Literature Review
Lab coursDeveloping a Resource Planning Platform for Autonomous Vehicles

I. LOCALIZATION UNIT

For the localization Unit, we chose 2 iterative estimation algorithms which are based on the history data of the vehicle. Those techniques can be useful in a context where some sensors of the cars cant operate. In a tunnel for example, the GPS signal gets weaker. Therefore, in this context the vehicle position estimation becomes difficult.

A. Application 1: Kalman Filter technique

Kalman filter is the most commonly used data optimal estimation algorithm in the field of autonomous driving. People's first impression of it is often its complicated linear algebra expression. According to Wikipedia, Kalman Filter uses a series of measured values (including statistical noise and other errors) observed over time to generate estimates of unknown variables. The estimated values are often estimated by the joint probability distribution of the variables in a time range, the result is therefore more accurate than the result based only on a single measurement.

It sounds a bit complicated, in a nutshell, the Kalman filter is an optimal estimation algorithm, which can estimate the state of the dynamic system from a series of incomplete and noise-containing measurements.

The Kalman Filter technique goes through two cycles:

Prediction Phase and Measurement update.

1) Initialization: Before we deep into prediction and measurement update phases, we first have to initialize our model.

Assuming a car is moving at a constant speed to the right on the road , we installed a sensor measuring the distance and speed of the car on the left. The sensor measures the position in s and speed v of the car every 1 second. We use the vector x_t to represent the current state of the car. This vector is also the final output result , which is called the state vector :

$$x_t = \begin{bmatrix} s_t \\ v_t \end{bmatrix}$$

Due to the measurement error, the sensor cannot directly obtain the true value of the position of the trolley, and can only obtain an approximate value near the true value. It can be assumed that the measured value obeys a Gaussian distribution near the true value.

Since it is the first measurement and there is no historical information of the car, we believe that the state x of the car at 1 second is equal to the measured value z, which is expressed as follows:

$$x_1 = \begin{bmatrix} s_1 \\ v_1 \end{bmatrix}$$

, where the 1 in the formula means the first second.

2) Prediction phase: Prediction is a very important step in the Kalman filter. This step is equivalent to using historical information to predict the future position. According to the position and speed of the car in the first second, we can infer that the position of the car in the second second.

Predict according to the state of the car in the first second, and get the predicted state x_{pre}:

$$x_{pre} = \begin{bmatrix} s_1 + v_1 \\ v_1 \end{bmatrix}$$

Among them, pre is the abbreviation of prediction; the time interval is 1 second, so the predicted position is distance + speed*1; because the car is moving at a constant speed, the speed remains unchanged.

3) Measurement update phase: At the second second, the sensor made an observation of the position of the car. We believe that the observation value of the car at the second second is $z2$. The observation result at the second second is represented by a vector:

$$z_2 = \begin{bmatrix} s_2 \\ v_2 \end{bmatrix}$$

To facilitate understanding, the state vector of the second second can be written as:

$$x_2 = w_1 * x_{pre} + w_2 * z_2$$

, where w_1 is the weight of the prediction result, and w_2 is the weight of the observation result. The calculation of the two weights is based on the uncertainty of the prediction result and the observation result. This uncertainty is the size of the variance in the Gaussian distribution. The larger the variance, the wider the waveform distribution, and the higher the uncertainty. The weight given will be lower.

4) Prediction and measurement: The initialization in the first second and the prediction and observation in the second second realize a cycle of Kalman filtering.

Similarly, we make a prediction for the third second based on the state vector of the second second, and then weight it with the observation result of the third second to obtain the state vector of the third second; then do the fourth based on the state vector of the third second The second prediction is weighted with the observation result of the fourth second, and the state vector of the fourth second is obtained. With this reciprocation, a real Kalman filter is realized.

B. Application 2: Range Flow-based 2D Odometry

This method relies only on laser scans to estimate the planar motion of a vehicle. Compared to traditional approaches, the RF2O method does not search for correspondences but performs dense scan alignment based on the scan gradients, in the fashion of dense 3D visual odometry. For each scanned point, a flow constraint equation is formulated as a function of the velocity of the sensor, assuming that the environment is static. The resulting geometric constraints is then minimized to obtain the motion estimate. The approach could be divided into 3 steps:

1) Lidar velocity and 2D range flow calculation: Let $\mathrm{R}(\mathrm{t}, \alpha)$ be a range scan where t is the time and $\alpha \in [0,\ \mathrm{N}) \subset \mathrm{R}$ is the scan coordinate. Taking two 2 scan pairs in two consecutive moments we can derive the geometric consistency formula described as :

$$\mathrm{R}(\mathrm{t}+\Delta\mathrm{t}, \mathrm{a}+\Delta\alpha) = \mathrm{R}(\mathrm{t}, \alpha) + \partial R \partial \mathrm{t}(\mathrm{t}, \mathrm{a})\Delta\mathrm{t} + \partial R \partial \mathrm{a}(\mathrm{t}, \alpha)\Delta\alpha + \mathrm{O}\left(\Delta\mathrm{t}2, \Delta\alpha_2\right) \tag{1}$$

Where $\Delta\mathrm{t}$ is the time lapse between consecutive scans and $\Delta\alpha$ represents the change in the scan coordinate of the point considered. The range flow constraint is then converted into the sensor velocity constraint described as

$$(\cos\theta + \mathrm{R}_\alpha \mathrm{K}_\alpha \sin\theta r)v_{x,s} + (\sin\theta - \mathrm{R}_\alpha \mathrm{K}_\alpha \cos\theta r)v_{y,s} + (x\sin\theta - y\cos\theta - \mathrm{R}_\alpha \mathrm{K}_\alpha)\omega s + Rt = 0 \tag{2}$$

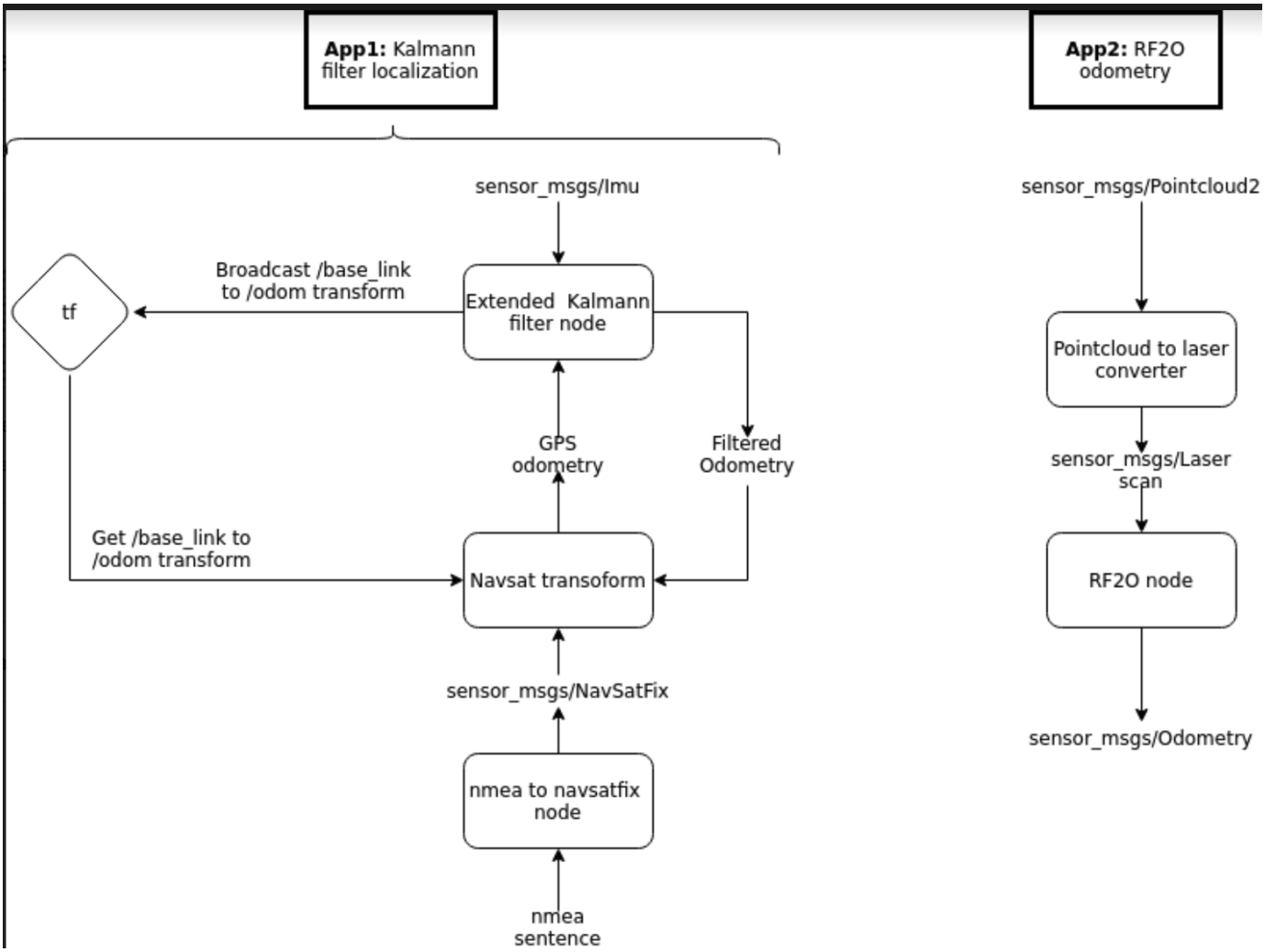
Where $v_{x,s}, v_{y,s}, \omega_s$ are 2D twist of the sensor.

2) Velocity estimation: Due to noise of the range measurements, the formula (2) is inaccurate, therefore we estimate the geometric residual for a given twist as

$$\rho(\xi) = \mathrm{R}_t + (x\sin\theta - y\cos\theta - \mathrm{R}_\alpha \mathrm{K}_\alpha)\omega + (\cos\theta + \mathrm{R}_\alpha \mathrm{K}_\alpha \sin\theta r)vx + (\sin\theta - \mathrm{R}_\alpha \mathrm{K}_\alpha \cos\theta r)v_y \tag{3}$$

To obtain an accurate motion estimate of the sensor, we use Cauchy M-estimator to minimise the the geometric errors. This process is not efficient against non linear variations of range functions ,therefore we follow a pre-weighting strategy for downweighting the residuals of these points.

3) Coarse-to-fine schema and scan warping: The final step consists in filtering consecutive range scans using 2 gaussian masks. At every transition from a scan to another the second scan is warped against the first one. The obtained warped scan indicates if the estimated velocity is converging to the real velocity which allows us to use the approximation (1)



II. PERCEPTION UNIT

Perception problems in robotics are problems that lie in the robot's necessity to know the surrounding environment. For a robot to know if there is a danger or a safe passage, it must sense and reason about its measurements. Here we use two ways to solve the perception problem.

The first one uses a Velodyne as the primary sensor, a non-visual sensor but has 360 degrees of view and can see all the objects around the car. It is a technique where we rely on very noisy sensor data.

The second technique uses a segmentation camera as the primary sensor. Even though there isn't any segmentation camera in real life, this module is treated as an abstraction of some neural network that publishes the segmented images. The segmentation camera publishes an image where each object has a pixel color related to their type, e.g., people have red pixels and cars blue pixels.

A. Object Classification

In a nutshell, for object detection we first need to preprocess the image so that we can count each object properly. Then, we use connected components to count and label the image. Finally we search in the images for each object stats including centroid in pixels, height and width, then we pass the result to object detector aggregator.

1) Preprocessing: change the color space: Here we first change the color space of the image from RGB to HSV. RGB color space is the most basic, most commonly used, hardware-oriented color space in image processing, and it is relatively easy to understand. The RGB color space uses a linear combination of three color components to represent colors. Any color is related to these three components, and these three components are highly correlated. Therefore, it is not intuitive to continuously transform colors. Adjustments need to change these three components. Besides, human eyes have different sensitivity to these three color components: In monochrome, the human eye is the least sensitive to red and the most sensitive to blue, so the RGB color space is a color space with poor uniformity. If the similarity of colors is directly measured by Euclidean distance, the result will have a large deviation from human vision. For a certain color, it is difficult for us to infer a more accurate

three-component value to represent. Therefore, the RGB color space is suitable for display systems, but not suitable for image processing.

For the above reasons, we transform the RGB space to HSV space, whic is used more in image processing because it is closer to people's experience of color perception than RGB and directly reflects the hue, vividness and brightness of the color, which is convenient for color contrast.

In the HSV color space, it is easier to track objects of a certain color than BGR, and is often used to segment objects of a specified color.

The way HSV expresses color images consists of three parts: Hue, Saturation (color purity), and Value (lightness).

The hue H is measured by angle and the value range is 0360, starting from red and counting in a counterclockwise direction. Red is 0, green is 120, and blue is 240. Their complementary colors are: yellow is 60, cyan is 180, and purple is 300;

The saturation S indicates how close the color is to the spectral color. A color can be seen as the result of mixing a certain spectral color with white. Among them, the greater the proportion of the spectral color, the higher the degree of color close to the spectral color, and the higher the color saturation. With high saturation, the color is deep and bright. The white light component of the spectral color is 0, and the saturation is the highest. Usually the value ranges from 0 to 100. The larger the value, the more saturated the color.

2) Preprocessing: image thresholding: Thresholding is a technique where we apply a threshold in each pixel. We specify different thresholding for different object types. Then we do image thresholding for each image.

More specifically, if the pixel's value is in the range interval the pixel value goes to white otherwise the pixel goes to black. E.g., In our application we have the range values where the car's color is in HSV space, so pass these range values to a function that applies the thresholding in the image. As a result, we get an image with only white and black pixels, the white pixels are the cars and black pixels are something else. We apply the thresholding to each object.

We compare the HSV values in each pixel if they are in our range. E.g. I know that every car will appear with blue pixels and these pixels are in the range for Hue=[95; 130], Saturation=[140; 255], and Value=[0; 220]. So, if the pixel's HSV values are in these ranges it's because they are from a car.

3) Preprocessing: Morphological operations: Binary images (consists of pixels that can have one of exactly two colors, usually black and white) may contain numerous imperfections. In particular, noises and textures can be distorted when the binary regions produced by simple threshold. Morphological Operations in Image Processing pursues the goals of removing these imperfections by accounting for the form and structure of the image.

We did two operations, erosion followed by dilation. Just like an opening that also is an erosion followed by dilation, each operation is performed with a different kernel. The selected kernels were selected among several for better performance. The erosion step has the effect of removing undesired white noise. The dilation will grow each object's area then the bounding box will have the whole object inside. [1]

4) Counting objects: Connected components labeling: Connected-component labeling is used in computer vision to detect connected regions in binary digital images.

The essence of the algorithm is to scan each pixel of an image, divide the pixels with the same value into the same group, and finally get all the connected components of the pixels in the image. The scanning method can be from top to bottom and from left to right. For an image with N pixels, the maximum number of connected components is N/2. Scanning is based on each pixel unit. For binary images, the set of connected components can be V=1 or V=0, depending on the difference between the foreground and background colors. For grayscale images, the pixel set of the connected image components may be a series of grayscale values between 0 and 255.

The algorithm flow is as follows: 1) First scan the eight neighborhood pixel values adjacent to the current pixel, and find connected pixels to be marked. 2) After scanning all pixels completely, merge all connected components according to the marks.

5) Search for object area: The connected components result image is an image where each pixel has the value of its object label. So, for each label, search for the leftmost pixel, the rightmost pixel, topmost pixel, and bottommost pixel with the label that we are searching. From these 4 pixels, we can calculate the object's centroid, width, and height.

Finally we will push the result as a list to object detector aggregator to do further process.

B. Object Detection

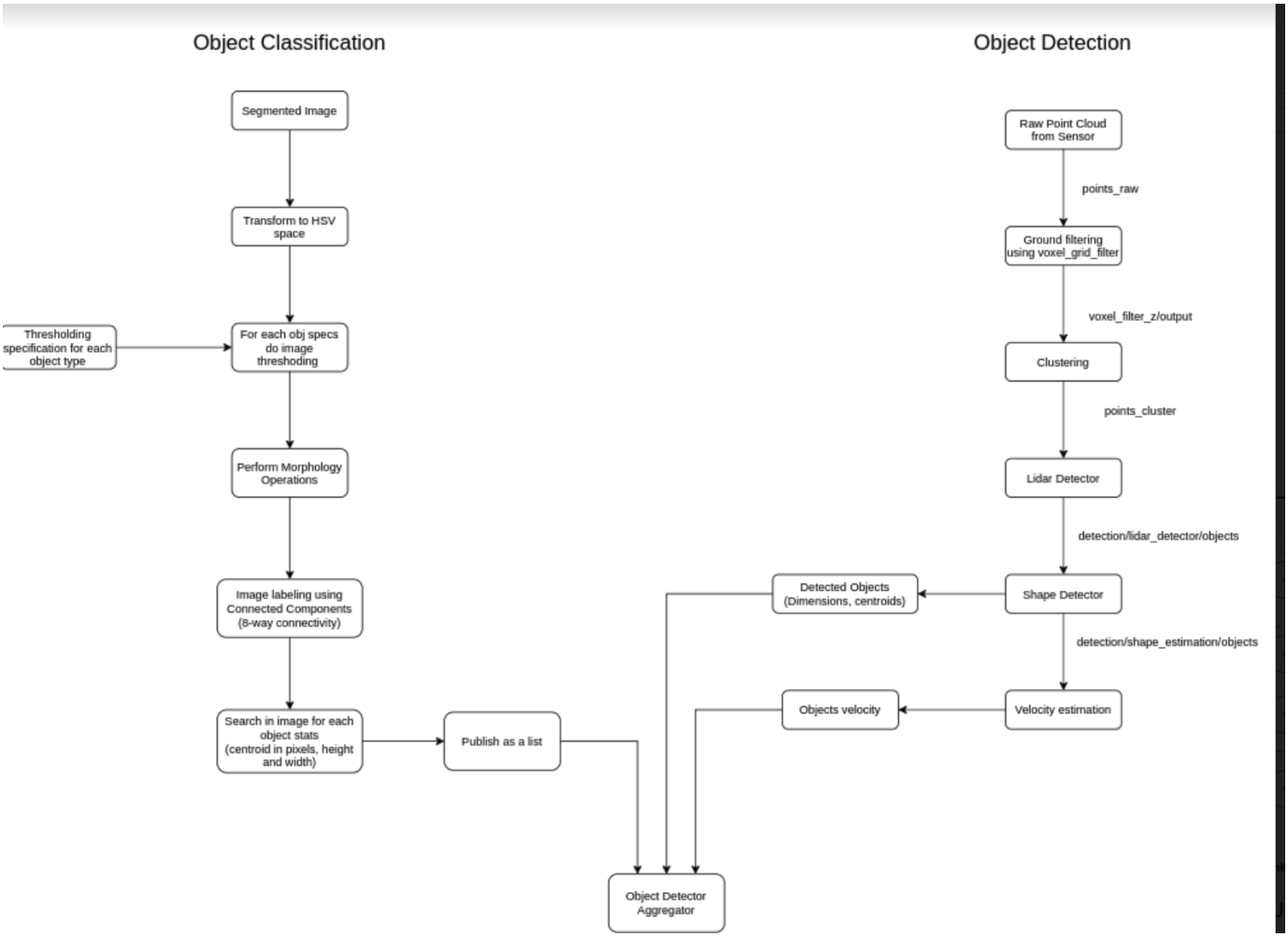
In a nutshell, to detect the objects that are around the car, we should first remove the ground. Then, we cluster the point cloud, and each cluster will be an object. Calculate the object's dimensions and its velocities.

Our input are raw pointer cloud from sensor. To remove noise, we first remove the ground using voxel grid filter and then cluster the resulting point cloud using Lidar detector. The output of the lidar detector is a clustered point cloud and the centroid of each cluster. Then we pass the results to shape detector, which calculates a bounding box that surrounds the object and publishes the result for all objects. Then the velocity estimator calculates the position derivate for each object and publishes its results (to object detector aggregator).

The object detect aggregator listens to all output topics and prints in the screen the speed, type, and direction of moving objects.

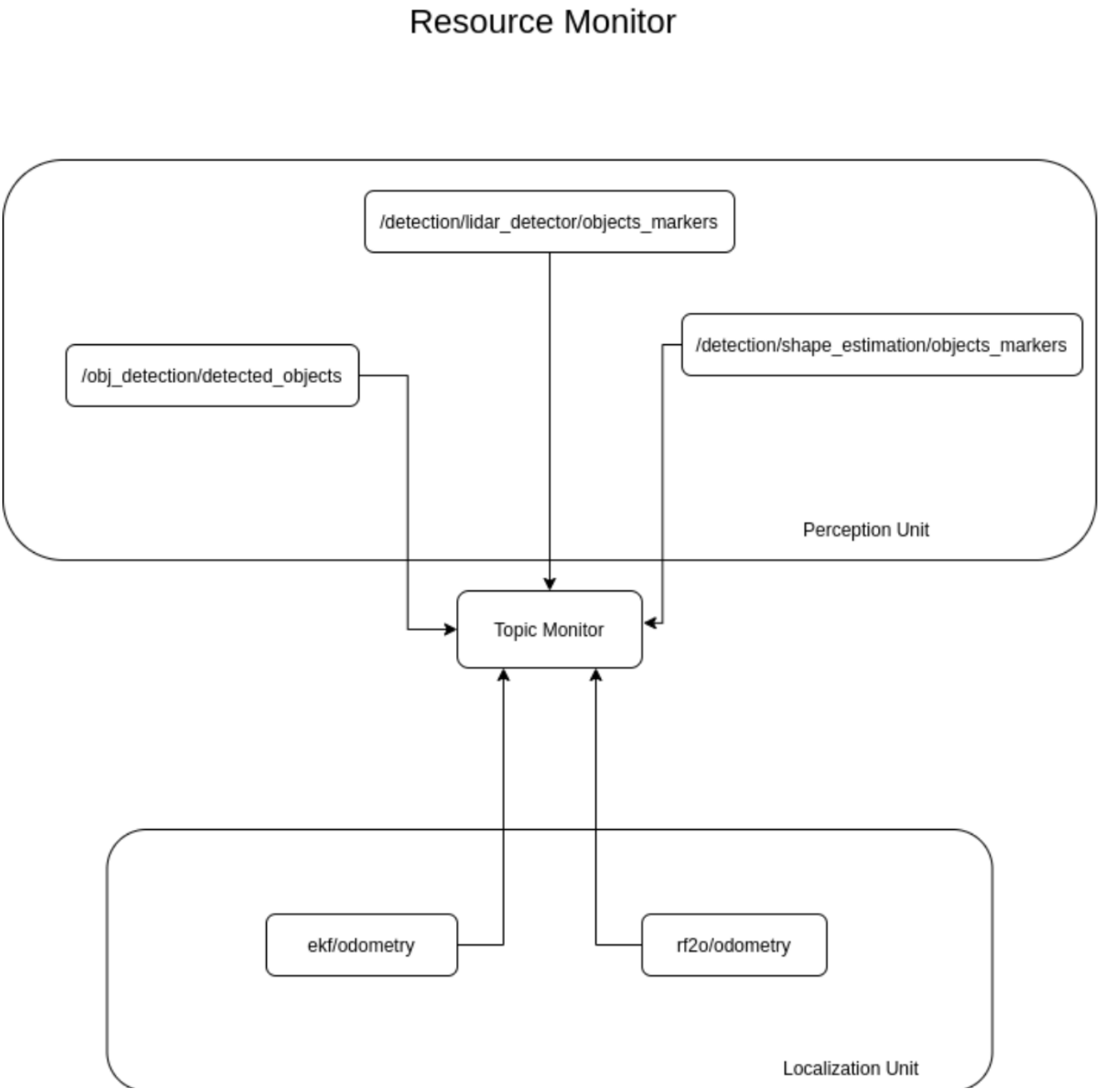
C. Conclusion

Cameras have better noise values, but they are challenging to process, and they lack field of view. The object classification is useful to know how many objects are in the scene and each type they are. It can also show where the object is in the camera frame, but it's challenging to know where each object is in real life without the depth. Non-visual sensors usually have a much larger field, and their data are generally much more comfortable to process than any commercial camera, but they are very noisy. The object detection is useful for knowing where each object is in real life and its real-life dimensions, but it is noisy because of the sensor.



III. RESOURCE PLANNER

There are two nodes in the resource unit. The first one subscribes to every output topic from the localization and perception unit and calculates the received msg latency. The messages are published with their published timestamp, so getting the time that arrives a message can calculate the latency as the difference of received time and published time. The latency of each topic is then published in another topic. The second node also subscribes to the topics and calculates some statistics like publishing frequency and standard deviation. The second node also plots the frequency of the topics and their latency. When some topic violates its designed frequency (outside of its frequency range), the node draws a red line and sends a warning message to the unit node.



REFERENCES

[1] https://docs.opencv.org/3.4/d3/dbe/tutorial$_{o}pening_{c}losing_{h}ats.html$