dynamic spatial augmented reality for lazy people like me who suck at programming

Sasha Wilkinson

January 26, 2020

1 Introduction

You're probably here because you want to look at some sort of example code or something to figure out how to do projection mapping on your robot. Maybe you've dug through the `ur_scooter` repository on Github trying to find the projection mapping code so that you can implement it. In reality, there is *zero code* involved in setting up the projection mapping. It functions on the principle that a projector is just a backwards camera.

2 How it works

A projector is the dual of a camera. In a camera, each ray of light from the world passes through the lens and hits the sensor. In a projector, each ray of light passing through the LCD (or whatever it is) passes through the lens and hits a surface in the world.

Using this principle, we can create a virtual camera in RViz with the same characteristics as the projector lens that will allow us to project visualizations in RViz directly into the world. This virtual camera needs to have the have the exact same lens characteristics (intrinsics) and position (extrinsics) as the projector. The intrinsics are published as a CameraInfo message and the extrinsics are published as a TF transform.

When this image is projected into the world, any visualization on (or near) a surface in RViz will be projected directly back onto its position in the real world. For the scooter, I use this to project segments of pointclouds from the depth cameras to highlight particular objects. It can show segmented objects as well as the grasp radius of the robot as shown here:

Figure 1: Object segmentation

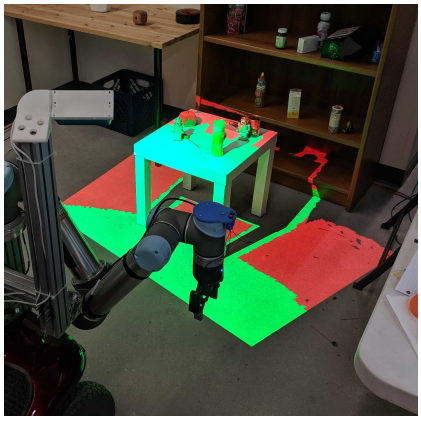


Figure 2: Grasp radius

It can also be used to display markers such as arrows on surfaces by publishing Marker messages that are visualized in RViz where they would be projected in the world.

3 Software setup

The system consists of the following components:

1. CameraInfo publisher with intrinsics of projector
2. TF transform publisher for position of projector
3. RViz instance on robot with visualizers for desired topics
4. `rviz_camera_stream` plugin subscribed to CameraInfo and TF transform
5. ROS image viewer displayed fullscreen on projector, subscribed to Image message from `rviz_camera_stream`

3.1 CameraInfo publisher

After determining the projector's intrinsics (much more on this later), you can place this into a yaml file that represents a CameraInfo message. For example, mine looked like this:

```
header:
  seq: 0
  stamp:
    secs: 0
    nsecs: 0
  frame_id: 'proj_view'
```

```
height: 600
width: 800
distortion_model: plumb_bob

D: [0.0, 0.0, 0.0, 0.0, 0.0]
K: [1250.0, 0.0, 400.0, 0.0, 1250.0, 567.0, 0.0, 0.0, 1.0]
R: [1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0]
P: [1250.0, 0.0, 400, 0.0, 0.0, 1250.0, 300.0, 0.0, 0.0, 0.0, 1.0, 0.0]

binning_x: 0
binning_y: 0
roi:
  x_offset: 0
  y_offset: 0
  height: 0
  width: 0
  do_rectify: false
```

Then, you can publish this message using `rostopic`. The following creates a publisher in roslaunch:

```
<launch>
        <node name="camera_info" pkg="rostopic" type="rostopic"
                args="pub -l -f $(find
  ↪  scooter_ui_proj_map)/src/proj.yaml /proj_view/camera_info
  ↪  sensor_msgs/CameraInfo" />
</launch>
```

3.2 TF transform publisher

Next, publish a frame for the position of the focal point of the projector. This frame should be the same as the `frame_id` value in your CameraInfo message. In my case the projector does not move w.r.t. the robot so a static transform is okay.

```
<launch>
        <node pkg="tf" type="static_transform_publisher"
  ↪  name="proj_view" args="0 0 0 0 0 0 base_link proj_view 10" />
</launch>
```

If the projector is on a gimbal, then it will be necessary to publish a frame that follows the motion of the gimbal. It is worth noting that a small amount of translational error is not as bad as a small amount of rotational error, since rotational error accumulates with distance. I found that even though I did not know where exactly the focal point of the lens was in space, I got a perfectly acceptable result by approximating the translation and then tweaking the rotation.

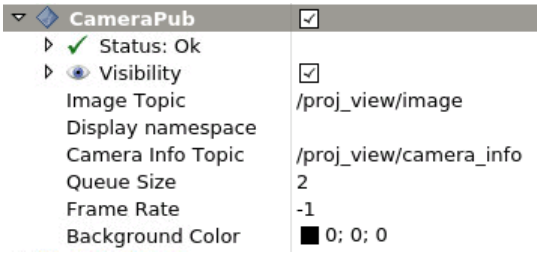
3.3 RViz instance

You will need to set up an RViz instance that runs on the robot that publishes an Image topic for the projector. First, install `rviz_camera_stream` from the following link:

https://github.com/lucasw/rviz_camera_stream

This plugin creates a virtual camera in the RViz world that simulates the lens of the projector with the intrinsics provided in the CameraInfo message. It then publishes

Configure it with the CameraInfo topic published previously, and give it a topic to publish the output image on.



You can do all this configuration through the GUI and then save it as a configuration file. Then, in your roslaunch file, you can run RViz with a specific config as such:

```
<launch>

        <arg name="rviz_config" default="$(find
↪  ur_scooter)/rviz/ur_scooter_projector.rviz"/>

        <arg name="limited" default="false"/>

        <node name="$(anon rviz)" launch-prefix="" pkg="rviz"
↪  type="rviz" respawn="false"
        args="-d $(arg rviz_config)" output="screen" />

</launch>
```

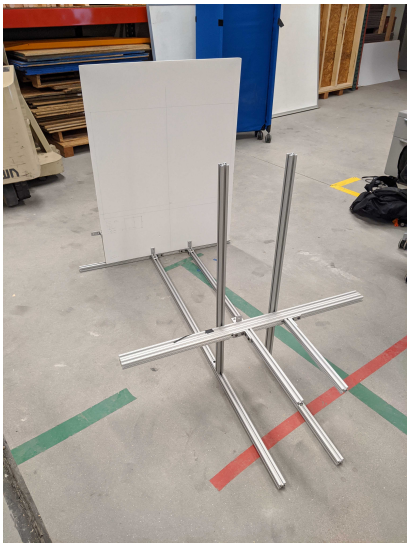
Now any visualizations you set up in RViz such as pointclouds and markers will show up in the Image message being published by the plugin.

3.4 Image viewer

Set up a fullscreen image viewer that views the output Image message from the RViz plugin and display it through the projector. It is a surprisingly difficult

problem to display a ROS Image message in fullscreen. I have a package that makes the window truly fullscreen but I don't have it up on Github right now and I need to go do some homework soon so I'm trying to finish this quickly oh geez oh heck oh man uhhhh come back later I'll finish this guide I promise

4 Method for Determining Projector Intrinsics



I built a horrendous 80/20 contraption that looks like this to calculate the lens intrinsics. It's in a corner near the scooter zone at the NERVE Center if you want to use it. It is very "square" and "precise."

(everything after this point is copied from a thing I wrote a long time ago, good luck.)

In order to determine the intrinsics of the projector lens, the projector was mounted a fixed distance away from a posterboard and marks were made at each of the four corners of the image, the center of the image, and the point where the optical axis intersects the image plane. The projector was then modeled using a pinhole camera model and the intrinsics were placed into an intrinsic camera matrix K:

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{1}$$

We can determine the focal length f of the projector in pixels using the projective equation. Where x is the width of projected the image in pixels, X is the distance from the optical axis to the edge of the image on the image plane, and Z is the distance from the projector to the image plane,

$$f = x\frac{X}{Z} \tag{2}$$

The principal point (c_x, c_y) was then calculated in pixels:

$$\begin{aligned} c_x &= W_{\text{PIX}} * \frac{x}{W} \\ c_y &= H_{\text{PIX}} * \frac{y}{H} \end{aligned} \tag{3}$$

where W and H are the width and height of the image respectively, and x and y are the distances from the optical axis to the edge of the image in the x and y directions respectively.

Given these values for the focal length and principal point, the camera matrix K can be constructed.

5 Conclusion

Good luck. Message me on Slack if you have any questions. I'll try to fix this guide later.