Lab 5: EKF Localization – Vision

In this and the next lab we will use the camera and colored landmarks to implement the localization based on the Extended Kalman Filter. Along the way, you will learn how to use the transformation tree to reconcile the frames of reference. In this lab, we will focus on vision processing and implement the measurement model. In the next lab we will implement the filter.

We already know how to track the robot based on odometry only, but we also know that the odometry drifts over time. As an analogy, imagine that you are hiking on a trail. In absence of any visible trail marks, we rely on our inner feeling to estimate how far we walked since the last time we saw a trail mark and that estimate becomes less precise over time. Spotting a trail mark gives us reassurance about the location and corrects the accumulated error.

Principle of Operation

The world we will use for this lab is called Turtlebot 3 Among Landmarks (it is a custom world available in the class Git repository). To start it, enter your Distrobox container and type this:

```
ros2 launch prob_rob_labs turtlebot3_among_landmarks_launch.py
```

When Gazebo GUI starts, zoom out the view and notice four colored cylinders. These are our landmarks. Each landmark has a different color and we will take advantage of that to identify it. We will use a camera projection model and known dimensions of the landmark to estimate the distance and bearing. So, in our EKF localization correspondences are known.

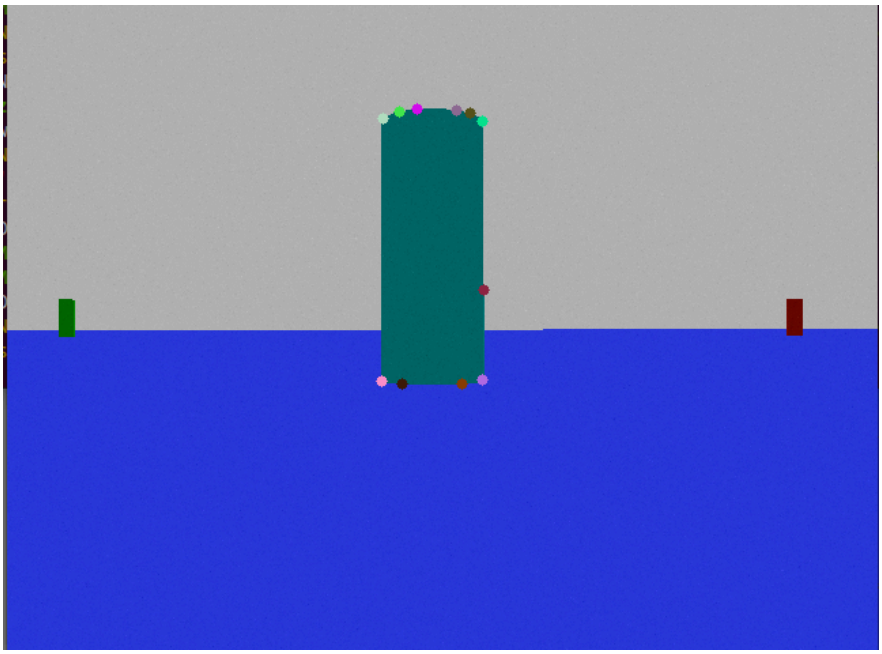
As in any Kalman filter, there are two steps, prediction and measurement. For prediction, we can use the odometry module from the previous lab, or we can incorporate the filter's own motion model based on the robot's instantaneous velocity. We will do the latter and use the arc-motion model as presented in the textbook. The velocity input can either come from the robot command, or from odometry topic. We will do the latter.

The algorithm presented in Table 7.2 of our textbook assumes that the sensor measures the distance and bearing to the landmark along with the associated variance. The textbook does not explain how such a measurement is accomplished nor does it assume a specific sensor nor does it state how to identify correspondences. In our system, the sensor is the camera and landmarks are identified by their unique color. The objective of this lab is to derive and implement the image processing algorithm that produces the distance and bearing measurements.

Vision Pipeline

In our world, the landmark cylinders have a radius of 0.1m and height of 0.5m and are placed upright, so the 2D projection to a monocular camera is approximately a rectangle of the same proportions regardless of the angle from which the landmark is observed. The size of the rectangle is proportional to the distance from which the landmark is observed, so if the size and camera intrinsics are known, it is possible to estimate the distance using a monocular lens only.

The vision pipeline we will use is a color filter that will extract only the color of interest for a given landmark and the feature detector that will identify the points along the perimeter of the landmark in sight. There will be up to 16 points to ensure reliable detection. An example view is shown below:



To see the view in your simulation, start the image viewer node as follows:

```
ros2 run image_view image_view image:=/vision_cyan/image_raw
```

To see the identified corner-points, use the topic tool as follows:

```
ros2 topic echo /vision_cyan/corners
```

Drive the robot around (use the joystick) and notice that whenever the robot faces the landmark, the corner points start appearing on the topic. Repeat the experiment for the other landmarks. The colors used in this simulation are: red, green, magenta, yellow, and cyan. Barring the deformation when the robot is very close to the landmark, the landmark appears as a rectangle in the camera view. The objective of the vision pipeline is to estimate the distance and bearing

angle between the camera and the landmark. We will use the identified cornerpoints to first estimate the height along the y-axis and the x-position of the vertical symmetry axis in the pixel space. Next, we use camera projection model to estimate the distance and bearing to the landmark.

Assignment 1: Look at the image and describe the algorithm you would use to estimate the height and position of the vertical symmetry axis of the landmark in the pixel space. Write a function that implements your algorithm and submit the code. You can test it either by subscribing to the topic or by feeding a few example corner-points.

Projection Model

The `/camera/camera_info` topic of type `CameraInfo` contains the projection model. Of interest is the P-matrix which is defined as:

$$\mathbf{P} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix},$$

where (f_x, f_y) are the focal lengths of the camera and (c_x, c_y) is the principal point. Note that there are several other camera calibration parameters that we are not using here. One of them is particularly important in real-world cameras and it is called the distortion model. The distortion model captures the non-linear image deformations that make the image look pillow-shaped instead of a straight rectangle. In this lab we will assume that such deformations don't exist and that the transformation from 3D camera optical frame to 2D pixel space is a pure projection. For more details, please see the definition of the `CameraInfo` message in ROS.

The camera optical frame is oriented such that the z-axis points out of the camera towards the object in sight, y-axis points down, and x-axis points from left to right from the when looking in the direction of the z-axis. The frame, thus, forms a right-hand coordinate system of a 3D space in front of the robot. Let p_0 be a point in the camera frame. This point transforms to the pixel space as follows:

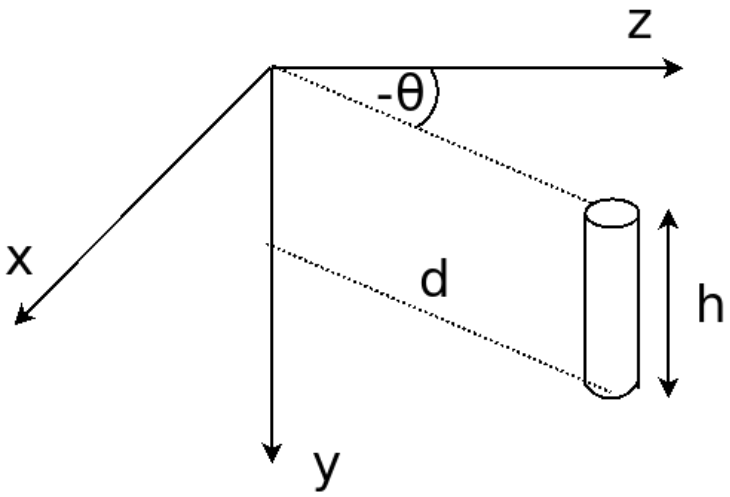
$$\begin{bmatrix} u_0 \\ v_0 \\ w_0 \end{bmatrix} = \mathbf{P} p_0 = \begin{bmatrix} f_x x_0 + c_x z_0 \\ f_y y_0 + c_y z_0 \\ z_0 \end{bmatrix},$$

$$p_{0p} = \begin{bmatrix} \frac{u_0}{w_0} \\ \frac{v_0}{w_0} \end{bmatrix} = \begin{bmatrix} \frac{f_x x_0}{z_0} + c_x \\ \frac{f_y y_0}{z_0} + c_y \end{bmatrix}.$$

The components of the vector p_{0p} are the pixel coordinates of the projected points.

Distance and Bearing Estimation

Referring to the figure, suppose that we have identified two points on the top and bottom of the cylinder along the vertical axis of symmetry. Let their coordinates be (x_{p0}, y_{p0}) and (x_{p1}, y_{p1}) which are the projections of points (x_0, y_0, z_0) and (x_1, y_1, z_1) in 3D space. By construction $x_0 = x_1$ and $x_{p1} = x_{p0} = x_p$. The cylinder height in pixel space is $\Delta y_p = y_{p1} - y_{p0}$ and the cylinder height in 3D space is $h = y_1 - y_0$.



The objective is to estimate the bearing angle θ and the distance d. Note that the angle shown in the figure is the negative of the bearing angle because the robot frame is oriented such that the x-axis points forward and z-axis points upward (the figure shows the optical frame).

From the projection model, we have,

$$\Delta y_p = \frac{f_y y_1}{z_1} + c_y - \frac{f_y y_0}{z_0} - c_y,$$

$$\Delta y_p = \frac{f_y y_1}{d\cos\theta} - \frac{f_y y_0}{d\cos\theta},$$

$$\Delta y_p = \frac{f_y}{d}(y_1 - y_2) = h\frac{f_y}{d\cos\theta}.$$

The last equation sets the relationship between the measured height in pixel space, and the distance-and-bearing measurement. The camera focal length and the known height of the cylinder are the parameters. If we know the bearing, we can determine the distance as

$$d = h\frac{f_y}{\Delta y_p \cos\theta}.$$

It is also possible to use the information from the measured width and the known radius, but the geometry gets more complicated, so we will only use the measured height.

Referring back to the figure, we can use the projection model to set the relationship between the x_p and the bearing angle:

$$x_p = \frac{f_x x_0 + c_x z_0}{z_0},$$
$$x_p = f_x \tan(-\theta) + c_x,$$
$$\theta = \arctan \frac{c_x - x_p}{f_x}$$

We now have mathematical foundations to implement the measurement branch of the Kalman filter. In this lab, we focus on calculating the distance-and-bearing measurements from pixels. We start from core functions.

Assignment 2: Write a ROS node that subscribes to the `/camera/camera_info` topic and the corner-points topic for a landmark of your choice and calculates the distance and bearing to the landmark. Your node will need to know the landmark height to use it in calculations and the landmark color to know which topic to subscribe to. Both should be the node parameters. Publish the calculated distance and bearing. Submit the code.

Bring up the node and drive the robot around pointing it to the landmark from various distances. Make sure the node handles the case when the landmark goes out of sight (it should stop calculating and publishing the measurement and it may not crash). Make sense out of the values you are getting. You can use the `rqt_plot` tool to visualize the measurement.

Place the robot a few meters away from the landmark and make it face it directly. Next, start rotating the robot in place and watch the distance measurement plot. The measurement should remain more or less constant (which is expected), except when the landmark reaches the edge of the camera's field of view. The reason this happens is that the cornerpoints from one side of the landmark are missing, which may result in incorrect calculation of the symmetry axis, which further skews the bearing measurement, followed by skewing the distance measurement. If you notice this problem, extend the code to drop the offending measurement.

Error Characterization

To complete the measurement branch of our Kalman filter, we need a way to estimate the measurement covariance. This process is more difficult to capture analytically because the error comes from a wide variety of sources. Empirical methods are often needed. In this lab we will make a simplifying assumption that the bearing and distance error are uncorrelated and that each is a memoryless function of the distance and bearing (either measured or calculated from prior). In other words we need to determine the following:

$$\sigma_d^2 = f_d(d_{\text{measured}}, \theta_{\text{measured}}, d_{\text{prior}}, \theta_{\text{prior}}),$$
$$\sigma_{theta}^2 = f_\theta(d_{\text{measured}}, \theta_{\text{measured}}, d_{\text{prior}}, \theta_{\text{prior}}).$$

Assignment 3: Write the node that subscribes to the ground truth topic (use the ground-truth publisher you wrote in previous labs), calculates the true bearing and distance to the landmark (don't forget to transform the pose from the robot frame to the camera frame) and publishes the measurement error for both distance and bearing. Log (or publish) the measured distance and bearing alongside with the calculated error (make sure it is easy to capture and parse the data). Submit the code. Drive the robot around and plot the distance and bearing error using rqt_plot tool to get the feeling of the error magnitude. Submit the plots.

We now have the tool that we can use to collect the data and estimate the measurement variance. Each record consisting of measured distance, measured bearing, distance error, and distance bearing is a datapoint that we are trying to fit in a simple regression model.

Assignment 4: Drive the robot systematically to cover all distances between 0 and 10m and the full range of bearing errors. Record the measurements and errors and come up with a simple model that can produce the variance for a given measurement and/or prior. Keep it simple, even a trivial "eyeballed" model is fine for the purpose of this lab, as long as your model is supported by the data. Note that the error variance may not depend on all four argument dimensions. Submit the code that estimates the measurement variance along with any plots or data samples that support your choice. Describe in a few sentences how you have come up with the model.