DSL

VAMA

Scene Understanding Guide

Overview

Scene understanding is essential for Autonomous Mobile Robots (AMRs) to carry out their jobs. For instance, for AMRs to enter the lifts given that there are passengers coming out and going in. The AMRs without scene understanding will attempt to enter lifts with lidar which is auto avoidance to object collision. After avoiding the passengers, there may not be vacant space in the lift for the AMRs. The AMRs would give up after several attempts to enter the lift. The whole process is not intuitive. By leveraging computer vision, AMRs would be able to see when is the right time to enter the lift and whether there is enough space for them.

On the other hand, AMRs could be deployed for specific purpose, for example in fighting CoVid-19. In that case, the following AMRs are being considered:

- chemical disinfectant robot
- UV-light disinfectant robot
- Social distancing compliance robot

Chemical and UV-light disinfectant robots should only be deployed when human is not around. Both chemical and UV-light are harmful to human. In this scenario, human detection is crucial. The AMRs have to trip the chemical spearing or UV-light once any human is detected near to the proximity. For social distancing compliance, the AMRs act like the officers, reminding the crowd to maintain social distance and put on the masks.

Scene understanding allows developers to easily integrate vision detection features within VAMA framework. The detections include

- Elevator entry indication
- Human and face detection
- Human distance projection
- Face mask detection

API Guides

Elevator Entry Indication

POST https://vama.dsldemo.site/v1/sceneunderstanding/liftentry/api

Request JSON body

```
{
  "requests": [
    {
      "features": [
        {
          "type": "ENTER_LIFT"
        }
      ],
      "video": {
        "uri": "rtsp://hostip/to/videostream"
      }
    }
  ]
}
```

Response example

```
{
  "ENTER_ELEVATOR": {
    "indicator": "Go/No-Go/Halt",
  },
  "HUMAN_DETECTION": {
    "boundingPoly": {
      "normalizedVertices": [
        {
          "x": 0.46406767,
          "y": 0.87452173,
          "width": 0.32457321,
          "height": 0.67452179,
          "score": 0.90772167
        }
      ]
    },
    "name": "ELEVATOR",
  }
}
```

Human detection

POST https://vama.dsldemo.site/v1/sceneunderstanding/humandetection/api

Request JSON body

```
{
  "requests": [
    {
      "features": [
        {
         "maxResults": 50,
         "min_height": 20,
         "min_width": 20,
         "score_th": 0.2,
         "nms_iou": 0.3,
         "type": "HUMAN_DETECTION"
        }
      ],
      "image": {
        "content": "base64-encoded-image"
      }
    }
  ]
}
```

Response example

```
{
  "HUMAN_DETECTION": {
    "boundingPoly": {
      "normalizedVertices": [
        {
          "x": 0.46406767,
          "y": 0.87452173,
          "width": 0.32457321,
          "height": 0.67452179,
          "score": 0.93772167
        },
        {
          "x": 0.5672081,
          "y": 0.87452173,
          "width": 0.27457373,
          "height": 0.57452121,
          "score": 0.95882109
        }
      ]
    },
    "name": "Human"
  }
}
```

Human Distance Projection

Camera Calibration Service

https://vama-app.dsldemo.site/v1/sceneunderstanding/calibration/

A frontend interface for camera calibration service is developed for users to upload calibration images. The following three camera parameters will be returned for use in the Human Distance API.

```
{
  "Fx": 813.282135427156,
  "Fy": 815.3081525954702,
  "centerX": 301.6707864964346,
}
```

Human Distance API

POST https://vama.dsldemo.site/v1/sceneunderstanding/humandistance/api

Request JSON Body

```
{
  "requests": [
    {
      "features": [
        {
          "minScore": 0.3,
          "MinDistance": 100,
          "Fx": 823,
          "Fy": 825,
          "centerX": 323,
          "type": "HUMAN_DISTANCE"
        }
      ],
      "image": {
        "content": "base64-encoded-image"
      }
    }
  ]
}
```

Response example


```
{
  "HUMAN_DETECTION": {
    "boundingPoly": {
      "normalizedVertices": [
        {
          "id":0,
          "x": 0.46406767,
          "y": 0.87452173,
          "width": 0.32457321,
          "height": 0.67452179,
          "score": 0.93772167
        },
        {
          "id":1,
          "x": 0.5672081,
          "y": 0.87452173,
          "width": 0.27457373,
          "height": 0.57452121,
          "score": 0.95882109
        },
        {
          "id":2,
          "x": 0.7637053,
          "y": 0.28956301,
          "width": 0.17457373,
          "height": 0.77452121,
          "score": 0.8569042
        }
      ]
    },
    "name": "Human"
  },
  "HUMAN_DISTANCE": {
    "humanNum": 3,
    "distancesMatrix": [
      [0.000000, 139.0871, 208.7654],
      [139.0871, 0.000000, 118.0034],
      [208.7654, 118.0034, 0.000000]
    ]
  }
 ]
}
```

Legends

```
{
  "Fx": 813.282135427156,      // Focal length in horizontal axis
  "Fy": 815.3081525954702,     // Focal length in vertical axis
  "centerX": 301.6707864964346,// camera center in horizontal axis
}


{
  "minScore": 0.3,         // human detection confidence score
  "distance": 100,         // distance (cm) between human
  "Fx": 823,               // Focal length in horizontal axis
  "Fy": 825,               // Focal length in vertical axis
  "centerX": 323,          // camera center in horizontal axis(pixel)
  "type": "HUMAN_DISTANCE"
}


"distancesMatrix": [
  [0.000000, 139.0871, 208.7654],    // Distance matrix is in cm
  [139.0871, 0.000000, 118.0034],
  [208.7654, 118.0034, 0.000000]
]
```

Face detection

POST https://vama.dsldemo.site/v1/sceneunderstanding/facemaskdetection/api

Request JSON body

```
{
  "requests": [
    {
      "features": [
        {
         "maxResults": 50,
          "type": "FACE_DETECTION"
        }
      ],
      "image": {
        "content": "base64-encoded-image"
      }
    }
  ]
}
```

Response example

```
{
  "FACE_DETECTION": {
    "boundingPoly": {
      "normalizedVertices": [
        {
          "x": 0.56406767,
          "y": 0.87452173,
          "width": 0.22457321,
          "height": 0.24452179,
          "score": 0.97772167
        },
        {
          "x": 0.6672081,
          "y": 0.87452173,
          "width": 0.17457373,
          "height": 0.18452121,
          "score": 0.98882109
        }
      ]
    },
    "name": "Face"
  }
}
```

Face Mask Detection

POST https://vama.dsldemo.site/v1/sceneunderstanding/facedetection/api

Request JSON body

```
{
  "requests": [
    {
      "features": [
        {
          "maxResults": 20,
          "type": "FACEMASK_DETECTION"
        }
      ],
      "image": {
        "content": "base64-encoded-image"
      }
    }
  ]
}
```

Response example

```
{
  "FACE_MASK_DETECTION": [
    {
      "boundingPoly": {
        "normalizedVertices": [
          {
            "x": 0.46406767,
            "y": 0.47452173,
            "width": 0.32452173,
            "height": 0.27452173,
            "score": 0.98216777
          },
          {
            "x": 0.8672081,
            "y": 0.57452173,
            "width": 0.27752173,
            "height": 0.27452173,
            "score": 0.99772167
          }
        ]
      },
      "name": "Face Mask"
    },
    {
      "boundingPoly": {
        "normalizedVertices": [
```

```
          {
            "x": 0.66406767,
            "y": 0.57452173,
            "width": 0.30452173,
            "height": 0.26452173,
            "score": 0.95216777
          }
        ]
      },
      "name": "No Face Mask"
    }
  ]
}
```

Legends

```
    "normalizedVertices": [
      {
        "x": 0.46406767,        // x_ori / width_ori
        "y": 0.87452173,        // y_ori /height_ori
        "width": 0.32457321,    // bbox_width / width_ori
        "height": 0.67452179,   // bbox_height / height_ori
        "score": 0.90772167     // confidence scores are in descending order
      }
    ]
```

Installation & Deployment

Prerequisite Software Installation

Docker

Follow the official installation guide based on the OS of the machine. You may consider to install Nginx and configure it to host multi workers for a particular container and do reverse proxy.

TLDR

Each API is containerized using Docker. To deploy the API for any of the following container,

- maskdetection
- humandetection
- humandistance
- elevatorentry,

follow the following steps:

- Build the docker container, go to the folder which contain Dockerfile
    - docker build -t maskdetection .
- Run the docker. Inside each folder, it contains the run.sh file
    - `docker run -d -p 80:80 maskdetection`
- Access the website using the web browser: http://localhost
- There is Python example code how to call the REST API in the html page

API Deployment in Details

1. In the terminal, navigate to the main folder containing the `Dockerfile` file.

2. In the terminal, enter

```
$ sudo docker build -t [imagename] .
```

This builds a docker image of the flask application. Note that `imagename` can be any name that you assign to it. The dot (.) following it refers to the path to the Dockerfile, which in this case, is the current folder.

3. Next, run the image as a container.

```
$ sudo docker run -d -p host_port:container_port --name
[containername] [imagename]
```

`containername` refers to the name you want to assign for the container. `-d` refers to running the container in the background instead of foreground. `-p host_port:container_port` refers to the port that the host maps with the container's port. An example will be `-p 5001:5000`. Note that the flask app's port running in the container is by default set to the default 5000.

4. To check that the container is running.

```
$ sudo docker ps
```

It should show something like this.

```
(base)                    $ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS          PORTS                    NAMES
0639cb74e99c   imagename      "python -u app.py"   10 seconds ago   Up 9 seconds    0.0.0.0:5001->5000/tcp   containername
```

5. The API endpoint is set as `/api` (this can be modified in the `app/app.py` file). To test that the API is working, use the URL of your `hostname:hostport/api`. Taking the example in step 4, and the host as localhost, it will be `localhost:5001/api`

The request to be sent to this API needs to adhere strictly to the API Guidelines for each module specified in the previous section. Below is an example of how to send a request using python v3 for facemask detection.

```
from urllib.request import urlopen, Request
import argparse
import base64
import json

def SendImageToSceneUnderstandingServer(url, Imagefile):
    with open(Imagefile, 'rb') as image:
        base64_bytes = base64.b64encode(image.read())

    data = {}
    request = {}
    feature = {}
    myimage = {}
    features_list = []
    requests_list = []
    myimage['content'] = base64_bytes.decode('utf-8')
    request['image'] = myimage
    feature['maxResults'] = 20
    feature['type'] = 'FACEMASK_DETECTION'
    features_list.append(feature)
    request['features'] = features_list
    requests_list.append(request)
    data['requests'] = requests_list

    params = json.dumps(data).encode('utf8')
    req = Request(url, data=params, headers={'content-type':
'application/json'})
    response = urlopen(req)
    json_response = response.read()

    print("\nRequest: %s" %Imagefile)
    print("Response: %s" %json_response)

    j = json.loads(json_response)
    return j

##########################################################################

parser = argparse.ArgumentParser(description=__doc__,
formatter_class=argparse.RawDescriptionHelpFormatter)
parser.add_argument('Imagefile', help='Image File location')
args = parser.parse_args()

Imagefile = args.Imagefile

url = 'localhost:5001/api'
SendImageToSceneUnderstandingServer(url, Imagefile)
```