Abstract—For mobile robots, mobile manipulators, and autonomous vehicles to safely navigate around populous places such as streets and warehouses, human observers must be able to understand their navigation intent. One way to enable such understanding is by visualizing this intent through projections onto the surrounding environment. But despite the demonstrated effectiveness of such projections, no open codebase with an integrated hardware setup exists. In this work, we detail the empirical evidence for the effectiveness of such directional projections, and share a robot-agnostic implementation of such projections, coded in C++ using the widely-used Robot Operating System (ROS) and rviz. Additionally, we demonstrate a hardware configuration for deploying this software, using a Fetch robot, and briefly summarize a full-scale user study that motivates this configuration. The code, configuration files (roslaunch and rviz files), and documentation are freely available on GitHub at https://github.com/umhan35/arrow_projection.

Index Terms—Robot navigation, navigation intent, projected augmented reality (AR), ROS, rviz, open science

I. INTRODUCTION

Robots are increasingly navigating around populous areas and moving along with people. This is true for service robots found in airports, hotels, restaurants, delivery robots on sidewalks, mobile robots in warehouses, and autonomous cars. For these robots to be deployed safely, humans around them must understand their navigation intent. HRI researchers have traditionally focused on arm movement intent [2], [3], exploring eye gaze [4], [5] and other non-verbal means [6] for stationary robots to better convey such intent. Recently, however, HRI researchers have begun to explore the externalization or visualization of robotic navigation intent (e.g. path plans) as well [7], [8], [9], [10], [11], [12], to enable more understandable robot’s navigation behaviors, so as to improve trust and acceptance.

One method for conveying navigation intent is through directional projections [7], [9], [10], [11] such as arrow projections [7], [11], where the robot is equipped with a projector and projects directional cues indicating which direction the robot intends to move. These cues can take the forms of lines [9], gradient bands [10], or arrows [7], [11]. Compared to head-mounted see-through augmented reality [15], [16], projector-based augmented reality does not require interactants to wear special hardware, allowing visualizations to be seen by many observers at once, facilitating usability in group, team, or crowd contexts [17], [18], [19]. As we will discuss in
Section III a number of human-subjects studies [7], [9], [10], [11] have shown the effectiveness of this approach [10], [11].

In this work, we share a practical solution for robotic arrow projections, describe the nature of the implementation of this solution, and provide access to the code for that solution. Additionally, we provide a specific robot and off-the-shelf projector that we have successfully used, as well as a summary of a full-scaled user study to provide more empirical evidence. For the implementation, we used the popular Robot Operating System (ROS) [20] and its visualization tool, rviz [21], for rendering the arrows. Using ROS made the implementation robot-agnostic, as most robots used in research and development have ROS support [22]. Even non-ROS frameworks [23], [24], and cognitive architectures, such as DIARC [25], typically include bridges to ROS. Using rviz as the rendering engine eliminates the need for practitioners and researchers to learn a tool outside of the ROS ecosystem, such as Unity. Moreover, rviz has a GUI and does not require any computer graphics programming.

Given the wide range of applications of arrow projection on different robots and its proven effectiveness and efficiency, we believe our work is highly relevant and beneficial to the HRI community. The readily available implementation with step-by-step documentation also allows researchers to focus on other interesting and emerging issues under social navigation, e.g., human navigation modeling [26], [27], [28], [29] and other interesting and emerging issues under social navigation, step-by-step documentation also allows researchers to focus on learning a tool outside of the ROS ecosystem, such as Unity. Moreover, rviz has a GUI and does not require any computer graphics programming.

By using arrows, other directional projections were proposed. Inspired by Park and Kim [32], Chadalavada et al. [9] proposed projecting a line with a grid onto the ground to indicate navigation path and collision avoidance range of a robot. Even with this primitive geometry element, Chadalavada et al. [9]'s work suggests potential for enhanced participant perception of the robot across multiple attributes when projections were used, including communication, reliability, predictability, transparency, and situation awareness. Although no statistical tests were ran, their descriptive statistics provide one of the first pieces of evidence for directional projection effectiveness.

Instead of lines, Watanabe et al. proposed projecting a gradient light band in a hallway to show the navigation trajectory to be followed by a robotic wheelchair with a wheelchair user [10]. The projection was compared with no projection, and with a projection paired with a screen. Their results suggested that adding projections enhanced comfortability and perceived motion intelligence.

Furthermore, Watanabe et al. [10] found that when projections were used, nearby walkers chose not to stop and stepped away towards walls earlier. These behavioral changes are further evidenced by Chadalavada et al. [11], who show that projection onto a shared floor space encourages participants to choose an alternative safer path.

Finally, Coovert et al. used arrow projection to show a robot’s path and investigated navigation intention prediction during different periods while the robot was avoiding traffic cones in a hallway [7]. Their work showed that adding projections led to observable differences at a variety of time scales, as participants were typically able to quickly identify the robot’s navigation intent. Moreover, participants also displayed high confidence in their identifications.

Researchers have also begun to recently compare between some of these previously presented visualizations. Chadalavada et al. [11], for example, recently compared line and arrow visualizations, and found that arrows are typically preferred compared to lines and blinking arrows. In our own work, we thus made an informed decision to use arrow projections to visualize robot navigation intent. We also added a solid circle to indicate navigation destination (Fig. 2 left).

III. HARDWARE SETUP: ROBOT, PROJECTOR, POWER

Before we detail the software, we first describe a hardware configuration in which we validated our software. In this configuration, a projector is mounted onto a Fetch robot [33], as shown in Fig. 1. Fetch is a mobile manipulator robot with a single 7-DOF arm mounted onto its chest. Its height ranges from 1.096m - 1.491m (3.596ft - 4.892ft). The projector we used is ViewSonic PA503W [34], as shown on the top of the images in Fig. 1. It used the DLP technology in its lamp, has a brightness of 3,800 ANSI lumens and 22,000:1 contrast ratio, making sure the projection is still legible in indoor environments with lights on. DLP has higher brightness than LCDs because LCDs are transmissive and the heat generated cannot be easily dispatched [35]. We chose this projector as our prototype, but other projectors will also work as long as the purchaser makes sure to consider these same three factors, i.e., the lamp technology, ANSI lumens and contrast ratio.

Although we chose the specific robot and projector, our implementation is robot- and projector-agnostic. The only requirement in terms of hardware is the mounting point of a...
Algorithm 1: Evenly Space Out ROS Nav. Path Points

Input: ROS Global Path Poses $P$ // Unevenly spaced
Input: Double $D$ // Distance between arrows
Input: Double $\varnothing$ // Destination circle diameter
Output: Array[x,y,z] $P'$

1. $i \leftarrow |P| - 1$ // From destination to starting point

2. repeat
3. $p \leftarrow P[i], P' \leftarrow P' \cup \{p\}, i' \leftarrow i$
4. try
5. repeat
6. $i' \leftarrow i' - 1$
7. $p' \leftarrow P[i'].pose.position$ // ROS quirk
8. $d \leftarrow \sqrt{(p.x - p'.x)^2 + (p.y - p'.y)^2}$
9. until $d < D$ or ($i = |P| - 1$ and $d < D + \varnothing$)
10. catch Array Out of Bound Exception
11. // Done. $i' < 0$ now. Line 13 breaks the loop.
12. $i \leftarrow i'$
13. until $i > 0$
14. return $P'$ // Evenly spaced

---

**IV. IMPLEMENTATION IN ROS**

Once a projector is mounted onto a robot and pointing towards the front of a robot’s base, the projector’s pose relative to the robot needs to be integrated into the robot’s transform hierarchy using the “static transform publisher” node [39] from the “tf” package [40]. A sample ROS launch file [41] is provided in “tf_publisher.launch”.

Note that to make our account of the implementation beginner-friendly, we added relevant ROS learning resources to references throughout this and following section.

To retrieve the global navigation path, we subscribed to the ROS topic “/move_base/trajectory/ROS/global_plan” exposed by the ROS navigation stack [42], [43]. The topic encloses a Path message [44] from the nav_msgs package [45], including poses discretizing the continuous navigation path.

However, our approach uses a voxelized map, an adaptive Monte Carlo localization [46], and an A*-based path planner [47], and as such, the poses in the path are not evenly spaced. Because an arrow has a physical size, the arrows may overlap and thus obscure each other if the distance between a pair of poses is too close. We thus created Algorithm 1 to solve this problem. The algorithm also allows specifying a navigation destination circle, as shown in Fig. 1 left.

Algorithm 1 iterates each point from the destination to the starting point (Line 1, 2, and 13). The destination point is always the first point in $P'$ (Line 3) because we wanted the destination circle bigger than an arrow. For each point $p$ in the path, it will skip to the point $p'$ when the distance between $p$ and $p'$ is bigger than the desired distance $D$ (Line 9 before “or”). We stop iterating once the index becomes negative and is accessed (Line 7), caught by the try-catch block (Line 4, 10, and 11), or the distance is shorter than $D$ in the beginning (Line 9 after “or”). The algorithm is implemented in C++ and available in the `get_sparse_points` function in “PathProjection.cpp”.

All the logic in this section is coded in the PathProjection ROS node [48] in “path_projection_node.cpp”, “PathProjection.h”, and “PathProjection.cpp”.

**A. Visualizing Arrows in rviz**

Once we get a list of evenly spaced-out points from the ROS navigation path, we create an array of markers [49] as a MarkerArray message [50] in the “visualization_msgs” ROS package [51], and publish it so it can be added to rviz [52].

In rviz, an arrow consists of a shaft and head. Under the hood, as seen from its `arrow_marker.cpp` and `arrow.cpp` source code, rviz uses four parameters for an arrow – shaft length, shaft diameter, head length, and head diameter.

However, the rviz GUI only allows for specifying three parameters, the scale 3D vector through the Marker message [49] interface, partly because the message was designed to be generic to specify other shapes such as cubes, spheres, or cylinders [49]. The generic design turned out to limit customization of the arrow. To tackle this problem, we spent significant time investigating rviz’s source code and directly modified line 108 of `arrow_marker.cpp` [53], in which the four parameters of the arrow’s shaft and head are set, in . Specifically, we made the shaft’s length the same as the arrow’s head’s. A diff file is available in “arrow_marker.cpp.diff”. We hope this can help roboticists to spend more time on their research of interest instead of on arrow customization.

**B. Projecting Arrows in rviz via Projector Lens**

Once the arrow list is published, we can subscribe to the arrow list’s ROS topic in order to render the arrows in rviz (Fig. 2 right). This renders each arrow at position, $(x, y, 0)$, where 0 denotes rendering on the ground plane. In theory,
if we can place a virtual camera in rviz at the pose relative to the robot, exactly the same as where the projector is mounted towards the ground (See Fig. 1). In our approach, any visualization in rviz (e.g., line, point cloud, cube, mesh) can be projected using this technique. As such, this experiment provided empirical evidence for the benefits of our approach. Moreover, we again stress that using speech-based communication only afforded 83.2% accuracy.

Another limitation is projection size. This can be clearly seen from Fig. 1 bottom, the slightly brighter area, which is limited to around $55 \times 98\text{cm}^2$ in a 16 : 9 ratio. This results in a small projection area, preventing projecting arrows farther along a navigation path, which may lead to problems in large spaces like warehouses. One solution is to mount another projector lower, pointing the ground further away, similar to the headlight of a car, so that the projection can be thrown further to cover a long area, although this may artificially inflate arrow size. Fortunately, because we know the distance between arrows and the projector lens, we can shrink the size of arrows proportionally to distance.

VII. Conclusions

We provided arrow projection software and summarized the demonstrated benefits of this software. As shown in Fig. 4 to project arrows, one needs to mount a projector onto a robot with a portable power station, integrate the projector’s pose into the robot’s transform hierarchy, space out the ROS navigation stack’s global path, render the associated arrows in rviz, and finally output to the calibrated projector. Additionally, we also showed the generalization of our technique, i.e., the capability to project any rviz visualizations.


“Ros wiki section for mesh resource rviz marker,” https://wiki.ros.org/rviz/DisplayTypes/Marker#Mesh_Resource_3D_28MESH_29