VIPO: A Spatial and Visual Programming Language for Mobile Robots and IoT Nodes

Leave Authors Anonymous
for Submission
City, Country
e-mail address

Leave Authors Anonymous
for Submission
City, Country
e-mail address

Leave Authors Anonymous
for Submission
City, Country
e-mail address

ABSTRACT
Programming and integration of robotic platforms and new technologies such as IoT Nodes are complicated to understand, develop, and to execute from a perspective of novice users. Robot programming requires deep understanding of several technical disciplines for development of software for planning autonomous mobile robot work flows. In this paper, we introduce *VIPO*, a web-based visual and spatial programming language that allows novice users and small industries to program mobile robots and IoT Nodes to execute planned tasks (locomotion and manipulation) in smart-industrial environments. In addition to the visual programming language, we also present a system that dynamically register architecture that deploys VIPO for programming a ROS-based mobile robot and Arduino-based IoT Nodes for a simple industrial use case. Future evaluations will demonstrate the understanding, time reduction, capabilities that the programming language and interface provides when programming robotic and integrated systems. We discuss the advantages and implications of VIPO in future applications in smart environments.

ACM Classification Keywords
H.5.m. Information Interfaces and Presentation (e.g. HCI): Miscellaneous; See http://acm.org/about/class/1998/ for the full list of ACM classifiers. This section is required.

Author Keywords
Authors' choice; of terms; separated; by semicolons; include commas, within terms only; required.

INTRODUCTION
Robotic platforms, IoT, and automated tasks require a high level of knowledge and programming skills to develop applications and improvement them in real life. Additionally, the integration of robotic platforms with other machines and IoT devices increase even more the limitations. This level required constrains the speed of technology development and the time required for novice users to be able to adapt and use robotic platforms. This platforms present different techniques of programming. Currently, They can be programmed by using text in different coding languages such as C, C++, python, logic blocks, that require training and understanding of systems and syntax.

To ensure success in the robot and IoT programming, it is required a method that condense all the information and provides the user an interface where he/she can easily interact and create the desired programs. In this paper, we present VIPO, a web based collaborative programming interface that allow novice users and small industries to develop software to control dynamically task flows, robot controls, and robot-IoT integrated systems inside smart environments. VIPO needs access to the smart environment. To achieve this capability, we need to make a scan of the smart environment. The scan is performed using a robotic platform and the IoT devices needs to be equipped with Lidar and an Ultra Wide Band Module (UWB), that will provide with the exact locations of the IoT devices, inside the scanned environment [2]. With this information acquired, the system will recognize the IoT Nodes and the location of each one to later generate a 2D map. It is important to mention that the environment scan needs to be done only once for every new environment. Now, we execute VIPO in a web browser, open the generated map and recognize all the elements in it. Once VIPO establishes connection with the server and recognizes the robots and IoT devices in the map, it starts obtaining information from the devices and is ready to start programming. The user can drag and drop the logic icons on the left of the interface on the map, and by joining with arrows it can create the logic flow of the program. The user can select between IoT order, number of sequences, robots appropriate for each task, and control the status of the IoT devices in real time. The program created by the user is sent to a Master computer developed using ROS (Robotic Operative System). This contains the capabilities to manage the resources identified in the smart environment. This is the part that is incharge to make the user program to be executed. Different programs can be developed in different devices and even collaborate between users to create more complex programs. We focus in a Factory and work environments to demonstrate the dynamic capabilities of VIPO. It allows fast generation and modifications of programs that in long term could improve productivity.

We plan to evaluate our system using expert and novice users. this will provide us with the results needed to validate our approach and the feedback to improve the concept. VIPO targets

UIST'19, May 07–12, 2016, San Jose, CA, USA

© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ISBN 123-4567-24-567/08/06. . . $15.00

DOI: http://dx.doi.org/10.475/123_4

the easy of programming, the programming reduction of time, the reduction of the required knowledge needed for programming, collaborative programming, and the programming of integrated devices. The evaluation will have four main aspects to be covered: 1) Demonstration, evaluation of the system through a set of use cases. 2) Usage, a set of expert and novice users will be exposed to the web based programming interface to evaluate their development into the context. Also, we will have a controll group developing the same tasks in different robotic platforms to obtain a reference point of our efficiency. For 3) Performance, and 4) Heuristics the analysis and the metrics of the the data collected during the evaluation scenarios purposed above will present a qualitative and quantitative values of VIPO.

The contributions of this work are the following:

1) Creation of a web based visual programming language for control flow in a functional spacial domain for collaborative robotic, IoT, and tasks integration.

2) Dynamically registering and updating properties, methods, and locations of IoT devices and Robots using Resource Description Framework (RDF) files[4];

3) Creation of a scan-update-program workflow that bridges Human-Machine-Robot (HMR) interactions and allow novice users to manage robots, IoT devices, and complete systems in new environments;

4) Creation of a library that allow a program generated in a visual interface be executed in ROS, allowing the the control of systems in real and simulation environments;

5) A group of user studies on the system performance and the cognitive behaviours of participants that will guide future research in the area; and a set of demonstrative applications that will demonstrate the system capabilities.

RELATED WORK

Our work builds on prior work in visual programming, web programming interfaces, and ROS programming interfaces. This areas constantly try to provide users with enough tools to enable programming characteristics in an easy, and fast way. Programming tools that allow users to visualize robots, input parameters, controllers in real time [1], or others that are specific programming interfaces that manage to provide high level commands for robots, and calibration of parameters [8]. Works that provide skills, task and parameters programming and configurations, with multiple robots ans providing support to create simple loops and workflows [10]. In a similar approach to VIPO, some work use the scanned map to create an interface to visualize the task assignment and the work area where the commands and programs are executed or displayed [7, 6]. Others expand this map environment to virtual reality scenarios and provide landmarks and allow the users create the navigation commands [5]. Given the importance of the robot programming, Microsoft created a software for robot programming, with the goal of standardize the practice of programming coordination between robots and control [3]. To expand this interfaces, industry, researchers and new technologies provide interfaces and services into web based platforms.

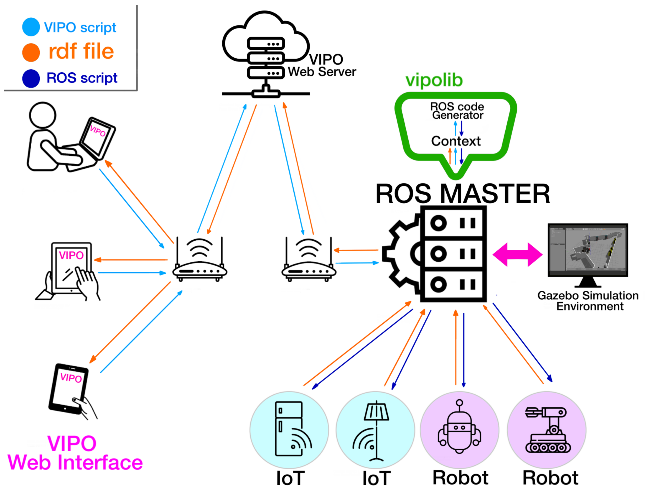


Figure 1. VIPO programming and control workflow

This allows to make the platforms universal, accessible, and multi-platform [6, 10, 5]

SYSTEM ARCHITECTURE

This section details the system architecture of the platform we developed for deployment of VIPO for programming Autonomus Mobile Robots(AMR) in IoT environments (Figure 1). The platform can be broken down into four major sub-systems (1) VIPO (UI and web-server) (2) IoT Nodes and (3). ROS Master (4) Simulation Engine. **Overview** VIPO sub-system consists of the web user interface and the web-server. The user performs visual programming on a browser-based user interface. Once the user completes the program, the user can compile the visual program he created to check for errors. Compilation happens at the back-end VIPO web-server and results in the generation of a high-level script (user comprehendible). Upon validation of the script, the user can either run a simulation of the program on the visualization platform or directly run the program on the AMR.

The AMR and Industrial Machinery make up the second subsystem namely the IoT Nodes. For the development of the system and demonstration of the use-case, ESP32 microcontroller units were used. Each IoT node wirelessly communicates to the ROS Master over WiFi using TCP. Real-time status (job-status, temperature etc.) and IoT specific information (ID, name, location etc.) are broadcasted to ROS-master in a modified Resource Description Format (RDF) [4]. It is possible to monitor and control each IoT node from the ROS Master.

ROS Master is essentially a ROS Server acting as a central node of communication and coordination between VIPO, IoT nodes, and the visualization platform. Being a server, ROS master has several sockets (web and serial) for operation. In addition to coordinating and communicating with other subsystems, ROS Master also functions as an interpreter that translates the script generated by VIPO (high-level user-readable) to machine codes to operate the IoT Nodes (AMR and Machinery).

After performing the visual program on the VIPO web interface, the user is given an option to simulate the program. This feature is made possible through the last sub-system which is the Simulation Engine that runs on Gazebo controlled by the ROS-Master. The simulation engine is designed to be used in two possible ways (1) to debug the program and (2) to remotely keep a check of the status of the robot and machinery as the program is run in physical space.

VIPO

VIPO is a web-based visual programming language for creating tasks for IoT Nodes and robots in a spatial domain. It contains three components: 1) a layout map of the smart environment where a user wants to program with, 2) the IoT Nodes and robots available in the environment, and 3) a spatial visual language for users to program tasks to control the IoT Nodes and robots. The first two components are defined by the users and dependent on their smart environment. The third component, however, is a general visual language that is applicable to different environments. The rest of section will discuss the workflow about how the first two components are obtained and how a user can program task with the spatial visual language.

Map Generation

Since VIPO is a platform to program tasks for IoT Nodes and robots, we assume a user has at least one robot to program with. The first step to apply VIPO in their environment is to let the robot scan the environment and generate a 2D layout map. Then the map information is sent to VIPO and rendered as the background, as shown in Figure 2a. Here, the map information also includes the dimensions of the room (i.e., width and height) and the resolution (i.e., meter per pixel) of the map. Figure 1 shows the communication flow: the robot first sends the map information to ROS Master, and eventually sends to VIPO. The communication between the robot and ROS Master will be described in later section, while the communication between ROS Master and VIPO is via web socket.

IoT Node Registration

Similar to the map generation, IoT Nodes keep sending their information (e.g., location, status, and supported operations) to ROS Master and further being pushed to VIPO. Thereafter, VIPO will automatically render the IoT Nodes at the corresponding locations on the 2D layout map, as shown in Figure 2b. Again, the communication detail between the IoT Nodes and ROS Master will be discussed in later section and thus ignored here.

Spatial Visual Language Description

After generating the map and registering the IoT Nodes, users can start programming by using a set of visual programming constructs. Users can click on the toolbar icon (Figure 2c) to draw each visual symbol directly on the layout map. The visual symbol and textual meaning of each construct is summarized in Figure 3.

As shown in Figure 3, there are two types of constructs: one is spatial (i.e., location-related), and the other is not. Spatial constructs include *move*, *pick*, and *drop*, which require users to specify the source and/or target. Non-spatial constructs include if, *while*, and *timer*.

Besides these primitive constructs, we also provide a higher-level abstraction, $function$. A user can create a $function$ by wrapping one or more primitive constructs or even another function as a cohesive chunk. This means that a user has no need to create it again and can even import $function$s defined by other users. It also means a commonly shared task can be reused to form a more complex task.

For example, suppose a user wants to program a robot to finish a simple task with three steps: 1) picks a box from IoT Node A, 2) moves to IoT Node B, and 3) drops the box at B. The user first needs to click on the Pick-and-Drop icon. Then he/she can click on the icon of A to specify the source of *pick*, and then click on the icon of B to specify the target of *drop*. Once source and target are specified, the user enters the number (e.g., 1) and object type (e.g., "box") to carry. Figure 2d shows the visual notations for this simple task. The system will automatically add a *move* between *pick* and *drop* since the locations are changed. Finally, if the user realizes that this task is frequent, he/she can define it as a $function$ and save it for reusing in the future. A more comprehensive example will be given in Use Cases section later.

By using those visual notations, users without programming experience can program tasks to control robots and IoT Nodes. Moreover, since the programs are shown directly and visually on the layout map, users can have a direct mapping between what is programmed and what will happen in the physical environment.

Task Script Preview and Execution

Once a task is created, users can preview the task in textual format if they want to double check the script before execution. So far the script is in Python language because its syntax is closer to a plain English in which a novice user can understand.

A compiler is used to convert each visual symbol into its corresponding textual format. The compiler will warn users if the syntax is not correct, for example, if they forgot to enter values in the Pick-and-Drop construct.

For example, the task in Figure 2d will be compiled into following code:

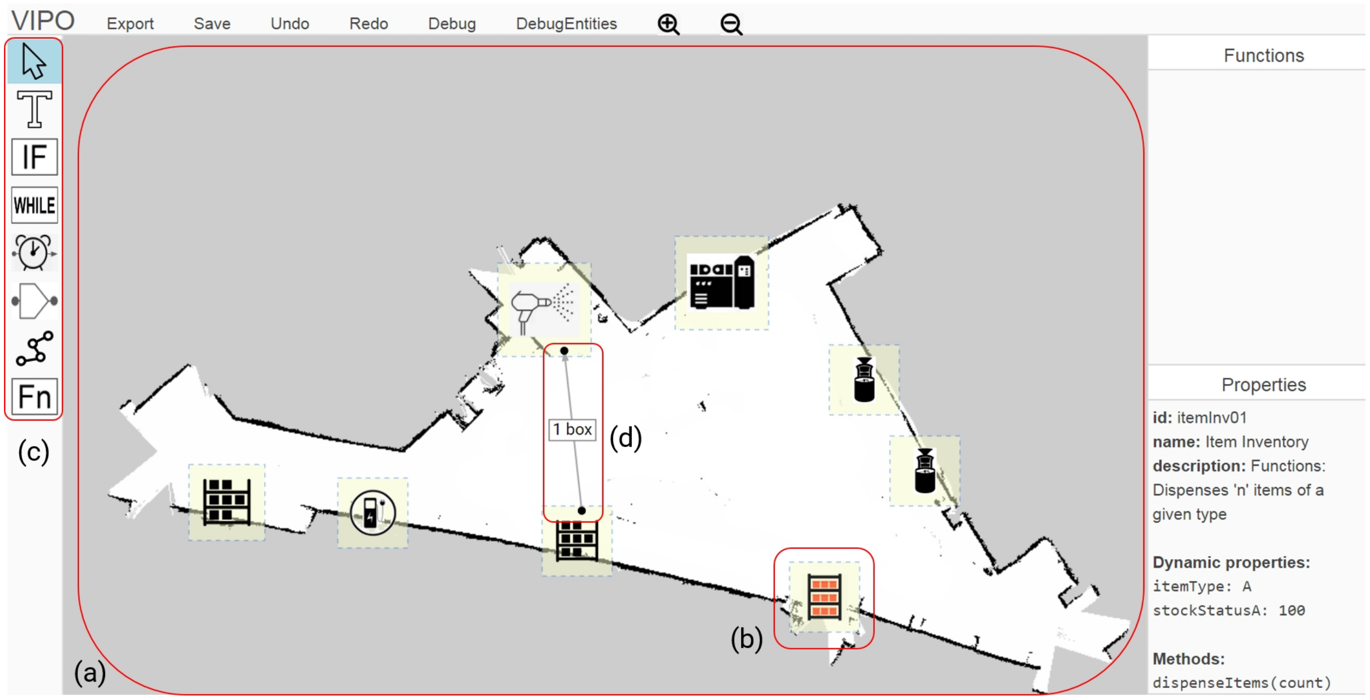
```python
class Task(object):
    def new_task(self):
        self.robot.pick({"value": 1, "object": "box"})
        self.robot.move(self.painting_machine.location)
        self.robot.drop({"value": 1, "object": "box"})

    def run(self, ctx):
        self.robot = ctx.get("robot01")
        self.painting_machine = ctx.get("paintMachine01")
        self.new_task()
```

Users can click the "Run" button and send the whole task script to the ROS Master, which will manage the task execution and communication between robots and involved IoT nodes.

IoT Nodes

IoT nodes are made for two broad categories of devices - Autonomus Mobile Robots (AMR) and Industrial Machinery.



id: itemInv01
name: Item Inventory
description: Functions: Dispenses 'n' items of a given type

Dynamic properties:
```
itemType: A
stockStatusA: 100
```

Methods:
```
dispenseItems(count)
```

Figure 2. The user interface of VIPO. a) Scanned layout map of a smart environment, b) IoT Nodes on the map, c) toolbar for creating programming notations, d) a Pick-and-Drop notation meaning robot picks 1 box from one IoT Node and drops to another IoT Node

Syntax	Toolbar Icon	Spatial Visual Symbols	Textual Meaning
Move		B B	`Robot.move(B.location)`

Pick and/or Drop		A obj
		obj B
		A obj B
			`1) Robot.pick(obj, A)`
			`2) Robot.drop(obj, B)`
			`3) Combine 1) and 2)`

Timer			`[IoT].operate(minutes)`
			`// E.g., Oven.curePaint(10)`
			`// wait for oven to cure`
			`// paint for 10 minutes`

If-Else	IF	If	`if (condition) {`
			`} else {`
			`}`

While	WHILE	While	`while (condition) {`
			`}`

Function	Fn	f(x)	`function f(x) {`
			`// one or more statements`
			`// defined above`
			`}`

Figure 3. A summary of VIPO toolbar icon, visual symbol, and according textual meaning (in pseudo code)

The system implements AMR and machinery using Turtlebot II and ESP32 microcontrollers respectively. A block diagram of a communication flow between ROS and IoT Nodes is presented in Figure 4.

ESP32 microcontroller act as a hardware interface layer for communication between ROS Master and Machinery. Working/control protocols for a given machine is specific and customized to the manufacturers' convenience which makes it nearly impossible for the ROS master to directly control the machine. ESP32 system is designed to bridge this communication gap between the ROS Master and machines. In addition to establishing a wireless communication, ESP32 is also intended to implement middlewares like the *MTConnect* for interoperability between machinery and software from different manufacturers.

The current system establishes and implements the protocol for wireless communication between ROS Master and ESP32 over the rosserial_server using the socket_node for interfacing multiple TCP clients. Currently, each ESP32 node is programmed to simulate a virtual manufacturing machine which has a set of operations predefined in the broadcasted *RDFMessage*.

RDFMessage is periodically broadcasted by each IoT node, it plays a vital-central role in our system. $ResourceDescriptionFramework/Format(RDF)$ is a family of $WorldWideWebConsortium(W3C)$ specifications to be used as a general method for conceptual description or modeling of information used by web-resources. In our system, we adopt a modified version of the RDF to suit our application, Figure 5 presents a sample RDF message for

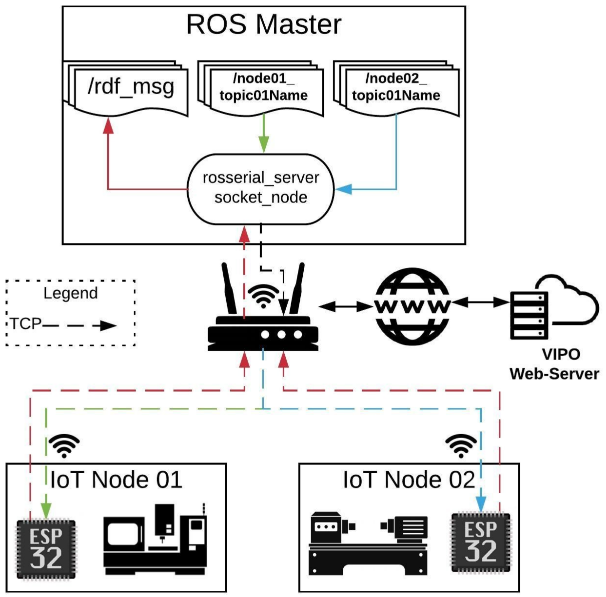


Figure 4. Block Diagram of communication flow between ROS and Iot Nodes

a paint mixing machine. The modified RDF message has device-information fields (ID, name, description, location etc.), machine-specific *Methods* and *Properties*. *Methods* are the functions that the machine is capable of performing (like mix-paint, set-temperature, start, stop etc.) and *Properties* are the real-time operation parameters (like job-status, temperature, coolant-level, run-time, health-status etc.)

The information from the RDF Message is of paramount importance for the functioning of our system. The RDF message fields discussed above are directly queried by the VIPO subsystem for (1) generating the $Web-UI$ and (2) generating the $Task-Script$. Fields like the ID, location, name, imgUrl, properties, methods etc. are used for generating-rendering the map and populating the drop-downs of the programming constructs (pick, drop, if, while etc.). Further, VIPO web-server uses information from the Methods-fields (like the name of the topic, ROS message type etc.) to generate the data access layer Task-Script for task execution.

Each IoT Node advertises the names of the topics that it subscribes and publishes on the $ROSMaster$. Each machine function has a dedicated topic and the machine can be controlled by publishing a message on this topic. A machine function is triggered by publishing a message to the topic subscribed the machine and is terminated with a $done = true$ (a field in the RDF Message) at the completion of the job. Referring to Figure 4, we can control the machine represented in IoT Node 01 by publishing an appropriate message on the topic node01_topicname.

ROS Master

ROS Master acts as the two-way communication bridge between VIPO and the IoT Nodes/robots in physical environments. It runs on top of ROS (Robot Operating System [9]). On one hand, it collects real-time working status of IoT Nodes/robots and sends to VIPO. On the other hand, it receives the user-defined task script from VIPO and sends to corresponding IoT Nodes/robots for execution. In particular, it uses a library called *vipolib* to automatically handle these two roles. The technical details are described below.

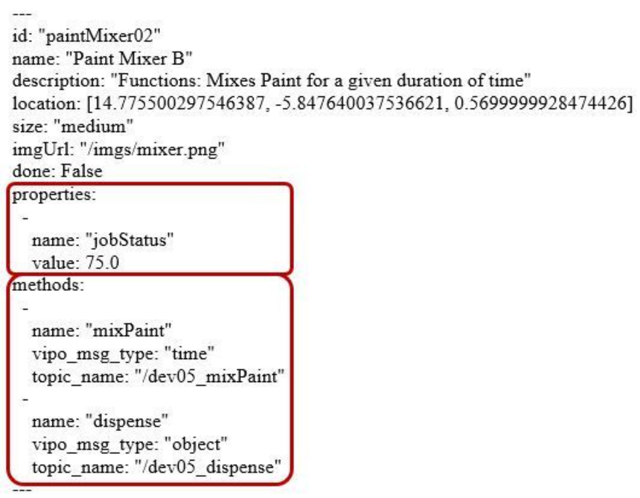

```
---
id: "paintMixer02"
name: "Paint Mixer B"
description: "Functions: Mixes Paint for a given duration of time"
location: [14.775500297546387, -5.847640037536621, 0.5699999928474426]
size: "medium"
imgUrl: "/imgs/mixer.png"
done: False
properties:
  -
    name: "jobStatus"
    value: 75.0
methods:
  -
    name: "mixPaint"
    vipo_msg_type: "time"
    topic_name: "/dev05_mixPaint"
  -
    name: "dispense"
    vipo_msg_type: "object"
    topic_name: "/dev05_dispense"
---
```

Figure 5. Sample RDF Message for Paint Mixing Machine

vipolib

A library called *vipolib* is designed to play as the core of the ROS Master. It has a *Context Manager* that stores all connected IoT Nodes/robots. It also has a *ROS Code Generator* that converts VIPO generated script to ROS executable code line by line and sends it to the IoT Nodes/robots. *vipolib* works in a manner as follows:

1) From IoT Nodes/robots to *vipolib*, then to VIPO
After receiving RDF messages from IoT Nodes/robots, *Context Manager* registers the ID, name, location, status, and capabilities of each newly connected IoT Node/robot. Note that the *Context Manager* is being updated once new RDF messages are sent to the ROS Master. If an IoT Node/robot has been registered, *Context Manager* will compare its subsequent RDF messages with the existing one and only send the difference to VIPO via web-based socket server in order to reduce the network traffic. After receiving those messages on VIPO's side, IoT Nodes/robots will be rendered accordingly, which has been discussed above.

For instance, ROS Master receives a RDF message from a robot, which contains ID, *robot01*, a capability called *move*, a required input called *location* and a subscribing topic name called *movetogoal*. Then in *Context Manager*, all information are registered together as a capability of *robot01* that can be visualized in VIPO while programming.

2) From VIPO to IoT Nodes/robots via *vipolib*
As mentioned before, VIPO will send a script about the whole task through socket server as pure string. ROS Master will call $exec()$ to convert the string into Python executable code and execute it in a new thread. Next, for each single line of

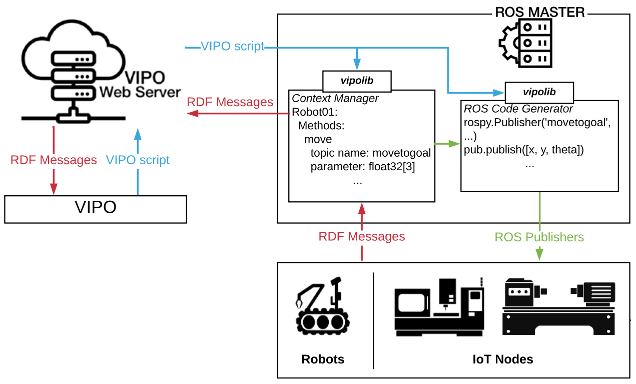


Figure 6. Block Diagram of ROS Master

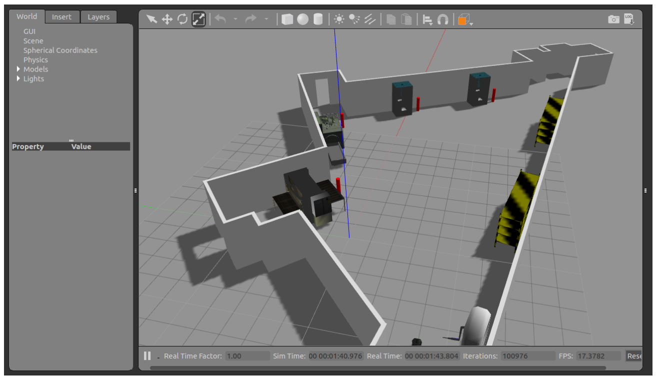


Figure 7. GAZEBO 3D Interface

code, *vipolib* will search in *Context Manager* required ID, capability, parameter and topic name and run the *ROS Code Generator* following a format given by pre-registered context. In specific, *ROS Code Generator* generates a publisher which publishes required parameters to corresponding topic name in order to execute the current command. To finish the whole task automatically, ROS Master needs to execute the next line of the script just after the previous command is done. Here uses one variable in RDF message called *done*. During working time of IoT Nodes/robots, this variable keeps to be *False*. Once the current job is finished, it changes to *True*. Since RDF messages are sent to ROS Master synchronously, *vipolib* is able to catch the change of the job status *done* and move to the next line of VIPO script.

For example, when executing a move command, *ROS Code Generator* fetches key words from the current line of VIPO script such as *Robot.move(IoT01.location)*. Given information from *Context Manager*, *ROS Code Generator* creates a publisher publishing the location of IOT01 to the topic *\movetogoal*. Since the robot keeps subscribing to this topic, once it receives new location, robot starts to move to the target location. After it reaches the target location, it publishes a RDF message with *done* to be *True*. Then, ROS Master recognizes the change of RDF message and starts the next line of VIPO script. Note that, once new command is being executed, *done* is reset.

In a word, ROS Master keeps publishing to corresponding topic name required information of each line of VIPO script and receiving updated RDF messages of IoT Nodes/robots and registering them to *vipolib/Context Manager*. Such architecture guarantees that VIPO is able to synchronize working status of IoT Nodes/robots and IoT Nodes/robots can receive new commands simultaneously. Moreover, ROS Master bridges physical IoT Nodes/robots together with virtual simulation through publishers as well. We will discuss it in the next section.

Simulation Engine - GAZEBO

A simulation engine is required to visualize the whole task generated by VIPO for two reasons. One is to pre-execute the task before pushing it to physical IoT Nodes/robots so that user may modify the task on implementation level before the task begins in physical world. While the other is to improve compatibility with virtual IoT Nodes/robots which will be discussed in Section Limitations and Future Work.

GAZEBO is a 3D dynamic simulator designed for robotic applications, which provides necessary tools to simulate robots in virtual 3D environments. It is introduced as the simulation engine for three reasons. First, it provides an accurate representation of physical properties, so that transitions from digital to real applications are easier to implement. Second, it removes any physical hardware constraints, allowing faster testing and prototyping. Third, GAZEBO presents well-defined ROS plug-ins which follow mainly used communication protocols between ROS Master and IoT Nodes/robots.

As for working space initialization, environment boundaries and locations of IoT Nodes/robots are manually generated based on mapping process. Meanwhile, 3D CAD models for IoT Nodes/robots are provided by vendors. Ideally, these models should be able to finish all actions described in RDF messages in GAZEBO. However, we provide a solution when vendors can only provide a static 3D model which will be discussed later.

After setting up the simulation working space, user can start programming tasks on VIPO. Once a task begins, all 3D models keep subscribing to same topics described in corresponding RDF messages of physical IoT Nodes/robots. For instance, a 3D robot model in GAZEBO is subscribing to a topic for moving, which is the same topic as physical robot subscribes to, called *\movetogoal*. As mentioned in ROS Master section, one line of the VIPO script will be converted to a publisher by *ROS Code Generator*. Since IoT Nodes/robots in GAZEBO and in physical world are subscribing to same topics, they will execute the same command simultaneously. Note that GAZEBO simulates all dynamic features happening in the real world such as frictions, inertia, and so on. The connection between ROS Master and GAZEBO is shown in Figure 8.

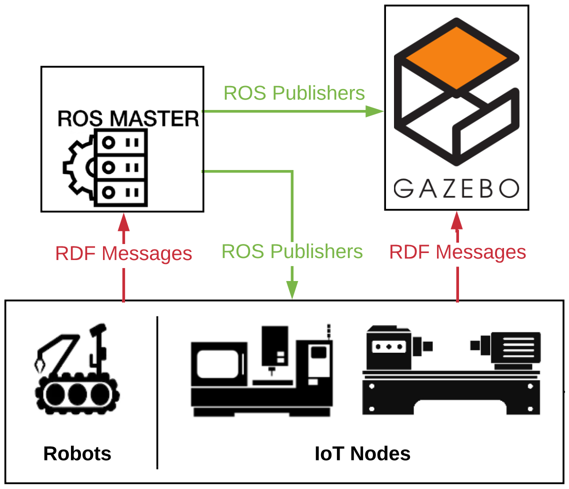


Figure 8. Block Diagram of Simulation Engine - Gazebo

As mentioned above, if vendors cannot provide a 3D model which can animate capabilities described in RDF messages, we will create an indicating object standing next to the model in GAZEBO and let it subscribe to topics of those capabilities. Once it receives new messages, it can appear/disappear, change color, blink based on different incoming published messages. For instance, suppose a robot model is able to fetch objects but cannot do it in GAZEBO. When it needs to execute *pick* command, in GAZEBO, an item will appear on the top of the robot after the robot finishes a *pick* command while in physical world, the manipulator on the robot will finish *pick* directly.

By introducing the Simulation Engine, the whole system becomes more robust in terms of being compatible with IoT Nodes/robots with more complicated functionality. Meanwhile, the system can handle virtual IoT Nodes/robots collaborating with physical IoT Nodes/robots with the help of GAZEBO visualization and simulation, which will be discussed later.

USE CASES

Painting Industry

To test and validate the system developed in an industrial setting, we chose to deploy our system to author an AMR for sequential tasks (locomotion and machine interaction) for a small scale painting industry.

The painting industry presented in the scenario is assumed to have an advanced factory setup with smart machine and AMRs. Floor plan of the factory with the smart-machinery (IoT Nodes) are presented in Figure 9. The following operations are required to happen in sequence for fulfilling a typical job-order with the help on an AMR.

1) Sourcing the parts to be painted from the Item Inventory;

2) Sourcing paint cans for the Paint Inventory;

3) Mixing the paint to obtain a given shade using Paint Mixing Machines;

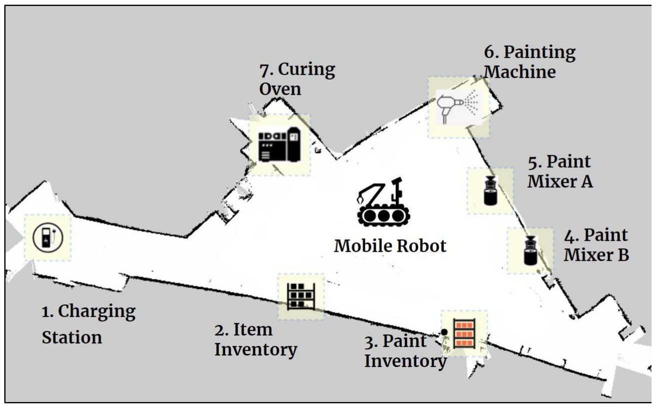


Figure 9. Factory Floor Plan

4) Paint the parts using the Painting Machine;

5) Cure the painted parts at an elevated temperature in the Curing Oven.

System Description

A local WiFi router was used to bridge the network between VIPO web-server, ROS Master and IoT Nodes. ROS Master was deployed on a Linux workstation and was running roscore, rosserial_server, socket_node, ros_master, Gazebo and websockets for communication with the VIPO web-server. VIPO web-server was deployed on a cloud-server hosted by Purdue Engineering Computer Network and the web-UI was set up on the browser of another laptop workstation. The use case deploys the Turtlebot II and seven IoT Nodes (ESP32 microcontrollers) for realizing the use case.

ESP32 modules were programmed to simulate virtual manufacturing machines which had a set of operations predefined in the broadcasted RDF Message. Each IoT Node had an LCD 1602 module for display of textual messages. For example, the start of a curing cycle would result in the message âĂIJCuring StartedâĂİ and completion of the same would result in the message âĂIJCuring CompleteâĂİ on the LCD.

Gazebo was used to simulate/visualize the virtual elements (AMR, 3D models of the manufacturing machines etc.). A virtual model of Turtlebot II (AMR) equipped with LIDAR was deployed on Gazebo to visualize navigation/locomotion. Interaction of the AMR with IoT Nodes was shown by the Status Lights placed beside each machine with the status light turning green after completion of the machine function. For example, completion of dispensing of items would result in green-lighting turning on followed by the robot picking the item and moving to the drop location.

A cardboard prototype of the scaled down floor plan of the factory was fabricated with ESP32 modules spatially positioned at their respective locations. The setup is presented in Figure 10

Workflow

The workflow deploys the Turtlebot II and seven IoT Nodes for realizing the use case. IoT Nodes are powered, followed by running rosserial_server and socket_node which establishes

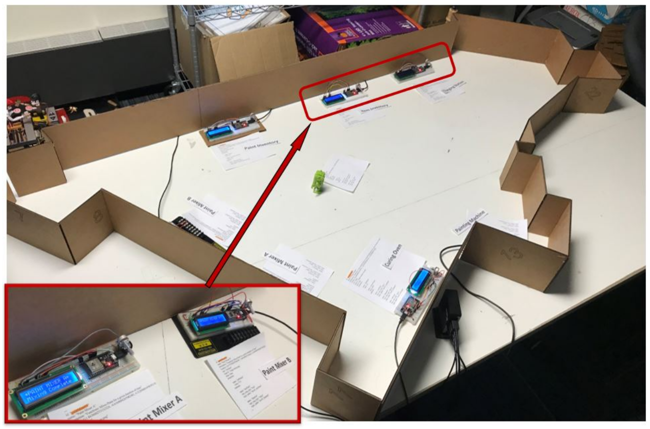


Figure 10. Prototype of the factory floor plan with IoT Nodes

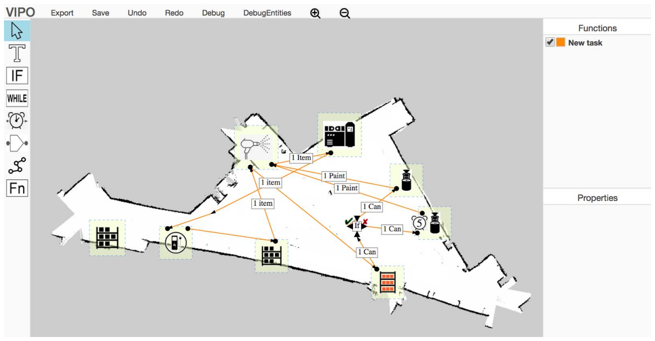


Figure 11. Web based visual programming interface for the painting factory

communication between the Linux workstation and IoT Nodes over TCP. Next, ROS Master is launched, which runs several startup applications such as the socket-client, IoT Node registration, launching Gazebo etc.. Successful connection to the VIPO web-server (socket-server) and IoT Nodes displays the âĂŸconnection successfulâĂŹ message marking the completion of the startup routine.

The next step is to deploy the AMR for generating the 2D Map (using LIDAR). The robot scans the workspace using the LIDAR and builds a SLAM map which is sent to VIPO web-server for rendering the web-UI. Further, the 2D map is extrapolated to generate the 3D Gazebo environment with 3D models of machines (performed manually).

VIPO web-server uses the SLAM map and the location data from the broadcasted RDF Message to generate the web-UI. Next, the user program the AMR to perform the sequential operations (step 1 through 5) mentioned in the previous section. A screenshot of the visual program for the entire workflow is presented in Figure 11. Successful compilation of the program results in the Task-Script that is manually checked for bugs, changes to the program are suitably made to correct the errors (if any). Further, the program is run to start the AMR workflow sequence.

The locomotion of the AMR can be seen to real-time on Gazebo. Interaction of AMR with the IoT nodes can be seen

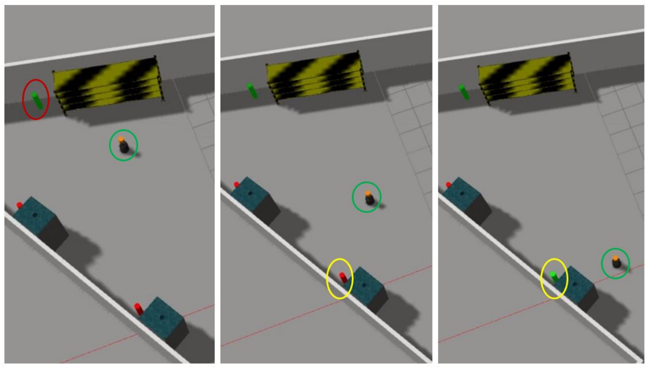


Figure 12. Paint-mixing Gazebo simulation

both on Gazebo and physical LCD screens. For example, consider the sub-task of mixing the paint which involves four sequential tasks - (1) Pick up the paint can from paint inventory (2) transport and drop the paint can to the paint mixer (3) start the paint mixer and (4) pick up the paint-can from the mixer. Task (1) results in a paint can appear on top of the robot, (2) results in the movement of the robot with the paint can to the mixer (3) results in commanding the mixer to start the mixing operation (LCD displays the âĂIJMixing StartedâĂİ text message) (4) Completion of the mixing operation will result in LCD displaying âĂIJMixing CompletedâĂİ text message followed by the Staus-Light turning green and paint can appearing again on top of the robot (in Gazebo). Results of Gazebo simulation for the above-discussed sub-task is presented in Figure 12.

LIMITATIONS AND FUTURE WORK

In this section, we will briefly discuss the limitations of the current visual language. Meanwhile, we will mention some promising paths to boost the power of the system.

Enhance Visual Language

Currently, we just provide Move/Pick/Drop/If-else/While to users. Next, we will add more functionality to VIPO such as Try-Catch to handle exceptions. Moreover, users can only define a *function* without passing any parameter. So the next step is to design a concise notation to allow users to define "function with parameter", which lets user create more reusable programs.

Support Multiple Robots Collaboration

Since VIPO is a web-based programming interface, multiple users can generate multiple tasks in the same working space simultaneously or single user can create multiple tasks for multiple robots. To achieve that capability, we may need to introduce optimization algorithm in order to let multiple robots collaborate efficiently.

Enable Virtual/Physical IoT Nodes/robots Collaboration

As discussed in Simulation Engine section, introducing GAZEBO provides possibilities to create virtual/physical collaboration tasks. User can program both virtual and physical IoT Nodes/robots on VIPO and visualize the task and collaboration on GAZEBO.

Evaluation and User Studies

Since one of the main contributions is the advantage of visual spatial programming, we must compare our programming method with commonly used methods such as textual programming and block programming. We will assign different programming tasks to users and let them program with different methods. At the same time, we will design evaluation criteria such as finishing time, total errors made, readability and so on in order to analyze and evaluate advantages and disadvantages of our system.

CONCLUSION

In this paper, we introduced VIPO, a programming language that enables novice users to program tasks for robots and IoT Nodes. VIPO consists of a novel spatial and visual programming language, a web-based editor for this language, and a ROS Master. Users' IoT Nodes and robot(s) can be dynamically/automatically registered into VIPO through ROS Master. This is achieved by sending the information (e.g., status, and location) as RDF messages.

Once users program a task on web-based UI, the visual program will be compiled into textual task script and sent back to ROS Master. Then ROS Master will control the execution of IoT Nodes/robots. To better understand the execution, a simulation engine called Gazebo is used.

REFERENCES

1. James P Diprose, Bruce A MacDonald, and John G Hosking. 2011. Ruru: A spatial and interactive visual programming language for novice robot programming. In *Visual Languages and Human-Centric Computing (VL/HCC), 2011 IEEE Symposium on*. IEEE, 25–32. https://ieeexplore.ieee.org/abstract/document/6070374
2. Ke Huo, Yuanzhi Cao, Sang Ho Yoon, Zhuangying Xu, Guiming Chen, and Karthik Ramani. 2018. Scenariot: Spatially Mapping Smart Things Within Augmented Reality Scenes. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. ACM, 219. https://dl.acm.org/citation.cfm?id=3173793
3. Jared Jackson. 2007. Microsoft robotics studio: A technical introduction. *IEEE robotics & automation magazine* 14, 4 (2007). https://ieeexplore.ieee.org/abstract/document/4437755
4. Kamna Jain. 2007. D-RDF: Dynamic Resource Description Framework. (2007). https://lib.dr.iastate.edu/cgi/viewcontent.cgi?article=15852&context=rtd
5. Joseph Lee, Yan Lu, Yiliang Xu, and Dezhen Song. 2016. Visual programming for mobile robot navigation using high-level landmarks. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*. IEEE, 2901–2906. https://ieeexplore.ieee.org/abstract/document/7759449
6. Joaquín López, Diego Pérez, Enrique Paz, and Alejandro Santana. 2013. WatchBot: A building maintenance and surveillance system based on autonomous robots. *Robotics and Autonomous Systems* 61, 12 (2013), 1559–1571. https://www.sciencedirect.com/science/article/pii/S0921889013001218
7. Joaquin López, Diego Pérez, and Eduardo Zalama. 2011. A framework for building mobile single and multi-robot applications. *Robotics and Autonomous Systems* 59, 3-4 (2011), 151–162. https://www.sciencedirect.com/science/article/pii/S092188901100011X
8. Emmanuel Pot, Jérôme Monceaux, Rodolphe Gelin, and Bruno Maisonnier. 2009. Choregraphe: a graphical tool for humanoid robot programming. In *Robot and Human Interactive Communication, 2009. RO-MAN 2009. The 18th IEEE International Symposium on*. IEEE, 46–51.
9. Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. 2009. ROS: an open-source Robot Operating System. In *ICRA workshop on open source software*, Vol. 3. Kobe, Japan, 5.
10. Franz Steinmetz, Annika Wollschläger, and Roman Weitschat. 2018. RAZERâĂŤA HRI for Visual Task-Level Programming and Intuitive Skill Parameterization. *IEEE Robotics and Automation Letters* 3, 3 (2018), 1362–1369. https://ieeexplore.ieee.org/abstract/document/8269311