Desarrollo de una interfaz para el prototipado y validación de un robot móvil autónomo de uso hospitalario

Amparo Tirado-Bou, Raúl Marín-Prades, Laura Baiguera Tambutti, Pedro J. Sanz, José V. Martí
Universitat Jaume I de Castellón
atirado, rmarin, al42081, sanzp, vmarti at uji.es

Resumen

El objetivo final del proyecto es la creación de un robot autónomo de soporte para el personal sanitario, en tareas relacionadas con la asistencia a pacientes infecciosos en aislamiento, como por ejemplo pacientes de COVID-19. La Metodología utilizada se basa en el Diseño de Prototipos, actualmente nos encontramos en la fase de diseño de la interfaz y desarrollo del prototipo experimental, que nos permite por una parte ir evaluando las características necesarias que ha de cumplir nuestra interfaz, así como evaluar a un bajo coste los componentes que formarán parte del producto final.

En el presente artículo presentamos nuestro proceso de desarrollo.

Palabras clave: COVID-19, Robótica Sanitaria, Desinfección, Sensores, Prototipo, Python, GUI

1 Introducción

En el contexto del proyecto H2020-ElPeacetolero se están desarrollando prototipos de robots manipuladores móviles para su uso en escenarios peligrosos, incluyendo el sistema de control remoto, la interfaz de usuario, y el ordenador embebido que permite ejecutar comportamientos autónomos.

El personal sanitario de la Facultad de Ciencias de la Salud de la Universidad Jaume I manifestó la necesidad de tener una adaptación del robot al ámbito hospitalario, considerando que el usuario del mismo no será un experto en ingeniería, sino una persona conocedora del entorno (e.g. enfermera, celador o médico).

Precisamente, el objetivo de este trabajo es establecer las bases para conseguir un prototipo que sea validado en un escenario hospitalario, y usado de forma simple y eficiente por su propio personal.

El desarrollo de un producto final es un proceso complejo y costoso que ha de ser planificado y ejecutado cuidadosamente.

Para este proyecto hemos decidido adaptar la metodología de prototipado para adaptarla a nuestros recursos y disponibilidad de los mismos.

La Universitat Jaume I nos permite disponer de los recursos necesarios para nuestra labor.

El prototipado nos permite materializar una idea a un bajo coste, obteniendo una retroalimentación en poco tiempo, lo que permite una evolución y mejora rápida del prototipo, se basa en la mejora continua, como se indica en "An introduction to rapid system prototyping" [6], Hardware y Software siempre han de ir de la mano en evolución constante.

El flujo de trabajo que hemos implementado lo podemos ver en la Figura 1, es un proceso que se retroalimenta constantemente.

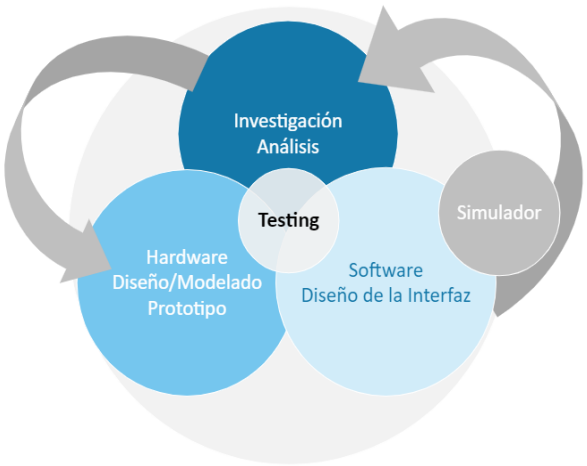

Figura 1: Diagrama Prototipado Rápido

El trabajo comenzó con dos equipos, uno dedicado al diseño y creación de un dispositivo de pequeñas dimensiones y otro equipo centrado en el desarrollo de la interfaz.

Los dos equipos comenzaron su desarrollo en paralelo.

https://doi.org/10.17979/spudc.9788497498418.0156

2 Sistema de prototipado rápido (RSP)

La técnica de prototipado rápido es una metodología de desarrollo que nos permite desarrollar modelos a menor escala que el producto final, estos modelos pueden ser manipulados y modificados de una forma ágil y a bajo coste, lo que nos permite una rápida evolución en el proceso de creación.

En el trabajo desarrollado por, Witte and Tichy [10] nos presentan un editor con vista previa gráfica, tal y como utilizan en Aldebaran Robotics con Choregraphe [8], Witte and Tichy basan su editor en ROS GUI framework, el cual contiene un conjunto de primitivas para definir las misiones del robot, por medio de un script se definen las instrucciones necesarias que pueden verse de forma visual y ejecutadas contra un simulador o robot real.

En el trabajo de Aldegheri and Bombieri[1] se propone una aproximación que permite el prototipado rápido a través de Matlab/Simulink,utilizando una librería de bloques que permite una simulación rápida.

En estos ejemplos vemos que es importante abstraer y encapsular en bloques los componentes necesarios, de forma que se puede colaborar en paralelo a través de diferentes grupos de desarrollo.

No solo el Software se beneficia del prototipado rápido, en el 2017 se presentó la NMMI [5] [1], plataforma modular con herramientas para el prototipado rápido principalmente de robótica suave, basada en una forma más fácil y efectiva de interactuar con el mundo real [7] utilizada principalmente en entornos dinámicos y no estructurados, ofrece bloques de componentes que permiten construir de forma sencilla y dinámica diversos robots funcionales.

3 Desarrollo de la Interfaz

En estos momentos el desarrollo del proyecto se encuentra en una fase de investigación y análisis, en la cual se está testeando y validando diferentes componentes.

Como paso previo se realizó un estudio y análisis de los requerimientos mínimos necesarios para desarrollar una interfaz, poder testear su viabilidad y eficiencia. Antes de comenzar el desarrollo, se realizaron varias reuniones para definir los pasos a realizar y el objetivo final. En la siguiente figura 2, se observa un borrador inicial, que representa los componentes que formarían parte de la interfaz.

La interfaz final ha de dar soporte a un robot autónomo que aún no está disponible, por este motivo se decidió utilizar un servidor de pruebas realizado en Unity basado en una arquitectura Cliente/Servidor que permite comunicación por http, de esta forma se pudo comenzar en paralelo con el equipo de Hardware.

Partiendo de este servidor se desarrollaron los primeros modelos de la interfaz, un cliente que pueda consumir sus servicios y enviarle comandos para el posterior control del robot autónomo.

En el punto inicial del desarrollo, se consideró que la interfaz debía estar formada por los siguientes módulos:

- Módulo de Navegación, que permitirá manejar el dispositivo de forma remota, en aquellas situaciones en las que el facultativo está en contacto con el paciente y requiera interactuar con él, para realizarle alguna consulta o para obtener datos por medio de los sensores que tendrá incorporado el dispositivo.

- Módulo Multimedia encargado de la comunicación, en el producto final deberá permitir realizar videollamadas, tanto con el personal sanitario del centro médico, como llamadas personales que permitan estar en contacto al paciente con su familia, y evitar en la medida de lo posible la sensación de soledad de los pacientes que se encuentran en aislamiento.

- Módulo de Trayectoria, debe permitir reconocer el entorno y programar trayectorias que serán realizadas de forma autónoma por el dispositivo, desde programar trayectorias para la desinfección de áreas del hospital, como hacer un seguimiento de pacientes por medio del uso de sensores que permitan la toma de muestras en remoto, así como la evolución de la temperatura del paciente.

- Módulo Control del dispositivo, representación en 3D para controlar los elementos del dispositivo remotamente, la interfaz dispondrá de una representación 3D del dispositivo que permitirá seleccionar sus elementos e interactuar con ellos. Por ejemplo se dispondrá de un brazo robótico, con capacidad para adaptarle diferentes sensores por ejemplo de temperatura, al seleccionarlo podremos aproximarnos al paciente y poder tomarle la temperatura.

[1] http://www.naturalmachinemotioninitiative.com

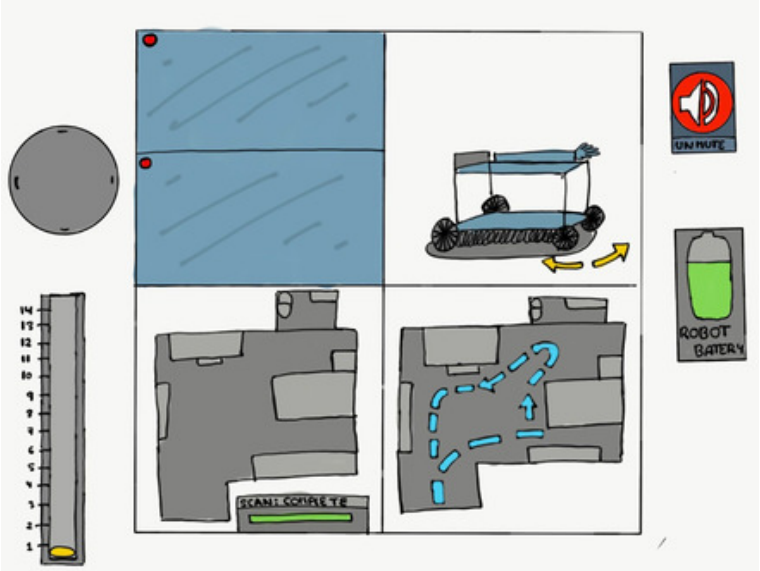

Figura 2: Borrador interfaz

Para comenzar el desarrollo de la interfaz se tomó la decisión de realizarla en Python, al ser un lenguaje ágil y sencillo de implementar lo que lo hace muy conveniente para el prototipado rápido. Existen investigaciones previas que han considerado Python como el lenguaje idóneo para el desarrollo rápido de interfaces, como podemos ver en [4] y [3]. Dentro de los paquetes disponibles en Python para el desarrollo de Interfaces gráficas podemos encontrar Tkinter un ejemplo de su utilización en [11] y [9] .

Se comenzó a trabajar con una versión inicial realizada con Tkinter, con la que se pudo validar la comunicación con el Servidor Unity, el proyecto final debe permitir lanzar la interfaz desde diferentes dispositivos (IOS, Android, Windows, Linux,) por ello necesitamos desarrollar una interfaz multiplataforma, para este fin se comenzó la revisión de las opciones que proporciona Python al respecto, aquí se muestran dos opciones:

- **Kivy**, funciona sobre Linux, Windows, OS X, Android, IOS, y Raspberry Pi, funciona completamente en Python, sencillo de implementar y dinámico para ser utilizado en el prototipado rápido, basado en OpenGL, incluyen las funcionalidades táctiles que son esenciales para las aplicaciones móviles.
- **PyQt**, desarrollado a partir de la biblioteca gráfica Qt, mas potente y complejo que Kivy, aunque no está desarrollado completamente en Python y tiene una curva de aprendizaje mucho mayor que Kivy.

Tras revisar las dos opciones nos decidimos por la librería Kivy , por su mayor facilidad y menor curva de aprendizaje

3.1 Prototipo Testing

La tarea inicial fue la de validar la comunicación con el servidor proporcionado por nuestro simulador en Unity. Para ello se realizó una interfaz sencilla, podemos ver un ejemplo en el la Figura 3 que permitía validar el módulo de navegación y trayectoria.

Los elementos gráficos de la interfaz se definen en un fichero .kv, por ejemplo definimos los botones(por medio del tag "Button"), componentes gráficos como textos (con el tag "text") todo ellos dentro de la distribución seleccionada por medio de componentes "Layout", por ejemplo "GridLayout").

Aquí podemos ver un fragmento

```
<TargetScreen>:
    GridLayout:
        cols:1
        rows:3
        padding: 5
        spacing: 5
        GridLayout:
            cols: 3
            rows: 1
            padding: 5
            spacing: 5
            size_hint_y: 0.15
            Button:
                text: 'Target'
                id: navigationButton
                on_press:
                root.goto('TargetScreen')
                size_hint_y: 0.1
            Button:
                text: 'Navegacion'
                on_press:
                root.goto('NavegationScreen')
                size_hint_y: 0.1
            Button:
                text: 'Settings'
                on_press:
                root.manager.current ='SettingsScreen'
```


https://doi.org/10.17979/spudc.9788497498418.0156

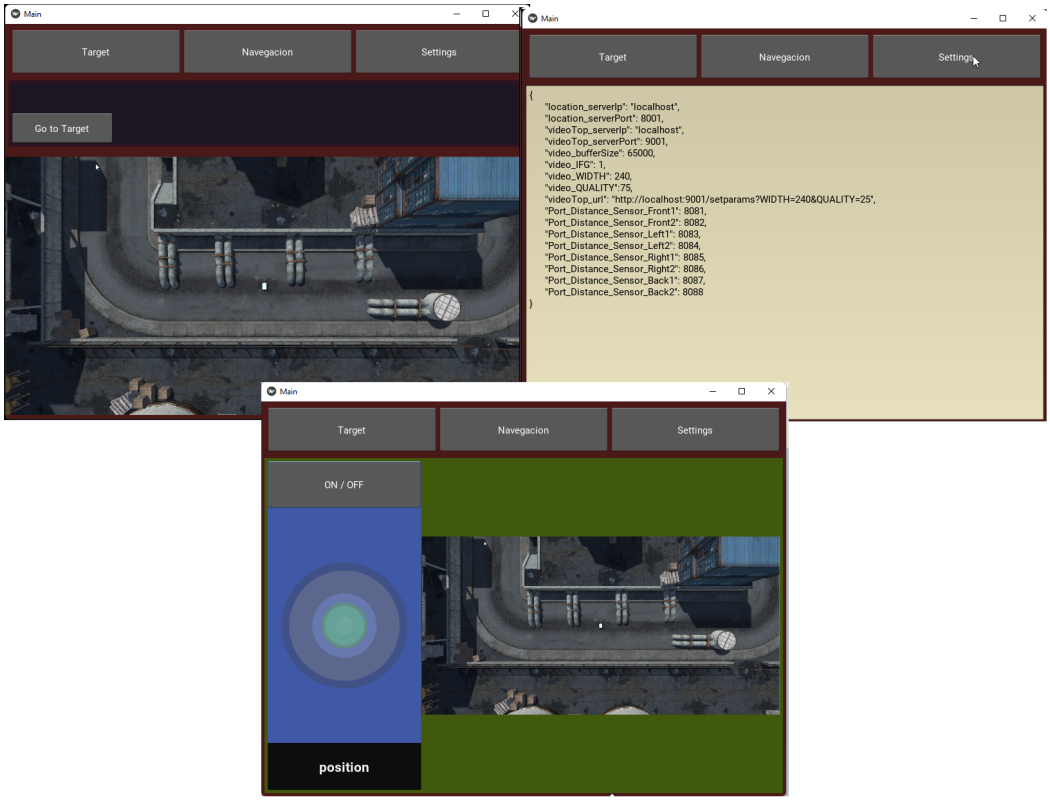

Figura 3: Interfaz V1

```
size_hint_y: 0.1
```

Listing 1: frontend.kv

La interfaz se comunica con el servidor por medio de diversos módulos que deberán adaptarse posteriormente al prototipo, de forma que se pueda abstraer la capa de presentación, por ejemplo se puede ver el módulo de navegación, donde se definen las órdenes que se pueden enviar al simulador. En el siguiente ejemplo vemos como comunicar al simulador la orden de desplazarse hacia delante.

```
navigation.forward(self)

def forward(self):
    openUrlNodata(self.serverIP,
                  self.serverPort, "/forward/50")

def openUrlNodata(serverIP, serverPort,urlNoHTTP):
    s = socket.socket(socket.AF_INET,
                      socket.SOCK_STREAM)
    s.connect((serverIP, serverPort))
    cad = "GET " + urlNoHTTP
    + " HTTP/1.0\r\nHost:"
    + serverIP + "\r\n\r\n"
    s.sendall(cad.encode())
    s.close()
    return
```

Listing 2: Fragmento

Paralelamente al testeo de la comunicación con el simulador, se comienza el proceso de mejora de los componentes visuales de la interfaz, así como su usabilidad. De forma que se evoluciona a una versión mejorada del simulador, un el ejemplo del mismo lo podemos ver en la siguiente Figura 4.

Como indicamos en le diagrama que se muestra en la Figura 1, el desarrollo software va paralelamente ligado al desarrollo Hardware, por tanto a medida que se va evolucionado en el desarrollo del dispositivo, también lo va haciendo la interfaz. Al disponer del dispositivo real, se comenzó la fase de creación del servidor explicado más extensamente en el punto 4 del presente artículo.

Con el nuevo servidor funcionando en el prototipo, se validó el cliente y se consideró que necesitábamos un cliente mas ágil y potente que permitiera la ejecución del mismo en diferentes dispositivos, tanto PC / IOS / Android /Linux, atendiendo a las capacidades técnicas del equipo las nueva versión del cliente se comenzó a desarrollar en HTML5, CSS, Jquery, Javascript, lo que permitió poder lanzar las pruebas desde el prototipo de una forma más ágil.

4 Desarrollo del Servidor

El equipo hardware presentó el primer prototipo del robot autónomo, en este punto se comenzó a preparar un Servidor para validar las comunicaciones , así como las instrucciones de control del mismo.

https://doi.org/10.17979/spudc.9788497498418.0156

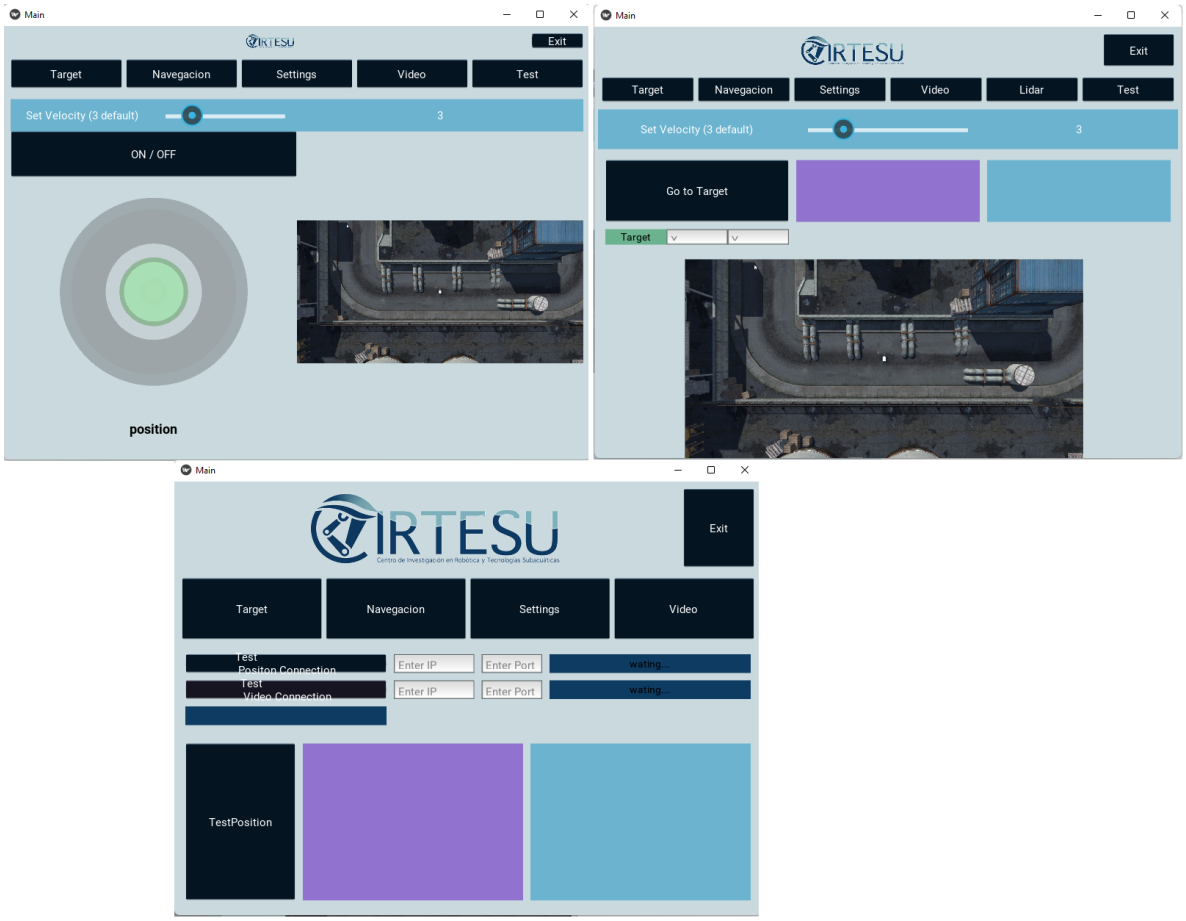

Figura 4: Interfaz V2

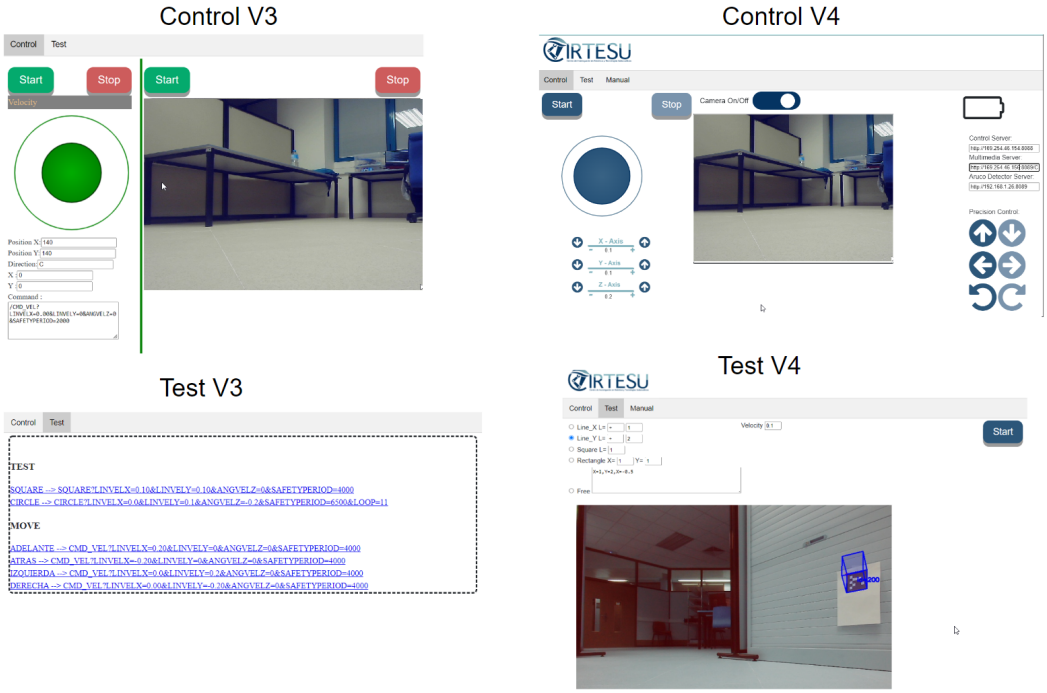

Figura 5: Interfaz V3 y V4

https://doi.org/10.17979/spudc.9788497498418.0156

Se partía de una placa Raspberry Pi, encargada de lanzar un servidor Python, que recibiría las peticiones del cliente y sería el encargado de comunicarse con el prototipo, en esta primera versión se disponía de una cámara USB y una tarjeta OpenCr para el control de los motores del dispositivo.

4.1 Prototipo servidor V1 (Cliente/Servidor Python)

El módulo de Python utilizado para desarrollar el servidor es http.server [2] , que sigue una arquitectura de cliente / servidor, de la cual utilizamos los siguientes componentes:

- HTTPServer : encargado de crear la conexion TCP, indicando la IP y el numero del puerto de conexión.
- BaseHTTPRequestHandler : encargado de recuperar las peticiones que le llegan al servidor dispositivo remotamente.

En esta primera fase, el servidor retorna una interfaz básica solo de texto donde se muestran las instrucciones que tienes implementadas el servidor, por ejemplo:

```
STOP
FORWARD
BACKWARD
MOVE LEFT (NO turn)
MOVE RIGHT (NO turn)
TURN LEFT
TURN RIGHT

CHANGE VELOCITY OF CURRENT MOVEMENT
[1][2] [3] [4] [5] [6] [7] [8] [9][10]
GET POSITION
```

Listing 3: Interfaz python- mainIndex.py

Los anteriores parámetros generan las siguientes peticiones al servidor

```
'/stop'
'/forward'
'/backward'
'/right'
'/left'
'/turnright'
'/turnright'
'/turnleft'
'/turnleft'
```

Listing 4: Servidor python- server.py

En la siguiente Figura 6 podemos ver un ejemplo de la navegación de una orden desde el evento del cliente hasta llegar al prototipo

Con esta versión se dió soporte al equipo de desarrollo Hardware para validar de forma sencilla los componentes mecánicos del prototipo y comenzar las revisión de la Odometría del mismo.

[2] https://docs.python.org/3/library/http.server.html

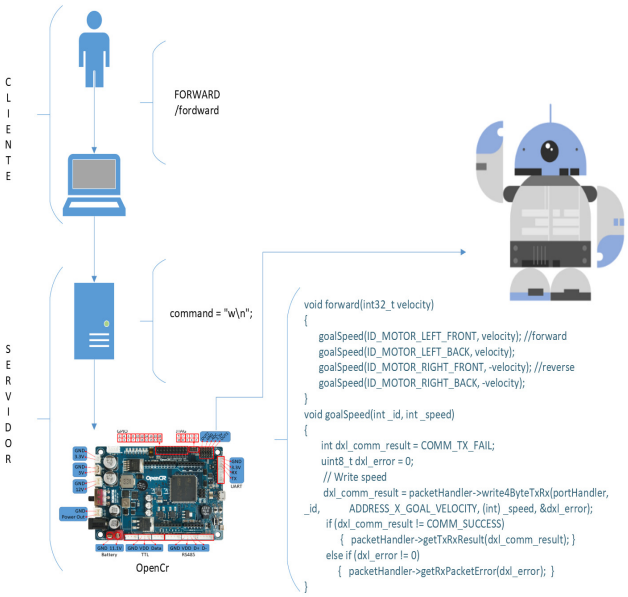

Figura 6: Diagrama flujo Orden

4.2 Prototipo servidor V2 (Cliente: Javascript / Servidor: Python)

La evolución del prototipo del servidor coincide con la Versión V3 de la interfaz, que podemos ver en la siguiente Figura 5.

En esta fase del desarrollo del proyecto, se mantiene un único servidor http que retorna todos los elementos web que necesita la interfaz, contenido (html,css,png,js) y permite comandos mas complejos para interactuar con el prototipo, las instrucciones se basan en el movimiento por velocidad / tiempo, hemos de indicar la velocidad con la que queremos movernos en cada eje (m/s), así como el tiempo que ha durar esa velocidad (milisegundos).

Aquí se muestran a modo de ejemplo, dos comandos que puede interpretar el Servidor:

- comando básico en el cual informamos la velocidad en cada uno de los ejes, X, Y, Z, así como un parámetro de seguridad indicando el tiempo que se han de mantener las velocidades, de forma que podamos movernos en todos los ejes ya sea de forma positiva como negativa. /CMD _ VEL?LINVELX=0.20&LINVELY=0&ANGVELZ=0 &SAFETYPERIOD=2000,
- comandos para validar la odometría del dispositivo /SQUARE?LINVELX=0.10&LINVELY=0.10&ANGVELZ=0 &SAFETYPERIOD=4000 /CIRCLE?LINVELX=0.0&LINVELY=0.1&ANGVELZ=-0.20 &SAFETYPERIOD=6500&LOOP=11

Este prototipo en el lado cliente muestra un componente dinámico a modo joystick que permite decidir las coordenadas del movimiento que transmite el usuario, estas coordenadas se transforman en las siguientes peticiones al servidor:

```
['C', '/CMD_VEL?LINVELX=0.00&LINVELY=0&ANGVELZ=0
&SAFETYPERIOD=2000'],
['N', '/CMD_VEL?LINVELX=0.20&LINVELY=0&ANGVELZ=0
&SAFETYPERIOD=2000'],
['S', '/CMD_VEL?LINVELX=-0.20&LINVELY=0&ANGVELZ=0
&SAFETYPERIOD=2000'],
['E', '/CMD_VEL?LINVELX=0.00&LINVELY=-0.20&ANGVELZ=
&SAFETYPERIOD=2000'],
['W', '/CMD_VEL?LINVELX=0.00&LINVELY=0.2&ANGVELZ=0
&SAFETYPERIOD=2000'],
['NE', '/CIRCLE?LINVELX=0.0&LINVELY=0.1&ANGVELZ=-0.
&SAFETYPERIOD=6500&LOOP=2'],
['SE', '/CIRCLE?LINVELX=0.0&LINVELY=0.1&ANGVELZ=0.2
&SAFETYPERIOD=6500&LOOP=2'],
['NW', '/CIRCLE?LINVELX=0.0&LINVELY=0.1&ANGVELZ=-0.
&SAFETYPERIOD=6500&LOOP=2'],
['SW', '/CIRCLE?LINVELX=0.0&LINVELY=0.1&ANGVELZ=-0.20
&SAFETYPERIOD=6500&LOOP=2'],
```

Listing 5: Cliente Javascript- index.html

En los comandos anteriores se pueden ver que los valores de velocidad y tiempo son constantes y su función es la de poder validar los elementos mecánicos del prototipo.

En esta fase del desarrollo del proyecto el equipo Hardware incorpora una cámara USB al sistema, lo que permite al servidor ampliar sus funciones y poder servir la imágenes emitidas por la cámara, a través de la librería OpenCV de Python.

4.3 Prototipo servidor V3 (cliente: Javascript / Servidor Multimedia + Servidor Control)

Por parte del equipo hardware se valora la posibilidad de utilizar una cámara mas potente como es la Intel® RealSense D453i[3], para poder generar mapas de profundidad y utilizar la IMU que incorpora, al intentar instalar y compilar el software necesario para hacer funcionar la RealSense en la placa Raspberry Pi, se vió que el proceso se quedaba bloqueado y no finalizaba la instalación, quedamos a la espera del soporte por parte del equipo de Intel para poder resolver el problema, para no paralizar el desarrollo del proyecto se planteó el uso de un nuevo procesador en este caso un Intel NUC[4].

Con estás últimas mejoras hardware y planificando módulos futuros, se toma la decisión de realizar un servidor dedicado para el tratamiento de las imágenes, de forma que la arquitectura actual queda como podemos ver en la siguiente imagen 7:

[3] https://www.intelrealsense.com/depth-camera-d435i/

[4] https://www.intel.la/content/www/xl/es/products/details/nuc.html

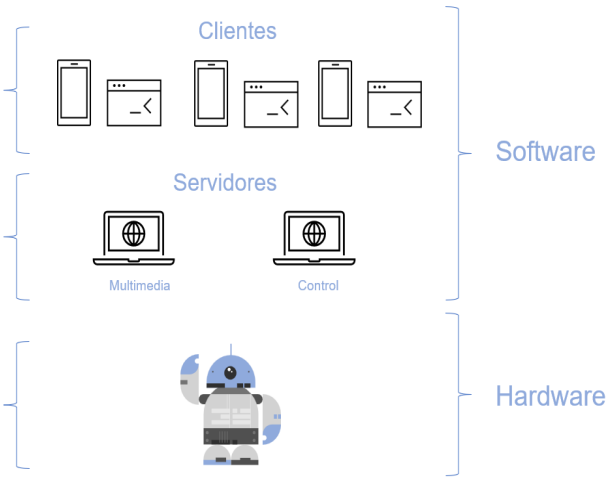

Figura 7: Esquema del sistema actual

5 Conclusiones y Trabajo Futuro

5.1 Mecatrónica y software modular.

Durante este último año se ha validado la metodología de desarrollo seleccionada (RSP)2, que ha permitido avanzar en paralelo tanto en el desarrollo Software (tanto cliente como servidor) y desarrollo Hardware.

Con el prototipado rápido se ha podido comparar diferentes CPU's, Raspberry PI[5], NVIDIA Jetson Nano[6], o Intel NUC[7]. al igual que el uso y validación de diferentes dispositivos de visión como la utilización de una cámara HD USB [8] o la Intel® RealSense D453i[9]

Se ha podido comprobar que la utilización del protocolo HTTP y el lenguaje Python para la validación del prototipo, ha permitido avanzar en todas las áreas involucradas, permitiendo la adaptación del sistema a la intervención concreta a realizar. Por todo ello se ha tomado la decisión de crear diversos servidores dedicados, uno para el control del dispositivo y otro servidor que será el encargado del módulo multimedia, ganando en eficiencia.

5.2 Uso docente

Está metodología de trabajo ha permitido avanzar en otros aspectos como el docente, permitiendo que los alumnos se involucraran en el proyecto, haciendo uso de los prototipos generados y

[5] https://www.raspberrypi.com/products/raspberry-pi-4-model-b/

[6] https://developer.nvidia.com/embedded/jetson-nano-developer-kit

[7] https://www.intel.la/content/www/xl/es/products/details/nuc.html

[8] https://bluerobotics.com/store/sensors-sonars-cameras/cameras/cam-usb-low-light-r1/

[9] https://www.intelrealsense.com/depth-camera-d435i/

https://doi.org/10.17979/spudc.9788497498418.0156

aportando su visión e ideas al proyecto.

5.3 Trabajo Futuro

El próximo paso de mejora software será la utilización de técnicas de aceleración software y hardware.

Se estudiará el desarrollo del Servicio en C++ para optimizar su rendimiento, así como el uso de Unity como interfaz de desarrollo óptima para su uso en diversos dispositivos.

Para la mejora y análisis actual de la interfaz, se está preparando un encuesta dirigida al personal sanitario en activo, así como a los futuros profesional que actualmente están en formación, tras la revisión de los resultados se adaptaran sus propuestas, que serán validadas con el prototipo.

El producto final debe tener un mayor alcance y control de seguridad ya que deberá funcionar en un ambiente hospitalario y tener acceso a información sensible de los pacientes, para ello se creará un módulo de seguridad encargado de los accesos al sistema y la gestión de los diferentes perfiles.

En el apartado de hardware, se incorporará un sensor de temperatura PIR, tipo D6T de Omron [10] y un sensor de pulso y nivel de oxígeno en sangre tipo MAX30100 de Maxim [11], integrándolos en el sistema como dispositivos periféricos conectados mediante un bus I2C. La adición de estos dispositivos dotará al robot de la capacidad de realizar mediciones de algunas constantes vitales del paciente (temperatura corporal, ritmo cardíaco y nivel de oxígeno en sangre) aproximando los sensores a éste.

El dispositivo final formará parte de un sistema multirobot, que actualmente está siendo desarrollado, en el siguiente artículo [2] podemos ver la explicación completa del sistema [2].

Cuya arquitectura podemos ver en el siguiente diagrama 8:

Agradecimientos

Este trabajo ha sido realizado por el equipo del CIRTESU, "Centro de Investigación en Robótica y Tecnologías Subacuáticas" de la Universidad Jaume I, grupo IRS-Lab (Interactive and Robotic systems Lab), y financiado por los proyectos PID2020-115332RB-C31 (COOPERAMOS), IDIFEDER/2018/013 (GV), UJI-B2021-30(AUDAZ), y el proyecto H2020-Peacetolero-NFRP-2019-2020-04.

[10] https://components.omron.com/us-en/datasheet_pdf/A274-E1.pdf

[11] https://www.maximintegrated.com/en/products/sensors/MAX30100.html

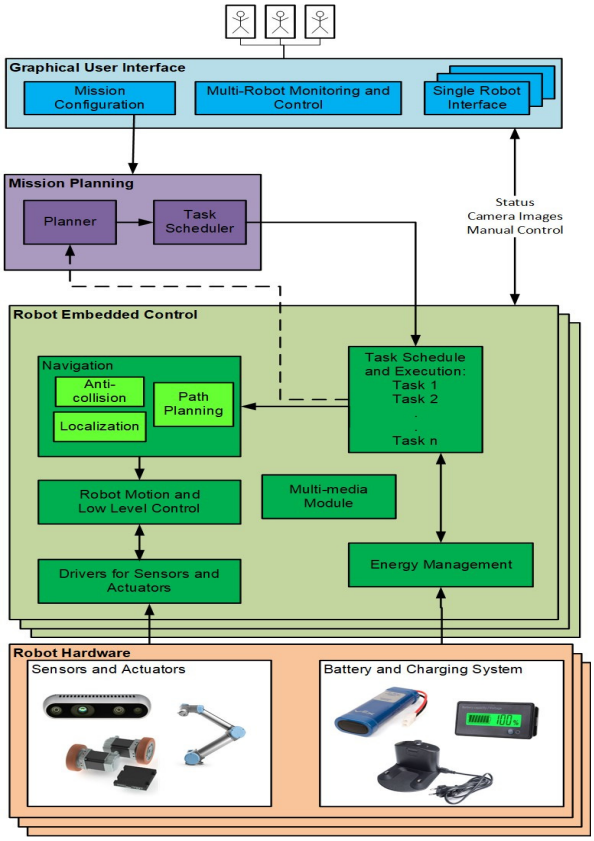

Figura 8: Sistema de control Multirobot

English summary

Development of an interface for the prototyping and testing of a autonomous mobile robot

Abstract

The final objective of the project is the creation of an autonomous support robot for health personnel, in tasks related to the care of infectious patients in isolation , such as COVID-19 patients. The Methodology used is based on the Design of Prototypes, we are currently in the design phase of the interface and development of the experimental prototype, which allows us, on the one hand, to evaluate the necessary characteristics that our interface must meet, as well as to evaluate a low cost the components that will form part of the final product.

In this article we present our development process.

Keywords: COVID-19, Sanitary Robotics, Sanitation, Disinfection, Sensors, Python, GUI.

Referencias

[1] Stefano Aldegheri and Nicola Bombieri. Rapid prototyping of embedded vision systems: Embedding computer vision applications into low-power heterogeneous architectures. In *2018 International Symposium on Rapid System Prototyping (RSP)*, pages 63–69, 2018.

[2] Laura Baiguera. State of the art in control systems for a cooperative distributed mobile robot fleet in a healthcare environment. In *Submited to the XLIII JORNADAS DE AUTOMÁTICA : LIBRO DE ACTAS*, pages –. 2022.

[3] Tarek Belabed, Alexandre Quenon, Vitor Ramos Gomes Da Silva, Carlos A. Valderrama Sakuyama, and Chokri Souani. Full python interface control: Auto generation and adaptation of deep neural networks for edge computing and iot applications fpga-based acceleration. In *2021 International Conference on INnovations in Intelligent SysTems and Applications (INISTA)*, pages 1–6, 2021.

[4] Jonathan S. Brumberg, Sean D. Lorenz, Byron V. Galbraith, and Frank H. Guenther. The unlock project: A python-based framework for practical brain-computer interface communication "app" development. In *2012 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pages 2505–2508, 2012.

[5] Cosimo Della Santina, Cristina Piazza, Gian Maria Gasparri, Manuel Bonilla, Manuel Giuseppe Catalano, Giorgio Grioli, Manolo Garabini, and Antonio Bicchi. The quest for natural machine motion: An open platform to fast-prototyping articulated soft robots. *IEEE Robotics Automation Magazine*, 24(1):48–56, 2017.

[6] F. Kordon and Luqi. An introduction to rapid system prototyping. *IEEE Transactions on Software Engineering*, 28(9):817–821, 2002.

[7] José Medina H and Paulina Vélez N. "soft robotic": Una nueva generación de robots. *Maskana*, 5:109–118, ene. 2016.

[8] E. Pot, J. Monceaux, R. Gelin, and B. Maisonnier. Choregraphe: a graphical tool for humanoid robot programming. In *RO-MAN 2009 - The 18th IEEE International Symposium on Robot and Human Interactive Communication*, pages 46–51, 2009.

[9] Jaskaran Singh, Chandan Taluja, Vijaya Choudhary, Kailash Chand, and Paramita Guha. Design and development of graphical user interfaces for detection of forest fires at early stages using open-source technologies. In *2021 6th International Conference on Signal Processing, Computing and Control (ISPCC)*, pages 218–222, 2021.

[10] Thomas Witte and Matthias Tichy. A hybrid editor for fast robot mission prototyping. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering Workshop (ASEW)*, pages 41–44, 2019.

[11] S.N. Wuth, R. Coetzee, and S.P. Levitt. Creating a python gui for a c++ image processing library. In *2004 IEEE Africon. 7th Africon Conference in Africa (IEEE Cat. No.04CH37590)*, volume 2, pages 1203–1206 Vol.2, 2004.

 https://doi.org/10.17979/spudc.9788497498418.0156