Detection and Tracking of Moving Objects

M. Perrollaz, A. Nègre

November 13, 2014

The objective of this project is to familiarize with some basic concepts of DATMO, and to understand the problems behind the use of DATMO with real data. The data used for this work has been recorded on the road for the application in Advanced Driver Assistance System, which is a promising field of application of DATMO. Two classical problems will be studied in this work:

- detection using a vision system,
- tracking a moving object from laser data.

These two parts are independent.

You will have to write a C++ code to detect obstacles using a stereo camera and to track an obstacle with a LIDAR sensor. A report has to be written to explain the implementation and to analyze the results. The proposed methods, as well as the clarity of the code and the report will count for the final mark.
If you prefere, it is also allowed to use python instead of C++. OpenCV library is availlable in both language.

The resources needed for this work (images, laser data, examples) are provided at this url :
`http://emotion.inrialpes.fr/people/perrolla/mosig/`

1 Prequisites

A significant part of this work relies on the OpenCV library.

1.1 Downloading/installing openCV

`http://opencv.willowgarage.com/wiki/`

1.2 Need some information ?

`http://opencv.willowgarage.com/documentation/cpp/index.html`

1.3 Compiling/running an example

```
$ g++ example.cpp -o example -I /usr/local/include/opencv/ -L /usr/local/lib/
    -lopencv_highgui -lopencv_calib3d -lopencv_legacy
$./example image_name.png
```

2 Detection using stereo-vision

The objective of this section is to study simple ways to detect objects from a pair of rectified stereoscopic images. This process is separated into 3 stages: (1) computation of range data from the stereo images, (2) separation of road and obstacle pixels from the range data, (3) clustering of the obstacle pixels for retrieving object representations, with estimated positions.

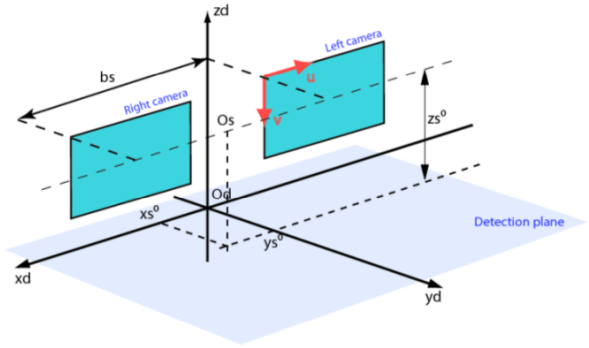

Figure 1: Coordinate system used for the stereo-vision

2.1 Computation of disparity data

First of all, the image data must be converted into a disparity map, which is equivalent to a partial 3D representation of the scene.

1. Starting from the file example.cpp, read and display the pair of stereo images left.png and right.png.

2. Using the class *cv::StereoSGBM*, compute and display the disparity map. Note that the values in the disparity image must be divided by 16 to be used in pixel unit. As parameters you can use *StereoSGBM(0, 32, 7, 8*7*7, 32*7*7, 2, 0, 5, 100, 32, true)*.
   Attention: the SGBM method produces a 16 bits disparity image. In order to display it properly, you must create another 8bits image for display, and call the *convertTo* method:
   `disparity_img.convertTo(display_img, CV_8U);`

2.2 Road/obstacles segmentation in Cartesian space

A simple way to separated pixels belonging to the ground surface from pixels belonging to the obstacles consists in converting the disparity values into 3D values and applying a threshold on the height. Assuming that the stereo camera is parallel to the road surface, a pixel (u,v,d) of the disparity map is converted into a 3D points (X,Y,Z) through the equation system:

$$\begin{cases} X &= \frac{(u-u_0)\cdot b}{d} - \frac{b}{2} \\ Z &= Z_0 - \frac{(v-v_0)\alpha_u \cdot b}{\alpha_v \cdot d} \\ Y &= \frac{\alpha_u \cdot b}{d} \end{cases}$$

with $u_0 = 258$, $v_0 = 156$, $\alpha_u = \alpha_v = 410$, being the intrinsic parameters of the camera. $b = 0.22m$ is the stereo baseline. $Zs_0 = 1.28m$ is the height of the camera.
Attention: Note that d is obtained by dividing the disparity value by 16 (due to OpenCV's implementation of SGBM).

1. For all the pixels in the disparity map, compute the 3D coordinates, and remove it from the image (set the disparity to 0) if $Z < 0.2m$. Test with other threshold values and comment.

2. Remove also the pixels that are above $2.5m$ height.

2.3 Road/obstacle segmentation in Disparity space

There are sometimes advantages in working in disparity space rather than in Cartesian space. One advantage is that the horizontal projection is easily implemented using the v-disparity approach. The v-disparity image is an image of size (height, max-disparity) where each row is an histogram of the disparity values of the image row.

The algorithm to compute the v-disparity is :

```
for v=0..height-1
     for u=0..width-1
         if (Disparity(u,v)>0)
             vDisparity(Disparity(u,v), v) += 1
         endif
      end
 end
```

1. Compute the v-disparity image associated to the disparity image computed in 2.1.2. Use 32 for the maximum disparity value.

2. The v-disparity image implements the horizontal projection in disparity space. Therefore, considering the road surface as a horizontal plane, it is projected as a straight line in the v-disparity plane.

   Manually measure the position h_0 of the horizon line and the slope p_0 of the road surface. You can retrieve the height and pitch angle of the camera through equation:

$$\begin{aligned} \theta &= -atan(\tfrac{v_0 - h_0}{\alpha_v}) \\ Z_0 &= -b \cdot cos(\theta) \cdot p_0 \end{aligned}$$

3. Now extract the road surface using the provided RANSAC algorithm. To do that, first apply a threshold (*cv::threshold*) to the v-disparity image, in order to remove some noise. (You can use a threshold value around 60). Then add all the remaining points into a vector of Point2f, using the following syntax: `vec.push_back(Point2f(i,j));` Finally, use the *FitLineRansac* function, provided in tp_util.cpp.

4. Filter the disparity image by removing pixels located under the road surface (plus a small margin) by using the detected line and the disparity value. Compare the results with 2.2.1.

2.4 Clustering

In order to detect the objects in the scene, we propose to extract the connected components of constant disparity in the image created in question 2.2.2 and in 2.3.4.

1. Apply a morphological erosion (*cv::erode*) and then a dilatation (*cv::dilate*) to the disparity image created in question 2.2.2 and 2.3.4.

2. Extract regions of connected components in disparity space, by using the provided *segmentDisparity* function.

3. For each labeled region, compute the bounding box (umin, umax, vmin, vmax) and the mean disparity value. Using the center of the bounding box and the mean disparity value, compute the position of each target in Cartesian space.

4. Filter out the regions whose surface is less than 50 pixels, to remove false positives. Draw a rectangle in the image around the remaining regions.

2.5 To go further

Try to detect obstacles in the other image pairs provided. Comment the problems that you encounter, if any.

3 Tracking from laser scanner data

In this section, we propose to detect and track objects using a 2D laser scanner.

3.1 Representing laser data

The provided laser data is a succession of planar scans. Each scan is a set of distance measurements, in a polar coordinate system. The frame-rate of the lidar is 12.5Hz.

1. Compile and run the program called readLidarData.cpp. This program shows the laser data projected on a grid.
2. We define a surveillance area as: $-10m < X < 10m$ and $0 < Y < 30m$. Modify the file readLidarData.cpp in order to display on the left image only the data situated inside this area.
3. On the first frame, a bicycle is situated in a region of interest (ROI) defined by $4m < X < 7.5m$ and $9m < Y < 11m$. Compute the mean position of the lidar impacts situated inside this area.

3.2 Tracking with Kalman filter

Now, we propose to track the lidar detection over time, using a Kalman filter. In order to avoid the difficult problem of data association, only one object will be detected and tracked.

1. Implement a Kalman filter using the KalmanFilter class of OpenCV. For this, the observed data is the mean position of the detected object. The evolution is a simple constant velocity model.
   The state contains the position and the speed of the object: $\mathbf{X} = (X, Y, V_X, V_Y)^T$.
   The transition matrix expresses the evolution model of the system, it is given by :

$$A = \begin{bmatrix} 1 & 0 & dt & 0 \\ 0 & 1 & 0 & dt \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

   As we observe only the position, the measurement matrix corresponds to :

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

   Measurement and process noise covariance matrices need to be provided, you can suppose that these matrices are diagonal.

   Kalman filter is based on a prediction/correction loop. After every prediction step, move the ROI so that it is centered on the predicted position. Then compute the observed position by taking the mean value of the lidar impacts inside the predicted ROI. Use this observation for the correction stage of the filter.

   For every loop, display on the grid both the predicted and the measured position, with a different color. Observe how the noise parameters have an influence on the result. Comment.

2. Experiment with different noise parameters and draw a plot of the estimated velocity of the target.