Proposed solution to RoboCupRescue EX3

Elorm Avevor

University of Cambridge
ea520@cam.ac.uk

October 22, 2022

1 The problem

Detect and **locate** different objects of interest in a maze. This capability has to be an autonomous background service for teleop or autonomous robots. Some items will be in the maze multiple times (e.g. hazmat signs), others rather rare (e.g. door). **Show the detection live in the GUI** and identify the object by name, e.g. "Oxidizer". Points will be adjusted accordingly (for example, hazmats might get 1 point, doors 5 points). **The points are doubled if the identified object is marked at the correct location in the map** (within 1 m of the true position). Points will be deducted for wrongly detected or located objects. The identified objects in the map have to be numbered and colored according to the standard (see Mapping Capabilities and 2D Map Format). There has to be a corresponding text file with the object number, the object type, the time they were found and location.

	A	B	C	D	E	F	G	H	I
1	pois								
2	1.2								
3	CUR								
4	England								
5	2022-07-08								
6	09:19:11								
7	Prelim1								
8	id	time	text	x	y	z	robot	mode	type
9	0	09:19:14	Chart M	0.548	-0.392	1.241	Arbie	T	QRCode
10	1	09:19:18	Chart N	0.058	0.447	1.247	Arbie	T	QRCode
11	2	09:19:29	infectious-substance	-0.007	0.478	1.289	Arbie	T	Hazmat
12	3	09:20:10	Chart H	-1.565	1.183	1.248	Arbie	T	QRCode
13	4	09:20:12	Chart C	-0.649	0.853	1.248	Arbie	T	QRCode
14	5	09:20:20	explosive	-0.711	0.902	1.267	Arbie	T	Hazmat
15	6	09:20:27	Chart G	-2.479	1.763	1.293	Arbie	T	QRCode
16	7	09:20:30	Chart J	-3.401	2.254	1.293	Arbie	T	QRCode
17	8	09:20:33	flammable-solid	-3.504	2.335	1.322	Arbie	T	Hazmat
18	9	09:20:37	Chart K	-4.313	1.309	1.279	Arbie	T	QRCode
19	10	09:20:41	Chart O	-4.585	-0.435	1.264	Arbie	T	QRCode
20	11	09:20:45	Chart B	-3.675	-0.845	1.254	Arbie	T	QRCode
21	12	09:20:46	explosive	-3.599	-0.899	1.277	Arbie	T	Hazmat
22	13	09:20:52	Chart E	-2.884	-1.24	1.233	Arbie	T	QRCode
23	14	09:20:56	Chart F	-2.313	-1.514	1.235	Arbie	T	QRCode
24	15	09:21:00	Chart L	-1.521	-1.893	1.232	Arbie	T	QRCode
25	16	09:21:03	Chart I	-0.238	-1.922	1.277	Arbie	T	QRCode
26	17	09:21:12	spontaneously-combustible	-0.214	-1.823	1.297	Arbie	T	Hazmat
27	18	09:21:35		-4.603	-0.41	-0.015	Arbie	T	fire extinguisher
28	19	09:22:48	inhalation-hazard	-4.588	-0.274	1.175	Arbie	T	Hazmat

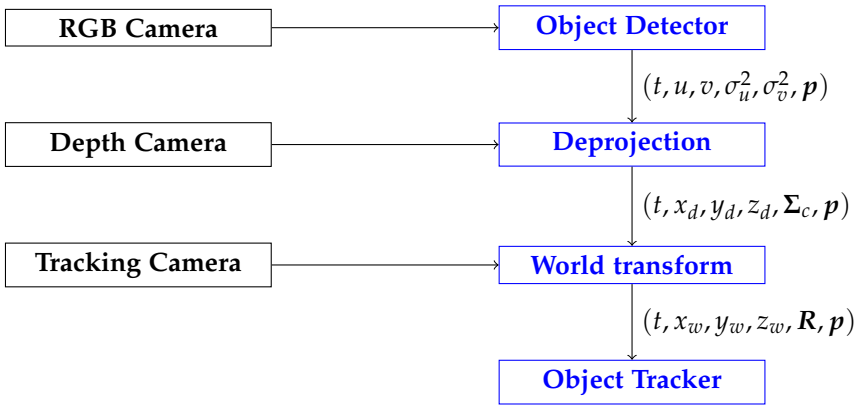
Figure 1: *An example of one such file in CSV format*

Objects for EXP 3 to be recognized in 2022:

- Fire Extinguisher
- Door
- Valves
- Baby dolls
- Hazmat Signs (see list on the RRL web site)
- QR codes (read the QR code)
- Exit signs; green (according to the standard of the country the competition is held in; 2019 for example: Austrailian exit sign)

- Fire Extinguisher signs; red (according to the standard of the country the competition is held in; 2019 for example Austrailian fire extinguisher sign),
- Heat source

2 Proposed solution



t	time
u, v	pixel coordinates
$\boldsymbol{p}$	estimate of Pr(class | image) for all classes
$\sigma^2, \boldsymbol{\Sigma}_c, \boldsymbol{R}$	variances and covariance
c	relative to depth camera
w	relative to the world
Black node	sensor
Blue node	software

2.1 The object detector

2.1.1 Training

- Find a large dataset of images (preferably labelled)
- Add images to the dataset if necessary
- Label the images/verify the images are labelled accurately
- Train using YOLOv5n/YOLOv7-tiny (640 x 480)

2.1.2 Deploying

- Export the model as ONNX.
- Convert the ONNX model to OpenVINO using the OpenVINO model optimiser. Try FP16.
- Use OpenCV C++ to read the model efficiently using the iGPU.
- Shouldn't need 30 FPS – frames 33ms apart are too similar to be useful (at least optically).

2.1.3 Testing

TO DO

2.1.4 Estimating the error in the centre

Use the validation dataset. Plot the squared error in the centre as a function of the width and height of the bounding box for each class. Check whether the error is constant or whether it varies with width and height.

Fit a model $\sigma_w^2(w), \sigma_h^2(h)$ (maybe one for each class).

2.1.5 Estimating the class probabilities

YOLO outputs a single class with its probability and bounding box.

Say class A is detected with probability p.

$$\begin{aligned}
\Pr(\text{actually class A} \mid \text{model detected class A}) &= p \\
\Pr(\text{A misclassified} \mid \text{model detected class A}) &= 1 - p \\
\Pr(\text{actually class B} \mid \text{model detected class A}) &= \Pr(\text{actually class B , A misclassified} \mid \text{model detected class A}) \\
&= \Pr(\text{A misclassified} \mid \text{model detected class A}) \\
&\quad \Pr(\text{actually class B} \mid \text{A misclassified, model detected class A}) \\
&= (1 - p)\ \Pr(\text{actually class B} \mid \text{A misclassified, model detected class A})
\end{aligned}$$

The distribution Pr(actually class B | A misclassified, model detected class A) can be approximated during training.

For example, during training, it may be the case that when an oxygen sign is misclassified, it is 10 times more likely for the actual object to be an explosive sign than a flammable sign. This is reflected in the final probability distribution.

2.2 Deprojection

2.2.1 Finding the position relative to the camera

RealSense API has a function to do just that.

2.2.2 Estimating the error

The set of transformations from pixels to world coordinates are as follows:

$(u, v) \rightarrow (u, v, d) \rightarrow (x_c, y_c, z_c)$

The first transformation is reading the depth value. If there is a small error $(\delta u, \delta v)$ in where the centre of the object is, and there's a small measurement noise in the depth reading at that pixelδd, what is the error $\delta d'$ that factors in both the depth error and the position error? The method used is from ([1])

$$(\delta u, \delta v, \delta d') \approx (\delta u, \delta v, \frac{\partial d}{\partial u}\delta u + \frac{\partial d}{\partial v}\delta v + \delta d)$$

$$\begin{aligned}
\begin{pmatrix} \delta u \\ \delta v \\ \delta d' \end{pmatrix} &\approx \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0.5(d(u+1,v) - d(u-1,v)) & 0.5(d(u,v+1) - d(u,v-1) & 1 \end{pmatrix} \begin{pmatrix} \delta u \\ \delta v \\ \delta d \end{pmatrix} \\
&= \boldsymbol{J_1} \begin{pmatrix} \delta u \\ \delta v \\ \delta d \end{pmatrix}
\end{aligned}$$

So the covariance matrix of (u, v, d') is $E\left[\begin{pmatrix}\delta u & \delta v & \delta d'\end{pmatrix}^T \begin{pmatrix}\delta u & \delta v & \delta d'\end{pmatrix}\right] = \boldsymbol{J_1}\boldsymbol{\Sigma}_1\boldsymbol{J_1}^T = \boldsymbol{\Sigma}_2$ where $\boldsymbol{\Sigma}_1$ is $\text{diag}(\sigma_u^2, \sigma_v^2, \sigma_d^2)$

The next transformation is deprojection. There is access to a function $(x_c, y_c, z_c) = f(u, v, d)$. The function is non-linear, so linearisation will need to be used again.

$$\begin{aligned}
\begin{pmatrix} \delta x_c \\ \delta y_c \\ \delta z_c \end{pmatrix} &\approx \begin{pmatrix} \frac{\partial x_c}{\partial u} & \frac{\partial x_c}{\partial v} & \frac{\partial x_c}{\partial d'} \\ \frac{\partial y_c}{\partial u} & \frac{\partial y_c}{\partial v} & \frac{\partial y_c}{\partial d'} \\ \frac{\partial z_c}{\partial u} & \frac{\partial z_c}{\partial v} & \frac{\partial z_c}{\partial d'} \end{pmatrix} \begin{pmatrix} \delta u \\ \delta v \\ \delta d' \end{pmatrix} \\
&= \boldsymbol{J_2} \begin{pmatrix} \delta u \\ \delta v \\ \delta d' \end{pmatrix}
\end{aligned}$$

The elements of $\boldsymbol{J_2}$ can be calculated numerically.

The covariance of (x_c, y_c, z_c) is therefore $\boldsymbol{\Sigma}_c = \boldsymbol{J_2}\boldsymbol{\Sigma}_2\boldsymbol{J_2}^T = \boldsymbol{J_2}\boldsymbol{J_1}\boldsymbol{\Sigma}_1\boldsymbol{J_1}^T\boldsymbol{J_2}^T$

2.3 World transformation

2.3.1 Finding the position relative to the world

The tracking camera translation $\boldsymbol{t}$ and rotation $\boldsymbol{Q}$ are available as a ROS topic.

The translation $\boldsymbol{t}_0$ and rotation $\boldsymbol{Q}_0$ of the depth camera from the tracking camera needs to be measured or estimated.

$$\begin{pmatrix} x_w \\ y_w \\ z_w \end{pmatrix} = \boldsymbol{Q} \left(\boldsymbol{Q}_0 \begin{pmatrix} x_c \\ y_c \\ z_c \end{pmatrix} + \boldsymbol{t}_0 \right) + \boldsymbol{t} + \boldsymbol{\epsilon}$$

Where $\boldsymbol{\epsilon}$ is the noise in the tracking camera position.

2.3.2 Estimating the error

$$\begin{pmatrix} x_w \\ y_w \\ z_w \end{pmatrix} = \boldsymbol{Q} \left(\boldsymbol{Q}_0 \begin{pmatrix} x_c \\ y_c \\ z_c \end{pmatrix} + \boldsymbol{t}_0 \right) + \boldsymbol{t} + \boldsymbol{\epsilon}$$

The final covariance is therefore $\boldsymbol{R} = \boldsymbol{Q}\boldsymbol{Q}_0\boldsymbol{\Sigma}_c\boldsymbol{Q}_0^T\boldsymbol{Q}^T + \boldsymbol{\Sigma}_\epsilon$.

2.4 Tracking the objects

2.4.1 Object representation

```
struct tracked_object_t{
  float position[3]; // later defined as μ
  float covariance[3][3]; // later defined as Σ
  float probabilities[num_classes]; // p
};
```

2.4.2 Kalman filter

Notation:

$\boldsymbol{\mu}$	Where the robot thinks the object is
$\boldsymbol{\Sigma}$	The covariance of the object position (small eigenvalues means the robot is sure where the object is)
$\boldsymbol{o}$	The observed position of the object
$\boldsymbol{R}$	The covariance of a single position reading from the sensors

2.4.3 Matching objects between frames [2]

Use the Mahalanobis distance $\left(m^2 = (\boldsymbol{o}_n - \boldsymbol{\mu}_n)^T (\boldsymbol{\Sigma}_n + \boldsymbol{R}_n)^{-1} (\boldsymbol{o}_n - \boldsymbol{\mu}_n)\right)$ as a metric for how close the objects are to where you think they should be. It's the higher dimensional equivalent of how many standard deviations a point is from the mean.

If there are N known objects and M objects observed in the last frame. Create the following matrix:

```
distances = np.zeros((M, N))
for i in range(M):
  for j in range(N):
    distances[i,j] = Mahalanobis(new_object[i], known_object[j])
```

The pairing algorithm is as follows:

```
i, j = np.unravel_index(np.argmin(distances), distances.shape) # get the index of the closest pair
while(distances[i,j] < threshold):
  pairs.append((i, j))
  distances[i, :] = threshold # make new object i unavailable for pairing subsequently
  distances[:, j] = threshold # make known object j unavailable for pariring subsequently
  i, j = np.unravel_index(np.argmin(distances), distances.shape)
```

- If the observations are normally distributed, $\Pr(m^2 > 9) \approx 3\%$ which is a reasonable threshold.
- It may be advisable to delete objects that you haven't seen in a while especially when they have large covariances.
- There's lots of redundancy in the algorithm, but it shouldn't matter if M×N is small

2.4.4 Kalman update step

$$\boldsymbol{\mu}_{n+1} = \boldsymbol{\mu}_n + \boldsymbol{\Sigma}_n \left(\boldsymbol{\Sigma}_n + \boldsymbol{R}_n\right)^{-1} \left(\boldsymbol{o}_n - \boldsymbol{\mu}_n\right)$$
$$\boldsymbol{\Sigma}_{n+1} = \boldsymbol{\Sigma}_n - \boldsymbol{\Sigma}_n \left(\boldsymbol{\Sigma}_n + \boldsymbol{R}_n\right)^{-1} \boldsymbol{\Sigma}_n$$

Choose sensible values for $\boldsymbol{\mu}_0$ e.g. (0,0,1) m and $\boldsymbol{\Sigma}_0$ e.g. diag(100, 100, 4) m^2 (expect objects to be around $(\pm 10, \pm 10, 1 \pm 2)$ m).

2.4.5 Combining the probabilities

Difficult because YOLO output distributions aren't independent (a misclassification in one frame will likely be there in the next frame)
Could just use the mean of all the distributions.

References

[1] O. Korkalo and T. Takala, "Measurement noise model for depth camera-based people tracking," *Sensors*, vol. 21, no. 13, p. 4488, 2021.

[2] H.-K. Chiu, J. Li, R. Ambrus, and J. Bohg, "Probabilistic 3d multi-modal, multi-object tracking for autonomous driving," *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021.