension plans is given.

II. SYSTEM DESIGN

A. Overview

Our interface, whose structure is summarized in Figure 2, is composed mainly of a depth sensor setup to perceive the grid state and the user’s body (represented with red and green in Figure 2), a visualization setup to provide the user with feedback of their actions and the grid state (represented with yellow and orange in Figure 2), and finally the motion controller to command and move the RB modules simultaneously (represented with violet in Figure 2). In the sections II-B, II-C and II-D respectively, these parts will be explained in detail.

Fig. 2. Overview of our user interface, denoting key components. Pointing gestures of the user and the grid state are recognized by depth sensors. Visual feedback using LEDs is given to the pointing gestures and to indicate the system state. Finally, the module selected by pointing gestures is moved to its target tile, also selected by pointing gestures.

Motivated by the existing natural control interface studies and the lack of their application to modular reconfigurable robots, we implemented and studied such an interface for RB. For simplicity, we limited our study to the structured grid. The user is able to select individual RB modules and move them to target locations by pointing at the modules and at the targets. The pointing gestures and the grid state are recognized by a dual depth sensor setup. The user’s pointing gestures cause the emission of visual feedback on LED setups on both the grid tiles and the RB modules, indicating where the user is pointing and which object is selected. In addition to this, the system state is also exhibited on the same feedback setup, further enhancing the user interface experience. This paper is organized as follows: Hardware and software design of all components are described in detail in Section II, experiments and results are examined in Section III and in Section IV, our outlook on the study along with future extension plans is given.

B. Depth Sensor Setup

The goal of our depth sensor setup is to capture the user’s body and the grid state as much as possible. Microsoft Kinect for Xbox [21] was selected as the depth sensor for its low price and the abundance of compatible open source software. The number of sensors, their placements and their orientations were first determined considering 3 main factors: The user skeleton tracking quality (discussed in section II-B2), the depth sensor interference and the area coverage/overlap.

A number of different setups were considered and tested in which the sensors were placed with as large angles between as possible. This tends to reduce the interference, as remarked by Susanto et al. [27] and analyzed in detail by Berger et al. [7]. The initial unsatisfactory configurations can be seen in Figure 3 where the configurations in Figure 3a and 3c failed to provide adequate skeleton tracking quality from certain user positions and the configuration in Figure 3b was infeasible since adding the third sensor did not contribute significantly.

The final configuration, seen in Figure 4 is formed of two Kinects, one of which is mainly used for user skeleton tracking and the other is mainly used for grid state tracking. The amount of sensor interference and the skeleton tracking quality in this setup was such that we were able to track the user’s upper body anywhere within 50 centimeters of the grid periphery, as long as the user’s crucial body parts (such as hands) are not hidden behind objects (e.g. the rest of their body or the grid wall) with respect to the user tracking Kinect. Also, in this setting, the amount of overlap in 3D surfaces viewed
by both of them enabled the extrinsic calibration (detailed in the next section) to converge.

1) Extrinsic Calibration: Once the Kinects are placed, their 3D transformations with respect to each other and to the grid must be measured; this allows the untransformed coordinates measured by the Kinects in the camera frame to be put in the same frame as the objects in the grid. This procedure, called extrinsic calibration, is performed by an auxiliary GUI application, seen in Figure 5.

The two Kinects are calibrated with respect to each other first. For this reason, and due to the calibration not having real-time constraints, 100 frames from each Kinect are first recorded. Each frame coming from a Kinect tends to be very noisy, and needs to be filtered. Suggestions by Susanto et al. [27] to apply a 9-frame running average filter to each depth pixel and then a 5x5 median filter to every unknown pixel seems reasonable at first. However, we observed that applying a running average filter caused spurious depth values to appear due to the mean of the existing pixels corresponding to irrelevant coordinates. For instance, a pixel directly on the border of two objects with different depths may be averaged to the midway between these depths due to noise, where none of the two objects exist. To address this problem, we apply temporal median filtering to the whole of the 100 frames and then 5x5 spatial median filtering to the unknown pixels.

The resulting depth maps are then converted into a point cloud format, provided by the Point Cloud Library (PCL) [4]. Next, these point clouds are transformed so that the “upwards” direction approximately corresponds to the Z axis of their frame, using the accelerometer data collected from each of the Kinects via the libfreenect library [2]. The point clouds are then presented to the user in a PCL Visualizer where he/she can zoom in/out, rotate and move the view in 3D. The user must do a rough alignment of the two point clouds manually by changing one of the clouds’ X, Y, Z, Roll, Pitch and Yaw transform values with respect to the other. Since the clouds are approximately “up” towards the Z axis, Roll and Pitch are mainly not required to be modified in this step.

Once the rough alignment is done, the clouds are fully aligned with a variant of the Iterative Closest Point algorithm by Besl and McKay [8] implemented in PCL. The number of maximum iterations is kept low as to allow the user to rapidly see if the calibration diverges due to bad initial alignment or lack of sufficient overlapping surfaces seen by both sensors. The algorithm can be applied iteratively to converge to better results. Once the two clouds are calibrated, an artificial point cloud representation of the RB grid is created and calibrated against the union of the two previous clouds, with manual and automated calibrations similar as before. In total, the extrinsic calibration only needs to be done once and the resulting calibration values are saved to permanent files and reused during the actual operation of the user interface.

2) Pointing Gesture Detection: In order to detect where the user is pointing at, we use the skeleton tracking facilities of the NiTE “middleware” [22] in the OpenNI framework [3]; this framework and middleware were successfully used by numerous authors for skeleton tracking in order to perform various tasks [18, 6, 26]. As in extrinsic calibration, we need to filter the depth maps before skeleton tracking; however, we now have real-time constraints. For this reason, we apply a temporal running median filter with window size 3 and then a 5x5 spatial median filter to unknown pixels. The window size is determined empirically and provides adequate smoothing while not degrading the dynamism of moving entities.

In addition, the NiTE skeleton tracker only accepts a single depth map coming from a Kinect and not point clouds. Therefore, we cannot use the union of two or more point clouds coming from different Kinects as the skeleton detection input. Instead, we project the points coming from the grid tracker Kinect into the user tracker Kinect’s frame and patch the user tracker depth map if any of these points corresponds to a pixel with an unknown depth. Once patched, the depth map is sent to the skeleton tracker, which returns the 3D locations of the 15 “joints” including head and left and right hands. These locations are observed to be noisy as well, particularly when the user’s body parts such as hands are at narrow angles with respect to the Kinect. For this reason, the joint locations are subjected to a weighted running average filter of 4 frame window size, a value determined again empirically to not degrade the inherent body motion. The 4 weights for each joint are confidence values returned by the skeleton tracker for each frame, allowing us to decrease the value of more noisy readings compared to clean ones.

In accordance with Jojic et al.’s methodology [13], the head-hand vector is extended and used as the pointing direction. Due to Kinect’s level of precision, smaller body parts than the hand (such as fingers or fingertips) are not detectable by the skeleton tracker; therefore we can only use the hand positions instead of fingertip positions. This vector is observed to be more robust compared to other direction such as shoulder-
hand and elbow-hand in terms of user body orientation with respect to the camera. Once detected, the head-hand vectors (for left and right hands) are extended and the first object that these rays hit (if any) are considered being pointed at. This method is depicted in Figure 6.

3) Grid State Detection: Using the union of the point clouds coming from the two depth sensors, the grid state perception system should detect the initial state of the grid, extracting the positions and orientations of the existing RB modules and (possibly) obstacles on the grid. While the interface runs, the perception system should also detect the possible additions or removals of obstacles, such as passive structural elements depicted in Figure 1. This will allow us to move the RB modules from one point to another in a collision-free path. Currently, we lack this perception system and initialize the grid state manually; however, it is among our future goals to study and implement such a system.

C. Visual Feedback Setup

In order to enhance the usability of the interface, we designed LED-based visual feedback systems to be integrated both to the RB grid tiles and the RB modules. Design of these systems are described in detail in sections II-C1 and II-C2 respectively.

1) Roombots Grid: During the usage of the interface, the user may point to and select grid tiles. Our goal is to illuminate these tiles in a specific way to give visual feedback concerning the physical gesture being performed. In addition, we may desire to expose certain aspects of the current system state, such as the path being taken by a specific moving RB module, on the grid. For these reasons, we designed and implemented an LED-based illumination system that enables us to illuminate each grid tile separately with a desired color. Since our goal is to illuminate grid tiles that are part of the inherently resizable and reshapable RB grid, the system design is modular. Furthermore, there may be many tiles that are illuminated (72 in our application) hence the circuits are designed to be as simple as possible to decrease production cost and time.

In our design, there are two electronic circuits, namely the LED circuit and the driver circuit, both implemented on single-sided Printed Circuit Boards (PCB) as seen in Figure 7. The LED circuit simply consists of 4 synchronous Red-Green-Blue (RGB) LEDs and a wire terminal, designed to illuminate a single grid tile. Each of these LEDs has a maximum average of 1075 mcd, 1570 mcd and 375 mcd of red, green and blue light intensity respectively. These 4 LEDs can be illuminated to a desired color simultaneously; they are not individually controllable. The driver circuit consists of a single LED driver IC (STP16CPC26 [25]) and wire terminals, capable of driving up to 5 LED circuits. The driver communicates via Serial Peripheral Interface (SPI), features a shift register and thus can be cascaded with other driver circuits. The LED boards are powered by an external power supply on a 15V bus, also cascaded through the driver circuits. When all 72 LED boards in our application are illuminated with white color (i.e. all individual LEDs are on), a maximum of approximately 65W power is drawn from the power supply.

![Fig. 6. Head-hand vector extension being used as pointing direction, the first object that the vector hits is considered “pointed at”.](image)

![Fig. 7. Roombots grid LED illumination boards.](image)

![Fig. 8. Roombots grid LED illumination topology. The microcontroller sends one bit for each LED color on each tile serially, which translate to LEDs being turned on/off by the LED drivers.](image)
At the beginning of the communication chain is an Arduino Uno [1] featuring an Atmel ATmega328 microcontroller [5], chosen for its low cost and ease of firmware development instead of integrating a microcontroller into our circuits directly. The connection topology can be seen in Figure 8. In our application, there are 15 cascaded driver circuits and 72 LED illumination circuits, where the microcontroller sends configuration bits (corresponding to the on/off state of a color of a tile) serially with an 8 MHz SPI clock. The microcontroller also communicates with the host computer running the user interface through USB, receiving LED intensities for each color of each tile. This allows us to emulate a 50Hz Pulse Width Modulated (PWM) signal with a 12-level duty cycle for each LED color, corresponding to 12 (non-linear) levels of intensity. In the end, the number of intensity levels is limited by the computational resources of the microcontroller and not by the SPI clock speed. The working state of the final mounted system can be seen in Figure 9.

2) Roombots Modules: The second part of our visual feedback consists of LED illumination electronics installed in RB modules. Similar to the grid illumination system, our goal is to give visual feedback to the user about the current system state, particularly about the associated module’s state; and about the current physical gesture being performed, such as pointing at or selecting the module.

Our design is made to be as compatible with the existing electronic and mechanical hardware design [24] as possible. It mainly follows the idea of illuminating the two outer diametrical DOFs instead of central points on the faces or corners of the module’s cubic structures. The benefits of this choice are twofold: First, illumination visibility from various angles is significantly higher than using any of the faces or edges due to simple geometry; second, we gain the ability to indicate the turning motion of the associated DOF via LEDs. Therefore, in order to illuminate these DOFs, we designed the electronic circuit whose implementation is visible in Figure 10. Each RB module has two such boards illuminating the outer DOFs through semi-transparent rings; these boards are mounted in the H1 and H2 hemispheres as seen in Figure 10c. The board’s shape is designed to accommodate the battery packs, motors and other electromechanical hardware already existing in these hemispheres. In this study, the DOFs equipped with LEDs are limited to the outer ones, but implementing a similar circuit for the middle DOF is among our future works.

Following the existing electronic hardware design for various boards of the RB module [24], we utilize the dsPIC33FJ128MC802 microcontroller connected to the existing RS485 bus through the ADM3078E interface IC. There are 6 equidistant RGB LEDs on the rim of the circuit, mounted such that the light is emitted outwards from the center. In our application, each LED has a maximum average of 55 mcd, 115 mcd and 30 mcd of red, green and blue light intensity respectively. The 18 individual color LEDs are driven by two LED driver ICs (MAX6965 [19]), communicating with the microcontroller via I2C, capable of driving LEDs individually by 16 level duty cycle PWM signals. All components are powered by step-down converter ICs (LTC3630 [17]) capable of generating 5V and 3.3V outputs efficiently off of the regulated 6V power bus in the current implementation. However, they also accept an unregulated input between 12-20V in the absence of a regulated 6V power bus, which is the planned design of the next version RB electronic hardware. During testing, approximately 130mA of current is observed to be drawn from a 12V external power supply when all RGB LEDs are white (i.e. all individual LEDs are on). The working state of the system can be seen in Figure 11.

The firmware on the microcontroller is designed to produce various effects on the LEDs; these effects are used to convey
different states of a module to the user:

- **Off**: All LEDs are off.
- **Constant**: All LEDs are lit in a single color in full intensity.
- **Breathe**: All LEDs are lit in a single color, intensity decreasing and increasing continuously with linear interpolation.
- **Turn**: LEDs are turned on and off sequentially and in a circular fashion in a single color, fading from one to the next with linear interpolation. May be clockwise or counter-clockwise.

Other effects can also be added to the firmware in the future. Cross-fading during the passage from one effect/color to another with linear interpolation is also implemented for aesthetic reasons. The host computer connected via Bluetooth commands a LED module by sending an effect and a color. Due to the low intensity resolution, we limit the usable colors to red, green, blue, yellow, cyan, violet and white, also reducing the communication overhead and increasing visual distinguishability of one color from another.

### D. Moving the Modules & User Interaction

The gesture detection and visual feedback systems described in the previous sections (II-B and II-C respectively) are combined to design an intelligent user interface in order to control the RB modules. In this stage, movement primitives and paradigms proposed by Bonardi et al. [10] are utilized. For simplicity, we limit ourselves to the structured motion of single RB modules on a planar grid, moving exactly one tile at a time. However, this can easily be extended to motion on a much broader grid structure by considering individual planar grids that share convex or concave edges when planning paths. These edges would have to be overcome by synchronized movement of the outer DOFs and by using more than one module respectively. Achieving this with real hardware is among our top priority future goals.

In order to command the individual RB modules from a host computer, a multithreaded software architecture is designed where a worker thread and a thread-safe task queue is allocated to each module. The worker thread transmits the low level motion commands and queries the module continuously whether the goals in the commands are reached, thus implementing closed-loop joint control. It also allows commanding paths to multiple modules simultaneously, allowing concurrent motion.

To accommodate the motion of the modules, a representation of the grid is built where voxels (cubic volume that encompasses half of a RB module) may be occupied or free. The voxels that are occupied by any moving module are marked as such in real time on this representation. Before moving, a module must allocate (in a thread-safe manner) all of the voxels that will be occupied during the motion. If any of these voxels are occupied or previously allocated, the module cannot allocate them and is not allowed to move. This enables concurrent collision-free motion of multiple modules and is elegantly compatible with the proposed D* algorithm for path planning [10]. Currently, path planning is not implemented yet and the paths are manually entered and executed. It is the very next step that will be taken in the near future of our study.

The user interaction begins by all RB modules idle, illuminated with the breathing effect in white color emphasizing the idle state. The grid tiles are not illuminated when idle, differentiating them from the “living and active” RB modules, another impression we expect to create on the user by the breathing effect. The user interacts with the modules and grid tiles by pointing at them with either hand; any tile or RB module that is being pointed at is illuminated with a cyan color, while the RB modules are still in breathing effect emphasizing the idle state. This allows the user to know where he/she is pointing at, and forms a closed-loop control mechanism to help the user correct the pointing direction if a wrong object is selected. Since we are tracking the head-hand vector, the user has to cover the object with their hand in their field of vision to point at it; this information is given to the user in the beginning to accelerate the adaptation process.

To select a tile or module, the user points at it for more than 2 seconds. When an object is selected, it is illuminated with a constant yellow color. As the extension of the visual feedback mechanism to indicate the object being pointed at, an already selected object is illuminated with a constant green color. When a RB module is selected when there is another one already selected, the previous module is deselected; tile selection behaves similarly. Selection of multiple tiles or modules is not yet designed or implemented. The user can also deselect an object by pointing at it for more than 2.5 seconds after having pointed away from the selected object at least once. This prevents constant select/deselect loops when an object is being pointed at indefinitely. As soon as a module and a grid tile is selected, the module is commanded to move to that tile, provided that there is a collision-free path. While moving, the module is illuminated with a violet color and the rotating joints are indicated via illuminating their associated LEDs in a turning effect.

During the operation of the interface, the user is presented with a simple representation of the system in a PCL Visualizer window in the host computer, seen in Figure [12]. Here, the user can zoom in/out and move in 3D, viewing the state of the grid and all modules on it. The point clouds (downsampled for better visibility) and head-hand vectors of the user (when available) are also presented. This extension is particularly useful for development purposes and when not all grid tiles or modules can be equipped with LED illumination boards due to lack of sufficiently many boards.
In order to test our interface, we equipped 12 grid tiles and one RB module with prototype LED illumination boards. We used two RB modules to test selection and simultaneous path execution with multiple modules. Due to the current lack of path planning, we request the user to command the modules to move to predetermined tiles where we know the path to take beforehand. Since the paths are not dynamically calculated, we simply stop the module when it cannot move due to obstruction from another module. When the obstruction clears, the user can reselect the module and reissue the command.

In order to demonstrate our visual feedback setup better, the modules are not placed and moved far away from the tiles that are equipped with LEDs. This limits the distance the modules travel. Since there is one module and some tiles unequipped with LED boards, the virtual visualizer was required to be used during testing. An example run of our interface is depicted in Figure 13, a video of which can be found at http://biorob2.epfl.ch/utils/movieplayer.php?id=274. In the near future when path planning and grid perception are designed and implemented, more tiles and modules will be fitted with LED boards to do more extensive tests. Following this, proper user study will be conducted with people completely unrelated with the study in order to justify our interface paradigms against the state-of-the-art GUIs.

IV. CONCLUSION & FUTURE EXTENSIONS

In this study, we have presented an intelligent user interface to control the RB modules on the RB grid. We introduced our design for a dual depth sensor setup to track the user’s skeleton in order to use the pointing direction to select target RB modules and grid tiles. We described our designs for LED based visual feedback hardware installed on RB modules and grid tiles. Finally, we combined the two aspects with a multithreaded motion controller to execute the user’s commands.

The current state of our study lacks the grid state perception design and path planning implementation which are handled
manually. For a truly intelligent user interface, both are indispensable; their design and implementation is the next step that will be taken in this study. Selecting and commanding multiple RB modules is certainly an idea deserving of a study, as well as enabling locomotion on multiple connected grid surfaces, such as multiple floors, walls or ceilings. Finally, a more comprehensive range of user gestures should be investigated; for instance, this may allow the user to stop the moving modules or describe the target module orientations as well as locations.

ACKNOWLEDGEMENTS

The authors would like to express their thanks to André Badertscher for his technical aid and to Dr. Alessandro Crespi for his technical aid and advice.

REFERENCES