From Robots to Humans: Visualizations for Robot Sensor Data

Miguel Campusano Johan Fabry
PLEIAD and RyCh labs, Computer Science Department (DCC), University of Chile, Chile
{mcampusa,jfabry}@dcc.uchile.cl

***Abstract*—In a robotics context, visualizing the data scanned by a robot is crucial to understand what the robot's sensors perceive about its environment. Consequently, robotic visualizations show these values in a 3-D world, such that they can be compared with the real world. However these visualizations do not allow developers to see this data in a manner that allows it to be interpreted for program construction. As a result, these visualizations are in many cases ineffective for programming robot behaviors. To address this issue, we have built several visualizations of robot sensor data for the programming of behaviors, and we report on them here. Our visualizations focus on better revealing the hard data, which allows developers to faster understand it and consequently to faster create and adapt robot behaviors.**

I. Introduction

When we build programs, we focus not only on their structure and behavior but also on their inputs, which cause them to behave in different manners depending on the data they receive. However building the behavior of a program may be a troublesome task when inputs are too complex to understand at first glance. For example, consider a function receiving a `10x2` matrix that represents the `x` and `y` positions of an object over time. It is difficult to imagine the route of the object over the `xy` plane just by looking at the matrix.

The above issue is especially true when we build programs using live programming [1]. In live programming, developers can change the program while it is running without the need to restart it, and these programs actively provide immediate feedback to developers about their execution. While live programming allows for fast understanding and prototyping of a program, if the inputs of this program are not easy to understand, then the live programming technique may lose its effectiveness. Considering the previous example: even when developers can see the numbers in the matrix, if it represents many points over time they will lose a significant amount of time analyzing the data to understand the route. This prohibits a fast understanding of the program and makes live programming impossible.

Within a robotic context, much of a program's input is provided by the robot's sensors. This data is likely to be difficult to understand at first glance because of its high complexity. For example, laser scanners in used in robotics provide scans that can have 1024 values or more, and these scans are received by the program every tenth of a second. As a result, if we combine live programming with robotics, to achieve a quick understanding of the robot behavior, it is imperative to have a fast understanding of the inputs of the robot. In our experience using live programming for robots, we have needed to dedicate a lot of time trying to understand the data given by a laser scan. Finally, when the data was understood, writing the behavior that we wanted was a simple task which was quickly accomplished.

A need for adequate visualization of robot sensor data has been identified before, for example in the talk about *seeing spaces* by Bret Victor [2], which shows a projection of a graph of robot sensor data next to the robot itself. Extrapolating from this, ideally we would have a general purpose robot dashboard that shows program execution and multiple dynamic visualizations of robot sensors geared to quickly understand the hard data. It should be generic to different robots and sensor configurations, configurable, and support live updating of data and programs.

Yet current robot sensor data visualizations fall far behind of this ideal. We noticed this while working with ROS [3], the leading middleware for robot programming. Even while there are visualizations for robotic data available, they typically focus on showing the integration of data of multiple sensors at once, in a simulated environment. An example of this is the standard visualization environment of ROS, RVIZ [4]. Using such a visualization achieves a better understanding of what the robot "sees", drawing a representation of the data in a simulated environment. It however does not provide an analytical visualization for the data of a specific sensor, making it difficult to understand the hard data required to build programs. By displaying laser scans in RVIZ, developers can see the outline of the object and have an idea of the distance and position of the object, but they cannot see its exact distance, nor its exact angle.

For our work we wanted to better and faster understand sensor data, this as existing visualizations do not allow for understanding the hard data at a glance, so that it can be used in robot behavior code that consumes it. To address this issue, we built visualizations for three different robot sensors. In this work we present these visualizations and show how they can be helpful in building behaviors for robots. The goal of these visualizations is to aid in quickly understanding the data used by these programs, such that we maintain the rapid development cycle of live programming.

More in detail: we built three visualizations: laser scans, touch sensors and robot speed. Each of these significantly

978-1-4673-7526-9/15 © 2015 IEEE

improves on existing RVIZ visualizations. The laser scan visualization shows at first glance an approximate distance and position of an object and provides tools to know the exact distance and angle of every datapoint of the laser scan. The touch sensor visualization allows knowing the state of every touch sensor in a robot, which is absent in RVIZ. Finally, the speed visualization shows the speed of a robot in a graph, while in RVIZ robot speed can only be estimated by looking at the movement of the robot over time.

This paper is organized as follows. Section II elaborates on the problem. Then, in Section III we present our three custom, dynamic visualizations. Lastly, we talk about related work in Section IV, and end with conclusions and future work.

II. Motivation: Live Programming and Robot Data

Live programming is a software engineering technique that accelerates the building of software by enabling continuous real time modifications of running programs, *i.e.,* changing the code without the need for restarting. Program understanding is reinforced by a continuous feedback loop of the changed behavior or a visualization of the running program. With this live feedback, developers immediately see the results of their changes and hence understand the changed behavior faster.

In previous work, we built a live programming language for robots, appropriately called Live Robot Programming [5] (LRP[1]). Using this language we have been performing experiments on how live programming can be used in a robotic context, with positive results. In LRP, robot behavior is written as nested state machines. A hierarchy of machines, each with their states, transitions and events, represents the different behaviors and change of behaviors. LRP provides a visualization where the hierarchy of machines and each machine's states and transitions can be seen. The visualization provides continuous feedback to developers when programs are running. Because of its liveness properties, in LRP developers can modify, add or remove behaviors and the resulting program can be immediately seen in the visualization and moreover is immediately present on the running robot. If the robot was executing a behavior, the behavior continues its execution even when the program changes.

We however experienced one important issue when working with robots in LRP: We need to better understand the continuous stream of sensor data coming from the robot. We found that on many occasions, at first glance the data of a sensor may be too complex to understand. For example, robots use laser scanners to measure distances to the nearest objects, over a wide angle. Laser scan data can be quite complex because the laser returns info about the limits of the scan – the minimum and maximum distance at which the laser recognizes objects –, the start and end of angle of scan, and last but not least the distance of every point in the scan. One example laser we have: the base laser on our PR2 robot [6], measures up to 60 meters in distance, covers 260 degrees, and one scan consists of 1040 datapoints.

[1]Web site with examples and download: http://www.pleiad.cl/lrp

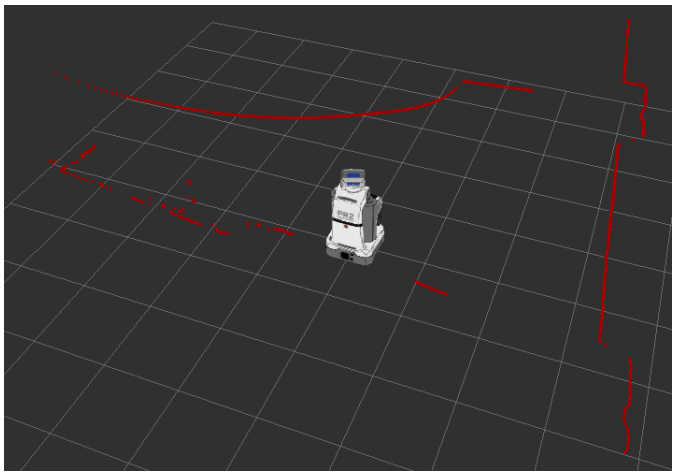

Fig. 1. A laser scan from a PR2 as seen in RVIZ. The laser scan shows the walls of a part of the Department of Computer Science.

A developer cannot interpret this data at a glance and needs to perform some analysis to obtain the information that is relevant for the task at hand. For example, establishing what the distance is between the robot and the nearest object, and where this object is, cannot be performed in a reasonable amount of time just by looking at the textual data of 1040 datapoints. This issue is more critical in a live programming context where we want to write prototypes and iterate over them as quickly as possible, and the focus is to quickly understand our program while it is executing. While the LRP programming language follows this principle, in our experience interpretation of data, *e.g.,* from a laser, is a major impediment to understanding the inputs for the program and thus to understand the behavior.

To address this issue we created a set of custom visualizations for robotic sensor data. Our main goal is to be able to interpret as quickly as possible the raw data received by the robot, so that it can be used in the programs being created.

In our robot experiments we use ROS [3] because it is becoming the de-facto standard robot middleware, with a large amount of packages and tools to work with different robots. ROS already comes with a standard 3-D visualization environment: RVIZ [4]. It provides a GUI to visualize sensor data, robot models and environment maps. With this, developers can visualize what is happening with the robot in the environment. Figure 1 shows an example visualization of our PR2 robot scanning the environment using its base laser.

As RVIZ focuses on representing the data in the environment rather than the data itself, it does not give access to this data for analysis. As a result it does not provide any help in interpreting this data so that it can be used in a program. Because of this, understanding and using the data received by ROS in our programs is not an easy task. For example, just by looking at the arrays of scan values of the PR2 laser it is not obvious that the first value of the array (the leftmost value) actually represents the rightmost point of the scan, while the

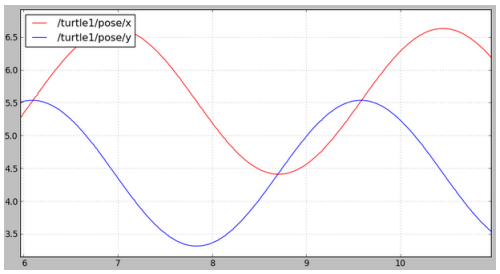

Fig. 2. rqt_plot visualization of the `X` and `Y` position of a robot.

last one (the rightmost value) is the leftmost point of the scan!

In addition to RVIZ, ROS provides another tool to visualize data: rqt_plot. It shows a scrolling time plot of data of a robot. In Figure 2 we can see an example of the `X` and `Y` position of a robot over time. This tool can be useful to see how the data changes over time, but it is limited to simple numerical values. Because of this, even when this tool allows to visualize multiple numerical values, it may be difficult to understand this values in their context. For example, in Figure 2 we can see the positions `X` and `Y` separately thanks to rqt_plot, but it makes more sense to have a map to show where the robot was positioned over time, as done by RVIZ. Moreover, rqt_plot does not have a way to interpret more complex data, such as a laser scan.

Note that even though we are motivated by our experiences with robots in ROS, our visualizations should not be coupled to be used only with ROS. The idea behind this is to allow for a transformation of the data from a given robotics system to our custom visualization. With this in mind, our work is not restricted just for ROS, but it may be adapted to any framework or middleware.

III. Visualizing Robot Sensor Data

We built specific visualizations to help developers quickly understand the data received by a robot in a live programming context. This understanding should aid the programmer in creating and modifying programs that handle this kind of data. Our visualization is also live, *i.e.,* the data is automatically updated over time.

So far we have built 3 visualizations for different sensors. We expect to build more as needed in order to build more behaviors. In this section we talk about the visualizations we have built: laser scan, robot speed and touch sensors.

A. Laser Scan

A laser scan is a sensor measurement that states the distance of objects in a wide angle of vision. The data we show in the visualization is not only the objects detected by the laser, but also we give visual indications about the intrinsic properties of the laser scan and the distance of the objects themselves, as is shown in Figure 3. Note that we visualize a 2-D laser scan, *i.e.,* a laser scan with data only on a plane.

The intrinsic property of a laser scan sensor is the area of object detection. It is defined by the minimum and maximum

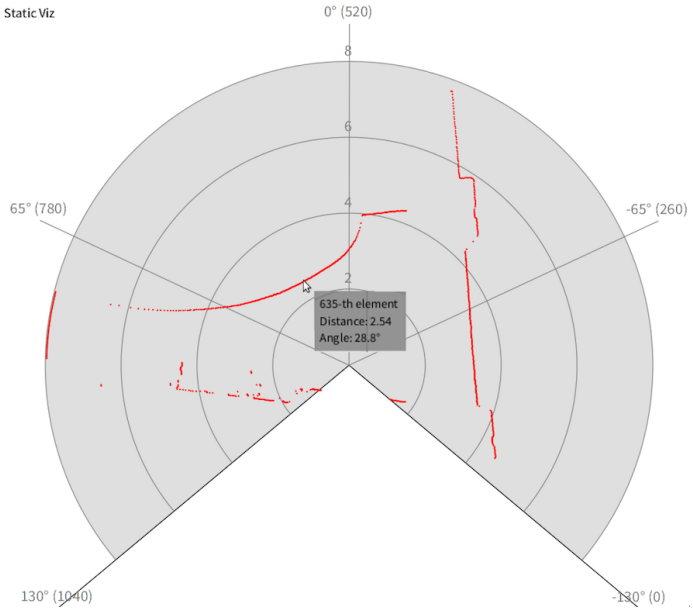

Fig. 3. The static version of our laser scan visualization, showing the same data as in Figure 1. The visualization shows the area of detection of points, indications about the position of points and a pop-up with more information about the 635th point in the array of datapoints.

distance an object can be detected and the sweep angles of the laser. This is shown in the visualization by a gray area, which is a scaled representation of the distances and sweep angles of what the laser can scan in the real world.

In the visualization we also give indications that help in approximately determining the position of diverse objects within the area of detection. We do this firstly by drawing curved lines that divide the sweep area, indicating the radial distances of the objects. We label the curved lines to indicate the radial distance of the different lines in the visualization. The amount of curved lines and their distances from the robot can be customized by the developer. Secondly, we draw a number of radial lines, customizable by the developer, to divide the area of detection in equal parts. We label the radial lines with the angle from the center of the robot and approximation of the index of the array of datapoints that represents that angle. All of this is done to understand as quickly as possible the laser scan data and the position of the objects inside. Scan data is plotted as a set of red dots, each red dot shows the distance measured to the nearest object for that scan angle. With all the indications, at first glance developers can approximately know where the objects are positioned inside the range of the laser scan. The red dots are updated live, *i.e.,* when the objects or the robot move, the red points also change to indicate the new distribution of objects in the detection area of the laser.

Moreover, the dynamic visualization can be frozen, providing a static part where developers can focus on a specific point in time and have more information about the data in that moment. The static visualization adds a pop-up on every

datapoint representing the laser scan that gives information about the index of the point in the array of datapoints, the distance and the angle of the point. Figure 3 shows the static part of the visualization, with a pop-up on the 635th point.

Figure 3 is an example of the static visualization. We can see the walls of an aisle inside the Department of Computer Science of the University of Chile. The figure shows the same data as in Figure 1, but with more information about the hard data. In the figure there is a pop-up showing more information about the point where the cursor is positioned: the 635th point in the array of datapoints. It is positioned at a 2.54 meters distance, at a 28.8 degree angle from the center of the robot.

If we compare this to RVIZ, their representation of the laser scan is a line of points that can be kept into the visualization to build maps and complete objects. However, it does not expose anything of the data itself, developers cannot see what is the actual distance of the object and the robot. Nor can developers know what is the range of the laser shown in the visualization as we do with the visualization shown in Figure 3.

In robotics, laser data is typically used as input for complex algorithms that have already been implemented, such as Simultaneous Location and Mapping or Path Planning. While multiple robotic behaviors can be realized by reusing such already implemented algorithms, our visualization allows developers to better understand the data used in these contexts, and to see if the robot is behaving as it should. Moreover, whenever developers want to build new algorithms, they will use the hard data of the laser to build them, and our laser visualization will be useful to them.

B. Speed

The speed of a robot is typically measured by a combination of linear speed and angular speed [7]. Both speeds are decomposed into the `X`, `Y` and `Z` axes of the robot. For simplification, in a vehicle we can say that a positive linear speed in `X` means that the vehicle moves forward and an angular speed in `Z` means that the vehicle turns on the spot. Technically speaking, the robot can provide speed as sensor data but speed can also be given as a control signal to the robot [7]. As a result, we can either visualize a speed control signal, considering it a sensor measurement of an ideal speed, or show speed as given by the sensor data, revealing all sensor noise.

If we know a robot's speed we can approximate how it is moving in the environment. We can know if the robot is moving too fast or not, and what kind of movement is it performing, *e.g.,* the robot is turning or moving in a straight line. We can see if the speed of the robot and the kind of movement at a single point in time is according to the behavior that a developer built for the robot. All of this can be seen while developers have access to the actual value of the speed.

We use a bar chart to visualize both linear and angular speeds, as shown in Figure 4. We choose a bar chart because it compactly shows all dimensions of the speed vector. When the robot moves backwards on `X`, the speed is considered to be negative, so in our graph bars go up and down to represent positive and negative speed, respectively. When the

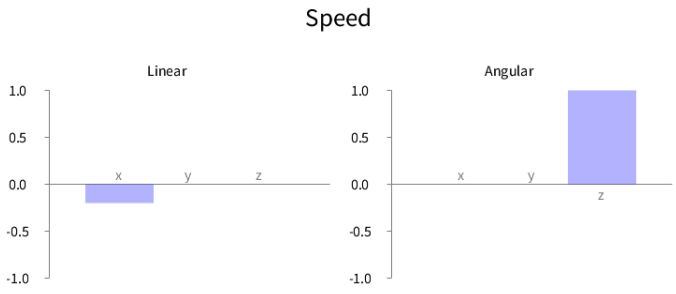

Fig. 4. Speed Visualization.

robot moves, the bars also change, visualizing the changes of speed of the robot. From our experience, a maximum speed of 1 and a minimum speed of -1 is a correct scale. Nonetheless both maximum and minimum speed are configurable.

Moreover, as the developer chooses the maximum and minimum, these can be the maximum speed of the robot itself, or an arbitrary limit the robot must not exceed, *e.g.,* for safety reasons. When a value exceeds the limit, the respective bar changes its color to red. This allows developers to see at first glance, without even reading the speed number, when the robot is exceeding maximum speed.

In contrast, RVIZ does not have an explicit visualization of the speed of a robot. So while the robot can be seen to move inside the 3-D world of RVIZ, developers can not know how fast the robot is moving, and if it is moving very slowly its movement may not be visible at all. In contrast, with our visualization, developers can see how the robot is moving while reacting to their programs, and also see the exact speed value. They can also see if the robot is moving or turning and can notice if the robot is accelerating by seeing that a bar is changing size over time.

Lastly, for robots moving on the ground, it makes more sense to just use the `X` and `Y` axis for linear speed and the `Z` axis for angular speed, because the robot is moving in just one plane [7]. Therefore, we built a variant of the speed visualization that only shows the `X` and `Y` bar for linear speed and the `Z` bar for angular speed. This instead of showing a bar chart where the `Z` bar of linear speed and the `X` and `Y` bars of angular speed will never change.

C. Touch Sensors

A touch sensor is a sensor that gives information whether an object is directly touching it. Our visualization of a touch sensor is a labeled rectangle that is white when there is no object touching the sensor and that turns green when an object touches it. The overall visualization can compose multiple touch sensors, as robots may have many. In Figure 5 we see our visualization for a Turtlebot robot [8]. It contains three touch sensors, positioned at the left, center and right of the robot. In this case, only the *center* sensor is being touched.

With this visualization a developer can understand which touch sensor is being activated when a robot is doing something – like moving – and make a decision based upon that.

Fig. 5. Visualization of three touch sensors of a Turtlebot robot.

For example, from Figure 5, if we see the Turtlebot robot being hit on the left bumper we can make the robot turn to the right to avoid that specific obstacle. If we see the robot being hit on the right bumper, we can make it turn to the left.

Such a visualization of touch sensors is however completely absent from RVIZ. There is no way for a programmer to see in RVIZ that a robot hit an object unless some other sensor detected it and in the simulation the object is drawn touching the robot. And even if the robot itself can be observed touching the object in RVIZ or in the real world, the programmer can not establish whether the touch sensor detects this.

IV. Related Work

RVIZ [4] is a fully 3-D visualization environment for robots in ROS [3]. RVIZ visualizes sensor data and state information from the robot, combining the data and visualizing it all into a single image that overlays the data on a simulated environment. However, RVIZ does not provide access to the hard data so that it can be interpreted by developers. One way to extend RVIZ such that it shows visualizations similar to ours is the use of *markers*. A marker is a simulated object inside the visualization that can be sized and placed programmatically. It would allow, for example, to draw the radial lines of the laser scan in the simulated world – curved lines are not available as markers – or to place cubes whose height represent current speed. These however still do not expose the hard data, *e.g.,* from a cube height the exact speed is not revealed. Moreover these constructs are positioned as normal objects in the 3-D environment, making them hard to read for certain camera angles, or if the camera is moving around.

An alternative is the use of augmented reality as in the work of Collet and MacDonald [9]. Their main focus is to represent the data in a real environment and they show examples using a laser and a sonar scan. The visualization augments a view of the real world, overlaying the area of detection of objects, cutting it short at the edge of detected objects. However the visualization does not provide a way to access the hard data.

Considering live programming, there are several live languages that have their own visualizations. We can trace back live programming to the work of Tanimoto on VIVA [1]. VIVA is a live programming language for image processing. It uses a graphical representation of programs as if they were electronic circuits: the data flows along wires, reaching components where it is processed. Data can be any Lisp value, but it is not shown in the visualization. A similar data-flow based visualization is used by Hancock in Flogo I [10]: a live programming language for robotics focused on teaching. The work shows numerical or boolean data for sensor readings, which is easy to interpret without any visualization.

V. Conclusion and Future Work

In this paper we presented three custom visualizations for different robot sensors, more specifically for a laser scan sensor, touch sensors and the speed of a robot. These visualizations are dynamic, *i.e.,* the data is automatically updated over time with the new data the sensors provide. In the laser scan visualization, developers can see the position of datapoints approximately at first glance, as well as their specific position by hovering the mouse pointer over a point. With the speed visualization, developers can see the exact speed of the robot and observe how it moves. Finally, with the touch sensors visualization, developers can visualize multiple touch sensors at once. None of the above can be done in RVIZ. The last visualization is not even present in RVIZ.

We built these visualizations because we wanted to understand at a glance the data given by robots, improving the experience of building programs using live programming. Because of this, our visualizations show important information about the hard data, such as the value of the speed for the speed visualization and the area of detection of the laser scan. In our experience using these visualizations in LRP [5] – our live programming language– on different robots, they significantly speed up the development of robotic behaviors.

Although we informally tested these visualizations, we still need to involve more developers to validate if they help in programming robot behaviors and if our visualizations are suitable for each case. We plan to perform a user study where developers will build behaviors either with our visualizations or using RVIZ. We will compare the solutions of both groups in regard to time taken and take a survey to obtain qualitative data about our visualizations.

Further future work is addressing a number of other sensors for robots that we have not included yet. We expect to include more sensors in our dynamic visualizations as needed. For example, the Kinect camera [11], used by different robots, produces point clouds which are nontrivial to interpret.

References

[1] S. L. Tanimoto, "Viva: A visual language for image processing," *Journal of Visual Languages & Computing*, vol. 1, no. 2, pp. 127–139, 1990.
[2] B. Victor, "Seeing spaces," Talk at EG conference, 2014, video recording available at http://vimeo.com/97903574.
[3] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in *ICRA workshop on open source software*, vol. 3, no. 3.2, 2009, p. 5.
[4] Willow Garage, "RVIZ – 3D visualization environment," web page http://http://wiki.ros.org/rviz.
[5] J. Fabry and M. Campusano, "Live robot programming," in *Advances in Artificial Intelligence–IBERAMIA 2014*. Springer, 2014, pp. 445–456.
[6] "PR2 robot," web page https://www.willowgarage.com/pages/pr2/.
[7] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics*. MIT press.
[8] "Turtlebot robot," web page http://www.turtlebot.com/.
[9] T. Collett and B. A. MacDonald, "Developer oriented visualisation of a robot program," in *Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*. ACM, 2006, pp. 49–56.
[10] C. M. Hancock, "Real-time programming and the big ideas of computational literacy," Ph.D. dissertation, Massachusetts Institute of Technology, 2003.
[11] Microsoft, "Kinect – camera sensor to interact with computers," Web page https://www.microsoft.com/en-us/kinectforwindows/.