Exploiting a touchless interaction to drive a wireless mobile robot powered by a real-time operating system

Davide Calvaresi
Scuola Superiore Sant'Anna
d.calvaresi@sssup.it

Andrea Vincentini
Scuola Superiore Sant'Anna
a.vincentini@sssup.it

Antonio Di Guardo
Scuola Superiore Sant'Anna
a.diguardo@sssup.it

Daniel Cesarini
Scuola Superiore Sant'Anna
d.cesarini@sssup.it

Paolo Sernani
Università Politecnica delle Marche
p.sernani@univpm.it

Aldo Franco Dragoni
Università Politecnica delle Marche
a.f.dragon@univpm.it

Abstract

Nowadays, touch-based user interfaces are widely used in consumer electronics. Recent trends confirm the high potential of touchless interface technologies to manage also Human-Machine Interaction, in scenarios such as healthcare, surveillance, and outdoor activities. Moving from pointers, keyboards or joysticks to touch-screens represented a significant challenge. However, a touchless approach needs to ensure intuitiveness and robustness. This paper describes a framework enabling the wireless control of a mobile robot through a contactless controller. The data provided by a complex sensor, composed of two stereo cameras and three IR sensors, are processed with custom algorithms that recognize the movements of users' hands. The result is promptly translated into commands for the robot running a real-time operating system. Usability tests confirm the compelling employment of contactless controllers for mobile robots and drones both in open and closed environments.

1 Introduction

In the recent decades, human beings are living in a technological revolution. The frenetic advent of technological devices is changing peoples daily life. Widespread consumer devices, new generation sensors, single-board computers, and other components, foster the design of many heterogeneous and different user-centered systems.

Although there are different kinds of systems with different peculiar goals, even within the same domain, they all face a common challenge: interaction with the users [Suc87] guaranteeing the compliance with their needs [CCS$^+$16]. Indeed, all systems involving human interaction provide a way to support a two-sided communication: human to machine (interfaces to manage inputs, command, etc.) and machine to human (interfaces to perceive the systems' feedback) [Min02, WG95]. To have robust, dependable and usable interfaces, they have to be accessible, as much intuitive as possible, and to respond in a time coherent with human requirements (if the interaction is too fast or too slow the user gets easily lost) [Fis01, Dix09].

Over the years, to produce more powerful interfaces, programmers tried to couple syntax and semantics, trying to associate the dynamics of a command to the actual output (gesture or movement) [Bro98]. Indeed, around the 40's, multi-purpose systems only employed panels with buttons (evolved into the modern keyboard), while systems dedicated to more specific contexts required properly designed interfaces. For example, in the case of moving mobile objects, both directly or indirectly, the joystick (emulating the older cloche employed in the aviation) became in the 80s the default choice for electronic controllers [Wol88, Gra81]. Even tough it was widely declared a big failure when employed as a pointer [CEB78], it kept evolving (maintaining its original basic features) becoming a standard interface for systems moving/driving objects.

The advent of the new touchscreen devices radically changed the human experiences when approaching technological devices. However, to face specific tasks like moving objects (e.g., gaming, etc.), even tough the touchscreen revolutionized the way of thinking about "how to perform an action", a simulated joystick still represents the more intuitive means to translate an intention to an action. In line with the current technological trends, this paper aims at moving a further step on behalf of the old joystick, bringing it in the touchless era [MBJS97].

The presented solution is a system, based on multiple devices, which detects hands movements, translating them into joystick-like commands, and finally communicating them to a mobile robot, that behaves accordingly to the received instruction. The rest of the paper is structured as follows: Section 2 presents an overview of the framework, its structure and its main components and functionalities. Section 3 addresses the system's requirements, functionalities and challenges. Section 4 presents the conducted tests summarizing their results. Finally, Section 5 concludes the paper, presenting the lesson lernt and the future works.

2 System overview

This section introduces the framework, presenting an overview of its structure, its main components and their functionalities.

The framework is mainly composed of three modules:

- the *User Interface Module* (UIM) , Figure 1(x);
- the *Communication System* (CS), Figure 1(y);
- the *Mobile Robot* (MR), Figure 1(z).

The interactions between the components of the framework are represented by the blue arrows in Figure 1.

Currently, the technological market offers a broad range of solutions for gathering and processing hands' position in space/time. Vokorokos, et al. [VMC16] analyzed the efficiency of the three most relevant devices on the market: the *Leap Motion Controller* [Leab], the *Microsoft Kinect* [Zha12] and the *Myo Armband* [Nuw13]. Depending on scenarios and settings, those devices showed up different peculiarities. For example, the *Microsoft Kinect* enables to track the entire body motion within a detection range of $0.5m$ and $4.5m$ [FP11], the *Myo Armband* is a wearable bracelet reading the muscular' electric impulses, that guarantees a high level of comfort, however limiting the amount of information that can be acquired. Compared with the previous ones, the *Leap Motion Controller* represents a better trade-off between flexibility, time-to-prototype, and precision [MDZ14], and thus it has been chosen for the presented project.

The applications in the virtual reality domain are increasing remarkably, both in presence [EFAA15] or in absence [PBAR15] of haptic feedback. Such solutions mainly aim at moving and virtualizing real objects or their models, and entire environments while enabling interactions with them. The *Leap Motion Controller* is massively employed in those systems, like the presented one, providing a visual rather than a haptic feedback [BGG+14, FSLS+16].

Briefly analyzing the *Leap Motion Controller*, it is a sensor composed of two stereo CMOS image sensors and three infrared LEDs. Such a sensor, coupled with a set of libraries, besides exporting the image of the area over the sensor, exposes APIs to get the kinematics of the hand and all its joints, thus enabling the hands' acquisition as in Figure 1(a). The *Command Generator Module* in Figure 1(b), running on a PC, operates over the obtained data to extract relevant information, and decide which commands to send to the mobile robot through the *Communication Client* in Figure 1(c). The communication system, running a custom *Communication Protocol* over an XBee connection [Fal10], Figure 1(d), transfers the commands detected by the User Interface Module to the *Communication Server*, Figure 1(e). The *Decoding Module*, Figure 1(f), elaborates the received information which is then transferred to a buffer shared by the two *Motor systems*, Figure 1(g), in charge of finally performing the robot's motion.

2.1 System's timing requirements

Robotic actuation and motion are widely known as safety-critical systems. Thereby, the guarantee that any task in the system respects its deadline, or at least that the system remains operative during possible deadline misses, is mandatory. To operate correctly, the system has to perform a series of periodic tasks. Those can be characterized as follows:

- $\Delta t_{det} \leq 50$ ms is the interval of time between the instant at which the user performs a command and its detection.
- $\Delta t_{el} \leq 20$ ms is the interval of time between the command detection and the message transmission to the *robot*.
- Δt_{com} is the time required for the communication to take place. It is unpredictable, but it can be minimized accepting that some data can be lost.
- $\Delta t_{act} \leq 50$ ms is the time between the reception of the message by the robot and the actual motion.

To obtain a smoother speed profile of the robot, a filter operation can be performed with an additional delay $\Delta t_{steady} \leq 150$ ms.

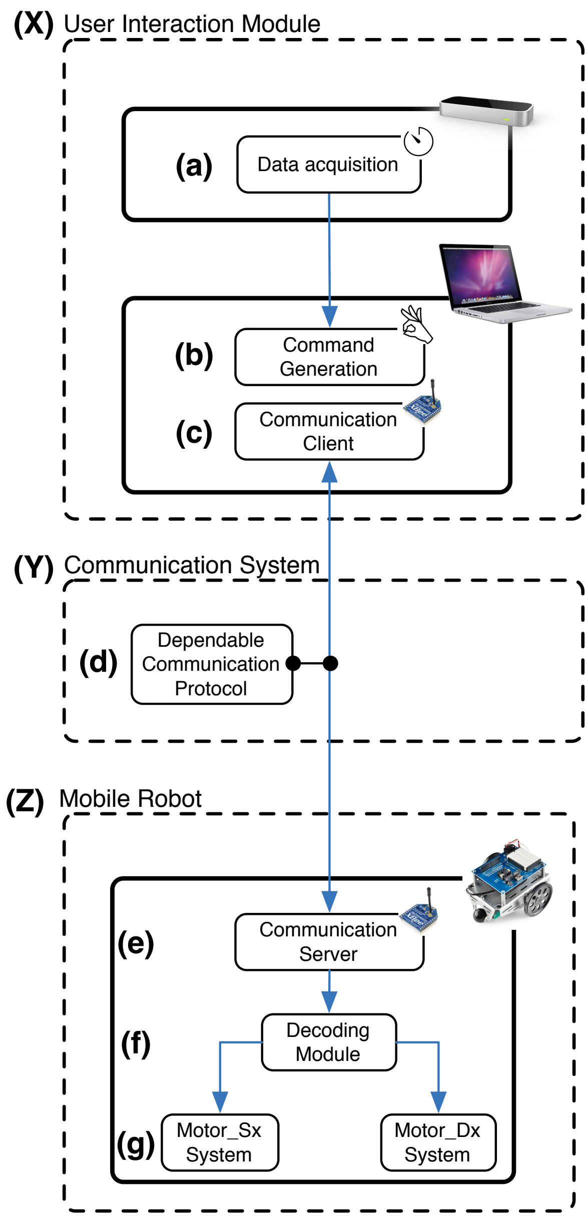

Figure 1: Frameworks' components

2.2 System's timing implementation

Meeting the timing requirements expressed in Section 2.1, the *Robot Module* and the *User Interaction Module* are implemented using real-time policies [But11]. The threads composing the *User Interaction Module* are Posix compliant [Har] and have periods of:

- 15 ms: thread handling the serial communication (thread tick);
- 50 ms: thread detecting the hands' motion and the commands creating the messages to be sent (thread hand).

A Rate Monotonic scheduler is employed to schedule the threads composing the *Robot Module*, which is powered by the *Embedded Real-Time OS ARTE [BBP$^+$16]*. The threads periods are:

- 10 ms: thread handling the serial communication;
- 40 ms: thread decoding new messages and managing the two servo motors.

3 System design

The *User Interaction Module*, the *Mobile Robot Module* and the *Communication System* can be characterized by different requirements, functionalities and challenges, that are presented in this section.

3.1 User Interaction Module

The *User Interaction Module* (UIM) is intentionally released without a GUI, since a more natural interaction is obtained by simply having the user actions reflected by the movements of the mobile robot. The UIM is implemented in the C++ language, and is platform independent. The only requirement is the support for the library of the *Leap Motion Controller* [Leaa]. Since the *User Interaction Module* is touchless, the user is supposed to be able to perform at least basic hands' movements. The interaction area is about one cubic meter centered above the sensor. The hands are continuously tracked within this area, and when specific paths, positions or movements recalling the semantic of a joystick are identified, the related commands are communicated to the robot.

The *Data acquisition* and *Command generation recognition* enable three functional units:

The *Controller* detects the hands' position, movements, speed, and rotation radius expressed according to the Cartesian axis placed as shown in Figure 2. Custom messages' structures defined by the *Communication Protocol* encapsulates such parameters.

Such a *Communication Protocol*, detailed in Section 3.2, manage a circular buffer containing the aforementioned messages, discriminating which have to be sent or deleted because out of date.

Two modalities have been implemented to drive the robot:

- Normal mode - the robot is driven like a normal 3-wheel car. The commands are: stop, go forward or backward, turn right or left.
- Rotatory mode - the robot rotates around itself, and the only commands are: rotate or stop.

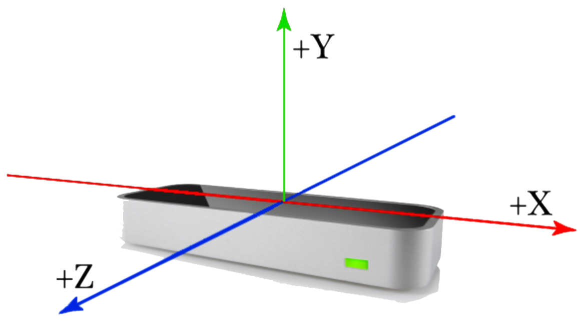

Figure 2: Leap Motion's axis

Normal Mode

The *stop*, "rest" state of the joystick, is established with a minimum margin of tolerance of 1 cm, around $(0, 20, 0)$ cm on the area above the *Leap Motion Controller* [WBRF13]. If the hand is perceived in such an area (no movements are required) a single message is sent requiring the robot to hold the position.

The *go forward or backward* is determined by the hand's position on the z axes: $-z$ move forward and $+z$ move backward. The distance of the hand from the origin of the z axes defines proportionally the speed of the robot.

The *turn left or right* is similar to *go forward or backward* but it is referred to the x axes: $-x$ turns left and $+x$ turns right. The distance of the hand from the origin of the x axes defines proportionally the rotation radius to be followed by the robot.

Rotatory Mode

By rotating the hand upside down and then back again activates the rotating mode. The command "rotate" defines the rotation speed by moving the user hand along the $\pm\ x$ axis.

3.1.1 Enhancement of usability and fluidity

The information acquired from sensors like the Leap Motion might introduce uncertainty or noise. Such a noise can generate incorrect or redundant information. Thus, undesired behavior might take place, generating feedback different from the expected. To avoid this possibility and ensuring a consistent reading, two techniques have been designed and implemented: the first is sampling in frequency, and the second is sampling the interaction area. The sampling frequency is about 20Hz, and to be sure that the identified command is correct, the user's hands have to be detected in the same position for at least 3 consecutive frames provided by the Leap Motion sensor. Recalling that the interaction area above the Leap Motion sensor is about a cubic meter, it is linearly clustered. Such a choice ensures fewer variations of the hand's position, thus generating fewer messages. The size of the partitions of the area are chosen accordingly to the feedback provided by the motors: passing from one area to an adjacent one results in a change of speed by a factor of 5. Such solutions provide a higher reactivity and a smoother and more usable mobile robot.

3.2 Communication Protocol

The users' experience and the actual system usability gain significant benefits by employing a wireless communication (by exchanging messages) between the *UIM* and the *Robot*. The proposed protocol aims at being robust.

Indeed, the TCP [KR07] inspired its design (in terms of ACKnowledgement mechanism and of timing constraints). This protocol aims at guaranteeing that if more than one message is queued in the buffer (more than one command is detected in a relatively short period, or possible communication delays happened) only the most recent is delivered to the robot. These behaviours are modeled by the state machine in Figure 4.

3.2.1 Protocol operating principles

One of the requirements for the custom designed protocol were the ability to handle undesired retransmissions in order to avoid delays and to guarantee the highest possible responsiveness of the system. Thus, during its development we have taken some concepts from the TCP protocol. In fact, the implemented mechanism to reduce the communication overload and the number of unneeded computations performed by the robot is shown in Figure 3(b).

In particular, the developed protocol allows to use 16 different message formats. The formats employed in this system's version are two: one communicating the ACK, Figure 6(a), and one communicating speed and rotation angle, Figure 6(b). The messages' structures are detailed in Section 3.2.2 and represented in Figure 6.

If a message is available, it is sent from the *User Interaction Module* to the *Robot*. The Controller keeps generating new messages (related to the identified commands) while it is waiting for an ACK from the Robot.

If the Controller receives an ACK with the expected Id (equal to the sent one), it means that the sent command has been received by the robot, so the buffer is scanned looking for the next message to be sent, Figure 3(a). On the Robot side, the *Communication Server* receives the messages sharing the information with the *Decoder Module* through a shared buffer while sending an ACK message (containing the same Id) back to the *User Interaction Module.*

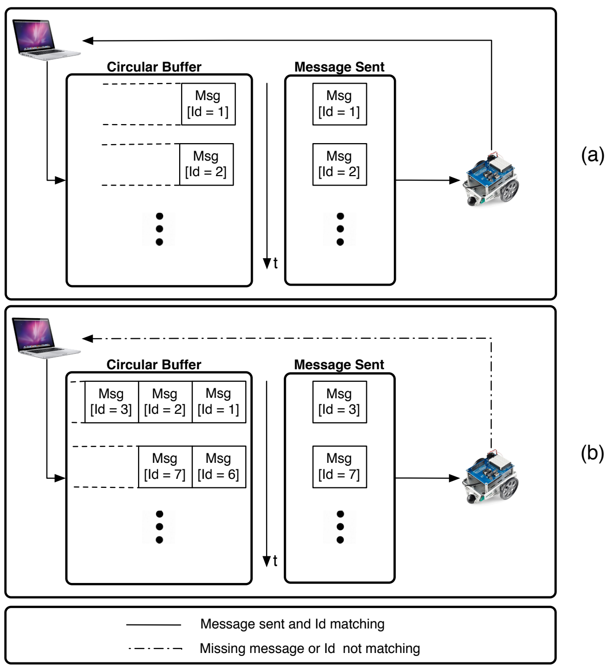

Figure 3: Communication protocol: (a) Normal scenario, (b) error or delay handling scenario.

If the Id contained in the ACK received by the *Communication client* does not match the expected one, or if no ACK is received before a predetermined period of time (named timer), the last message (the most recent) inserted in the controller buffer is sent removing all the older messages, Figure 3(b). Moreover, an internal timer is set once a message is sent. If such a timer expires before the UIM receives a proof of reception (ACK), the message is sent again.

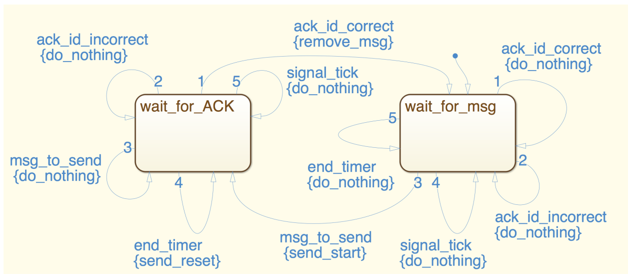

Figure 4: State machines of UIM's communication behaviours - Simulink model

Promoting the robustness analysis through formal verification, two virtual state machines are used to model the *Communication Client* (UIM), Figure 4, and the *Communication Server* (MR), Figure 5.

3.2.2 Messages structure

The structure of the messages has been inspired by the message structure of the MIDI protocol [Mid] (MSB to LSB). As aforementioned, the system uses two formats for the messaging, Figure 6.

Similarly to the MIDI protocol, the messages are composed of *StatusByte* (Figure 6(x)) and *DataByte* (Figure 6(y)). The StatusByte (8bits) contains:

- *Header identifier* [1 bit]
  The value identifying header is 1;
- *Message Id* [3 bits]
  Used for feedback mechanism;
- *Type of message*[4 bits]
  If the message is an ACK the value is `0b0000`, if the message contains data the value is `0b0001`.

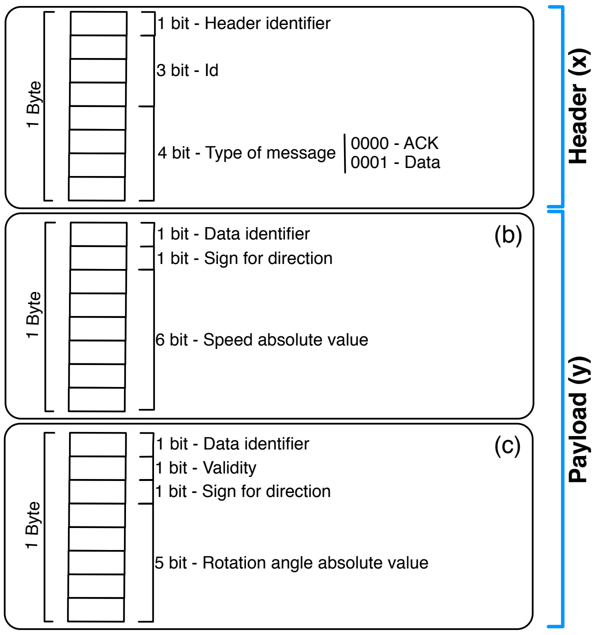

Figure 6: Messages architecture and formats: (x) StatusByte, (y) Payload [Speed (b) and Rotation radius (c)]

If the *Type of message* has value `0b0000`, the *Robot* knows that the message is complete (there are no more bytes/information to be read). An ACK message is generated by the *Robot* and it contains only the *StatusByte*. If its value is `0b0001`, the *Robot* knows that the payload contains two more bytes of information. When a message is sent from the *UIM* to the *Robot*, 3 bytes are sent: one *StatusByte* and two *DataByte* as payload.

The *Speed* format, Figure 6(b), is composed of the following three parts:

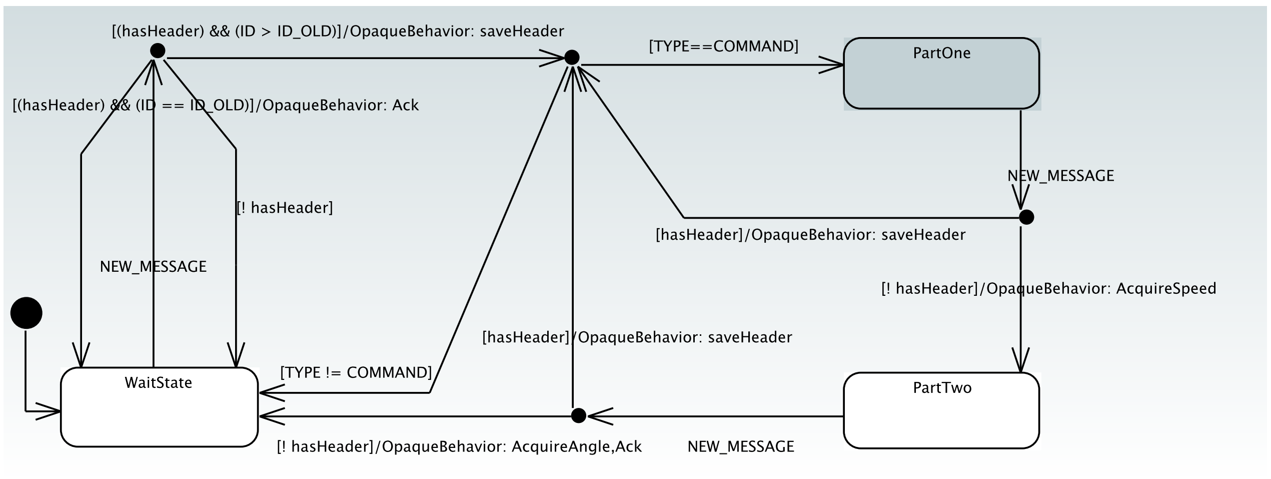
[(hasHeader) && (ID > ID_OLD)]/OpaqueBehavior: saveHeader
[TYPE==COMMAND]
PartOne
[(hasHeader) && (ID == ID_OLD)]/OpaqueBehavior: Ack
NEW_MESSAGE
[! hasHeader]
NEW_MESSAGE
[hasHeader]/OpaqueBehavior: saveHeader
[! hasHeader]/OpaqueBehavior: AcquireSpeed
[hasHeader]/OpaqueBehavior: saveHeader
[TYPE != COMMAND]
WaitState
PartTwo
[! hasHeader]/OpaqueBehavior: AcquireAngle,Ack
NEW_MESSAGE

Figure 5: Communication Server's (Robot Module) state machine - SysML model

- *Data identifier* [1 bit]
  The value identifying data is 0;
- *Direction* [1 bits]
  Go forward is identified by 0 and go backward by 1;
- *Speed value* [6 bits]
  The absolute value of the speed to be approached;

The *Rotation radius* format, Figure 6(c), is composed of the following three parts:

- *Data identifier* [1 bit]
  The value identifying a data is always 0;
- *Validity* [1 bits]
  The straight motion (rotation $= \infty$) is identified by 0;
- *Direction* [1 bits]
  Turn left is identified by 0 and turn right by 1;
- *Rotation radius* [5 bits]
  The absolute value of the rotation angle to be approached;

3.3 Robot Module

The *Robot* operates on horizontal or reasonably sloping surfaces. The possible Robot's movements, described in Section 3.1, are powered by two continuous speed step motors, controlled by the Arduino board running ARTE and finally equipped with a Li-po battery.

The system powering the *Robot* is composed of four elements:

- the *Communication Server* receives the messages as presented in Section 3.2;
- the *Decoding Module* elaborates information about speed and rotation radius encoding the actual commands for the *Motor Modules*;
- the two *Motor systems* (one for each motor) manage the servo dynamics.

3.3.1 Decoding Module

The *Decoding Module* elaborate the commands depending on the required behaviour:

- curving;
- straight motion;
- motion around itself.

Normal mode: Curving

Rotating the robot around the axes centered on itself requires different speeds on the wheels (Figure 7(a)). Defining speed (v) and rotation radius (r), the wheels' speed is calculated as shown in equation 1:

$$\begin{aligned} V_{right} &= v - \delta(v,r) \\ V_{left} &= v + \delta(v,r) \end{aligned} \tag{1}$$

Figure 7(b) identifies the *differential speed* enabling the turning action (named $\delta(v,r)$) which is obtained as shown in equation 2:

$$\delta(v,r) = C\frac{v}{r} \tag{2}$$

where C is the distance between the wheel and the center of the robot wheelbase.

Normal mode: Straight motion

The speed of two motors is simply set equal to v.

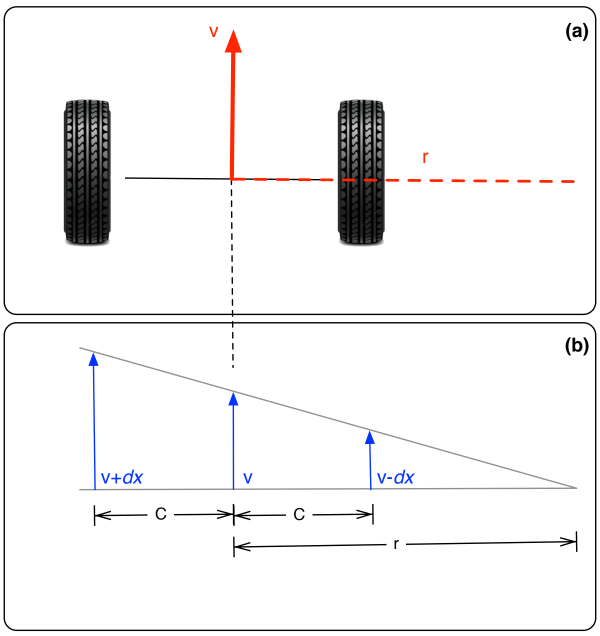

Figure 7: Speeds diagram in the case of Curving (Normal mode) and positive rotation radius.

Rotatory mode: Motion around itself

The velocities of two motors are opposite and their absolute value is equal to v (the sign depends on the direction of rotation).

3.3.2 Motor system

When the *Motor System* receives a command, it converts the speed or rotation radius into engines' instructions. Thus, developing a linear profile for the wheels' velocities, it needs particular care to prevent sliding and overturns.

Figure 8 exemplifies how a speed profile can be realized when a new desired speed is requested before the previous one is reached:

- at $t = 0$ a speed request arrives, Figure 8(a). The system provides to the motors a linear speed profile to satisfy the request in a fixed ΔT;
- at $t = 3$ (with $t < \Delta T$) a slower speed request arrives, Figure 8(b). The system calculates a new profile starting from the current robot's speed;
- finally, the stop command arrives at $t = 6$, Figure 8(c). Since no more new requests arrive in a period where $t < 6 + \Delta T$, the robot is stopped within the expected ΔT, holding that speed (in this case 0) till a new command arrives.

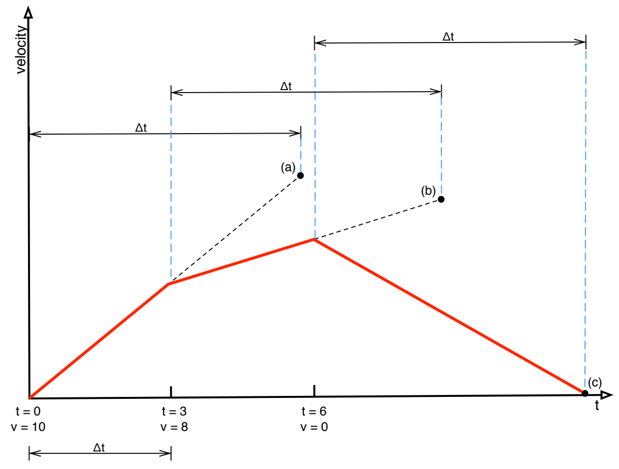

Figure 8: The speed profile produced by the controller in the cases of three different speed requests arrived before the robot reach such values

4 Testing and Results

The presented system is composed of several elements. Considering that design and implementation errors can happen in several of those elements proper testing is needed. As manual testing is a cumbersome procedure, test automation is highly needed. Thus, we searched for existing test automation frameworks and decided to adopt the CUnit [Ham04] and the gcov systems [Rid04]. Furthermore, Usability Testing is a key aspect when dealing with HMI systems. In the rest of this Section we present what has been adopted in this work.

4.1 CUnit Test e gcov

Performance analysis and dynamics verification are two strategic tests. A useful framework to automatically test functions code is CUnit. Such a framework has been employed to check the coherence of the expected outputs of Command detection and messages generation, reception and transmission which are implemented as state machines. Code Coverage Analysis [CCA] is crucial to identify and purge the dead or obsolete code from the system. Moreover, the coverage analysis counts the number of times each program's statement is executed. Finally, the profiling tool Gcov is used to increase the code optimization.

4.2 UIM usability test

The *User Interaction Module* aims at proposing a set of manageable and effective commands.

To test their functionality several approaches are viable, and the most effective are the *usability test* [DR99] and the *usability inspection* [Nie94]. The first approach is a *user-based method* that involves the end users in the testing phase, while the second is an *expert-based method* that during the testing phase involves only experts and context aware users. Both of them aim at identifying possible issues related to the usability. They mainly differ in the categories of the identified problems. Indeed, although the *usability test* identifies fewer issue's occurrences, these are more general and frequent issues [JMWU91].

The first usability test conducted concerns the types of commands: different semantics (joystick, steering wheel, etc.) were taken into account. This first experiment, named *A/B test*, consisting in executing the basic commands as move forward or backward and turn left or right with different speed and rotation radius, was useful to gather testers' opinion about efficacy, accuracy, and personal perception. Indeed, 27 testers out of 30 expressed their preference for the joystick-like interface. The second test, named *Hallway testing*, involves different groups of testers randomly selected to test the whole system. The test consists of driving the Robot between a series of obstacles. After a few commands, the testers gained confidence with the system and easily accomplished the test. Imagining to use a Cartesian system to represent the lessons learned, putting on the vertical axes the number of attempts and on the vertical axes the coefficient of experience matured during tests, the output is a learning curve which can be approximated by an exponential function. Finally, even though the current system does not provide a GUI (the system's feedback is directly provided by the motion of the robot) all the testers appreciated the system's response, classifying the system as "user-friendly".

5 Conclusion

The presented project aimed at bringing a traditionally physical user interface for human-machine interaction like the joystick in the touchless era. In the proposed work, a touchless joystick has been realized and employed to drive a wireless mobile robot customized for that purpose. All the system's components have been tested and satisfied the conducted formal verifications. Thanks to the adoption of real-time policies, the timing constraints have been respected throughout all tests. Moreover, according to the testers' feedback, the whole system was easy-to-use, responsive and effective. The current system's version perceives both the user hands, but it uses only one of them to get the commands to guide the robot. The next step is introducing the acquisition and elaboration of the second hand in charge of handling sensors equipping the robot. An initial idea we are working on is to equip the robot with a mobile camera. Such a camera could be handled (zoom, movements, taking pictures, etc.) by the user with one hand while driving the robot with the other. This new feature involves updates in the data acquisition, communication protocol, and the actuation module. While the data acquisition and the actuation modules have to be restructured, the communication protocol, thanks to its implementation, requires only to be extended with the definition of the new types of messages. An alternative might be including voice commands to control the camera and empower the interaction with the robot [CSM+16]. Finally, the introduction of a GUI is under evaluation to enrich the user experience when operating on the mobile camera.

References

[BBP+16] Pasquale Buonocunto, Alessandro Biondi, Marco Pagani, Mauro Marinoni, and Giorgio Buttazzo. Arte: arduino real-time extension for programming multitasking applications. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, pages 1724–1731. ACM, 2016.

[BGG+14] D Bassily, C Georgoulas, J Guettler, T Linner, and T Bock. Intuitive and adaptive robotic arm manipulation using the leap motion controller. In *ISR/Robotik 2014; 41st International Symposium on Robotics; Proceedings of*, pages 1–7. VDE, 2014.

[Bro98] C Marlin Brown. *Human-computer interface design guidelines.* Intellect Books, 1998.

[But11] Giorgio Buttazzo. *Hard real-time computing systems: predictable scheduling algorithms and applications*, volume 24. Springer Science & Business Media, 2011.

[CCA] CCA. Code Coverage Analysis. `http://www.bullseye.com/coverage.html`.

[CCS+16] Davide Calvaresi, Daniel Cesarini, Paolo Sernani, Mauro Marinoni, Aldo Franco Dragoni, and Arnon Sturm. Exploring the ambient assisted living domain: a systematic review. *Journal of Ambient Intelligence and Humanized Computing*, pages 1–19, 2016.

[CEB78] Stuart K Card, William K English, and Betty J Burr. Evaluation of mouse, rate-controlled isometric joystick, step keys, and text keys for text selection on a crt. *Ergonomics*, 21(8):601–613, 1978.

[CSM+16] Davide Calvaresi, Paolo Sernani, Mauro Marinoni, Andrea Claudi, Alessio Balsini, Aldo F. Dragoni, and Giorgio Buttazzo. A framework based on real-time os and multi-agents for intelligent autonomous robot competitions. In *2016 11th IEEE Symposium on Industrial Embedded Systems (SIES)*, pages 1–10, 2016.

[Dix09] Alan Dix. *Human-computer interaction.* Springer, 2009.

[DR99] Joseph S Dumas and Janice Redish. *A practical guide to usability testing.* Intellect Books, 1999.

[EFAA15] Ruffaldi Emanuele, Brizzi Filippo, Filippeschi Alessandro, and Carlo Alerto Avizzano. Co-located haptic interaction for virtual usg exploration. In *2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 1548–1551. IEEE, 2015.

[Fal10] Robert Faludi. *Building wireless sensor networks: with ZigBee, XBee, arduino, and processing.* O'Reilly Media, Inc., 2010.

[Fis01] Gerhard Fischer. User modeling in human–computer interaction. *User modeling and user-adapted interaction*, 11(1-2):65–86, 2001.

[FP11] Valentino Frati and Domenico Prattichizzo. Using kinect for hand tracking and rendering in wearable haptics. In *World Haptics Conference (WHC), 2011 IEEE*, pages 317–321. IEEE, 2011.

[FSLS+16] Ramon A Suỳrez Fernỳndez, Jose Luis Sanchez-Lopez, Carlos Sampedro, Hriday Bavle, Martin Molina, and Pascual Campoy. Natural user interfaces for human-drone multi-modal interaction. In *Unmanned Aircraft Systems (ICUAS), 2016 International Conference on*, pages 1013–1022. IEEE, 2016.

[Gra81] J Martin Graetz. The origin of spacewar. *Creative Computing*, 18, 1981.

[Ham04] Paul Hamill. *Unit Test Frameworks: Tools for High-Quality Software Development.* O'Reilly Media, Inc., 2004.

[Har] Michael González Harbour. Programming real-time systems with c/c++ and posix.

[JMWU91] Robin Jeffries, James R Miller, Cathleen Wharton, and Kathy Uyeda. User interface evaluation in the real world: a comparison of four techniques. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 119–124. ACM, 1991.

[KR07] James F Kurose and Keith W Ross. *Computer networking: a top-down approach.* Addison Wesley, 2007.

[Leaa] LeapMotion. Leap Motion documentation. `https://developer.leapmotion.com/documentation/cpp/index.html`.

[Leab] LeapMotion. Leap Motion. `http://www.leapmotion.com`.

[MBJS97] Mark R Mine, Frederick P Brooks Jr, and Carlo H Sequin. Moving objects in space: exploiting proprioception in virtual-environment interaction. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 19–26. ACM Press/Addison-Wesley Publishing Co., 1997.

[MDZ14] Giulio Marin, Fabio Dominio, and Pietro Zanuttigh. Hand gesture recognition with leap motion and kinect devices. In *2014 IEEE International Conference on Image Processing (ICIP)*, pages 1565–1569. IEEE, 2014.

[Mid] Midi Association. Midi Association - The official midi specification. `http://www.midi.org/specifications`.

[Min02] David A Mindell. *Between human and machine: feedback, control, and computing before cybernetics.* JHU Press, 2002.

[Nie94] Jakob Nielsen. Usability inspection methods. In *Conference companion on Human factors in computing systems*, pages 413–414. ACM, 1994.

[Nuw13] Rachel Nuwer. Armband adds a twitch to gesture control. *New Scientist*, 217(2906):21, 2013.

[PBAR15] Lorenzo Peppoloni, Filippo Brizzi, Carlo Alberto Avizzano, and Emanuele Ruffaldi. Immersive ros-integrated framework for robot teleoperation. In *3D User Interfaces (3DUI), 2015 IEEE Symposium on*, pages 177–178. IEEE, 2015.

[Rid04] Marty Ridgeway. Using code coverage tools in the linux kernel. 2004.

[Suc87] Lucy A Suchman. *Plans and situated actions: The problem of human-machine communication.* Cambridge university press, 1987.

[VMC16] Liberios Vokorokos, Juraj Mihal'ov, and Eva Chovancová. Motion sensors: Gesticulation efficiency across multiple platforms. In *Intelligent Engineering Systems (INES), 2016 IEEE 20th Jubilee International Conference on*, pages 293–298. IEEE, 2016.

[WBRF13] Frank Weichert, Daniel Bachmann, Bartholomäus Rudak, and Denis Fisseler. Analysis of the accuracy and robustness of the leap motion controller. *Sensors*, 13(5):6380–6393, 2013.

[WG95] Matthias M Wloka and Eliot Greenfield. The virtual tricorder: a uniform interface for virtual reality. In *Proceedings of the 8th annual ACM symposium on User interface and software technology*, pages 39–40. ACM, 1995.

[Wol88] William Wolf. German guided missiles henschel hs 293 and ruhrstahl sd1400 x fritz x, 1988.

[Zha12] Zhengyou Zhang. Microsoft kinect sensor and its effect. *IEEE multimedia*, 19(2):4–10, 2012.