University of Florida
Department of Electrical and Computer Engineering
EEL 4665/5666
Intelligent Machines Design Laboratory

Special Sensor Description

Introduction

In the frenzy of technological advancements, there has been a strong movement towards automation of tasks with the main goal of improving our conditions of living. With this in mind, the goal of our autonomous towing machine, is to be able to detect and actuate when an automotive has fail and needs a towing service; this task is to be perform autonomously. The application of this idea is to be extended to full-scale automotive and, to accomplish the vision in mind, we begin advancing this idea on small scale robots.

The inspiration of the project is derived at the current technology that is available, that is, Global Positioning System (GPS). The idea is to have an eye-, God-camera that would locate objects in its visibility and it would be able to provide specific location coordinates of those objects. The initial stages of the project consisted of performing obstacle avoidance. This is a key feature for the robot because as the robot navigates to the desired object, in our case to the broken car, it must be able to avoid obstacles that are on its way. As explained above, the special sensor used in this project will be an eye-, God-camera.

Procedure

The procedure can be broken down into three main categories: Image Processing, Mathematics, and Feedback System. We'll examine each one of them carefully.

I. Image Processing

All the image processing is performed by a Raspberry Pi 2 and PS3 Eye Camera with OpenCV and Numpy libraries using Python 2.7 as the coding platform. The overview steps for image processing are shown below with their respective OpenCV function name. The complete code can be found in *Appendix A.*

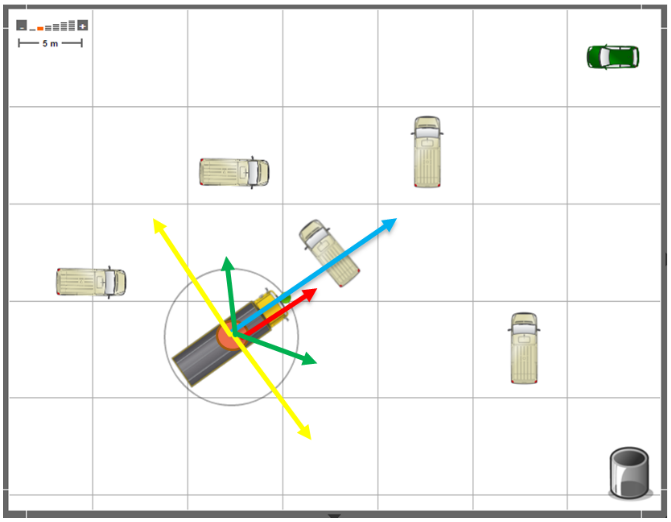
- Image Processing:
  - Convert image from Color to HSV Values
    - **cv2.cvtColor(im, cv2.COLOR_BGR2HSV)**
  - Mask the image for color detection
    - **cv2.inRange()**
  - Erode and dilate the image for noise reduction
    - **cv2.erode(), cv2.dilate()**
  - Find contours of objects
    - **cv2.findContours()**
  - Find centroids of objects
    - **cv2.moments()**

II. Mathematics

The mathematics are the brain of the entire data processing that is able to guide and command the towing robot towards the desire location. The necessary vectors components are shown in the list below.

- Mathematics:
  - Find the origin and object centroids
  - From the objects centroids, create vectors
  - Vector 1 is Robot’s heading
  - Vector 2 is path direction
  - Vector 3 is orthogonal to vector 1
  - Vector 4 is sum of vector 1 and 3 and is effective path direction

Fig. 1 shows all the different vectors that are utilize for heading estimation, path planning, and obstacle avoidance. The correct magnitude representation of speed has not be calculated yet. This is the current phase the project is at. See *Appendix B* for a trajectory simulation.



●	Origin	-
→	Vector 1	Heading
→	Vector 2	Path
→	Vector 3	Ortho.
→	Vector 4	Sum

Figure 1 Vector representation of the robot in order to arrive at the desired location while avoiding obstacles.

III. Feedback System

The feedback system is the most important component for the obstacles avoidance. The robot will use three ultrasonic sensors that will be able to detect the approximate distance of objects. This sensors serves as feedback system to the Raspberry Pi. The Teensy 3.2, which activates both the motors and the ultrasonic sensors, will be communicating with the Raspberry Pi via radio frequencies using an XBee. Below is an overview of the feedback system.

- Feedback System:
    - Receive objects distance from Ultrasonic
    - Communicate from Teensy to Raspberry Pi via Xbee
    - Process logic in Raspberry Pi
    - Send command to Teensy for actuation

The feedback system has not been implemented yet.

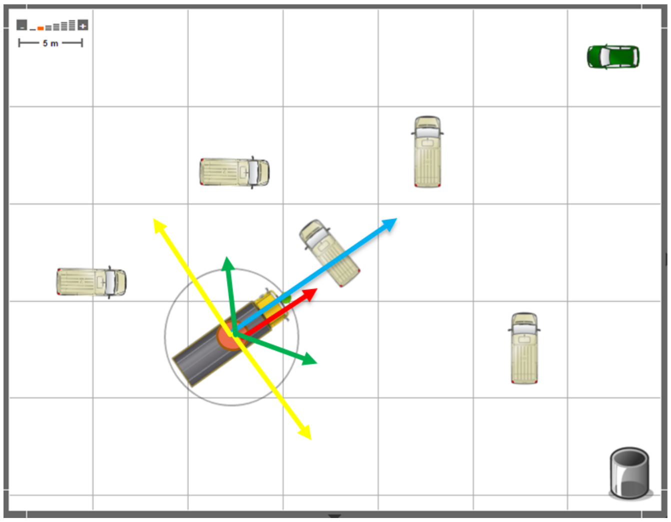
Appendix A

Python Code

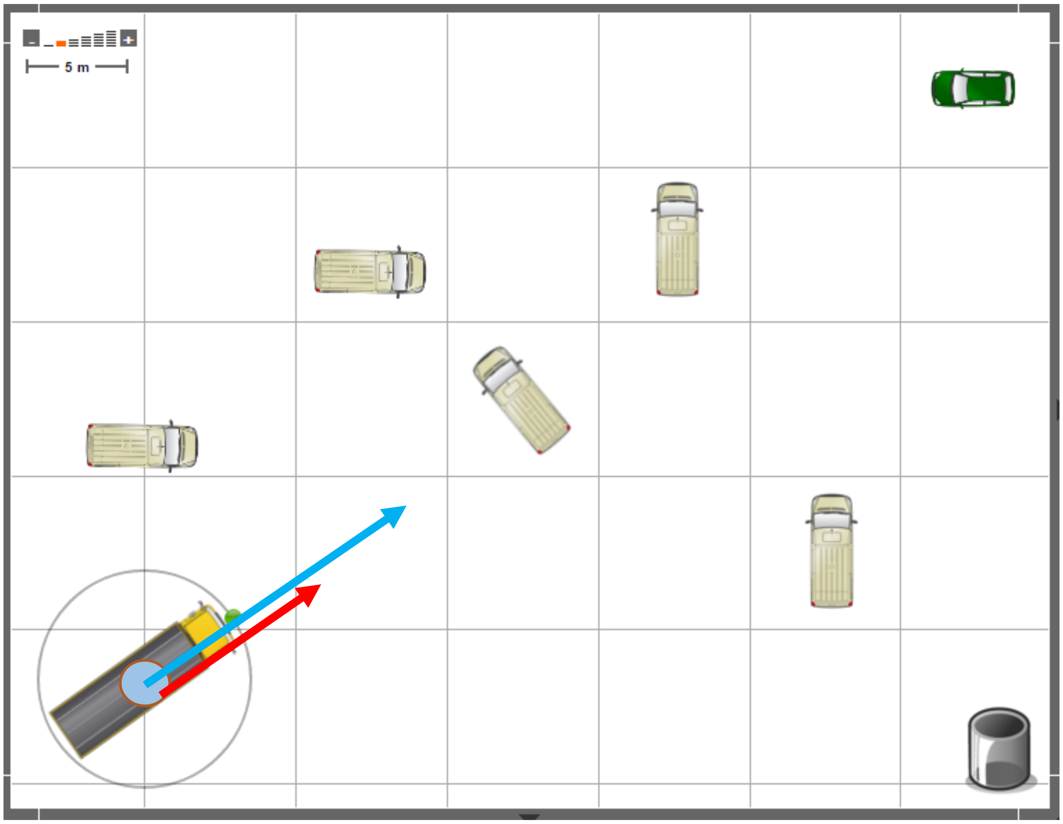
```
import numpy as np
import cv2
from numpy import matrix
import math
# Import the image
cap = cv2.VideoCapture(0)
while(1):
    ret, im = cap.read()

    # Convert to HSV colorspace
    hsv = cv2.cvtColor(im,
cv2.COLOR_BGR2HSV)

    # Define color range for masking
    lower_blue = np.array([105,201,78])
    upper_blue = np.array([150,255,255])
    # Define color range for masking---2
    lower_red = np.array([0,174,130])
    upper_red = np.array([70,255,255])

    # Apply the mask
    blue_mask = cv2.inRange(hsv,
lower_blue, upper_blue)
    # Apply the mask---2
    red_mask = cv2.inRange(hsv, lower_red,
upper_red)
    mask= blue_mask + red_mask

    # Apply filters to clean up noise
    morphd_blue = cv2.erode(blue_mask,
None, iterations =2)
    morphd_blue = cv2.dilate(morphd_blue,
None, iterations =2)
    # Apply filters to clean up noise---2
    morphd_red = cv2.erode(red_mask,
None, iterations =2)
    morphd_red = cv2.dilate(morphd_red,
None, iterations =2)

    # Find contours
    image, contours_blue, hier =
cv2.findContours(morphd_blue,
cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    center = None
    # Find contours---2
    image, contours_red, hier =
cv2.findContours(morphd_red,
cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    center = None

    #Displays only the Contours
    outline_blue = cv2.drawContours(im,
contours_blue, -1, (255,0,0), 3)
    #Displays only the Contours---2
    outline_red = cv2.drawContours(im,
contours_red, -1, (0,0,255), 3)
    #Moment Function Implemented
    center_blue = []
    if len(contours_blue) > 0:
        print contours_blue, 'blue object
detected'
        for i in range(len(contours_blue)):
            M_b =
cv2.moments(contours_blue[i])

center_blue.append((int(M_b['m10']/M_b['
m00']),int(M_b['m01']/M_b['m00'])))
            cv2.circle(im, center_blue[-1], 3,
(0,255,0), -1)

    else:
        print 'No blue object detected'
    #Moment Function Implemented---2
    center_red = []
    if len(contours_red) > 0:
        for i in range(len(contours_red)):
            M_r = cv2.moments(contours_red[i])

center_red.append((int(M_r['m10']/M_r['m0
0']),int(M_r['m01']/M_r['m00'])))
            cv2.circle(im, center_red[-1], 3,
(0,255,0), -1)
        print contours_red, 'red object detected'
    else:
        print 'No red object detected'

    # Display the image, esc to kill the
window
    cv2.imshow('image',im)
    cv2.imshow('blue',blue_mask)
    cv2.imshow('red', red_mask)

    k = cv2.waitKey(5) & 0xFF
    if k == 27:
        break

cap.release()

cv2.destroyAllWindows()
```

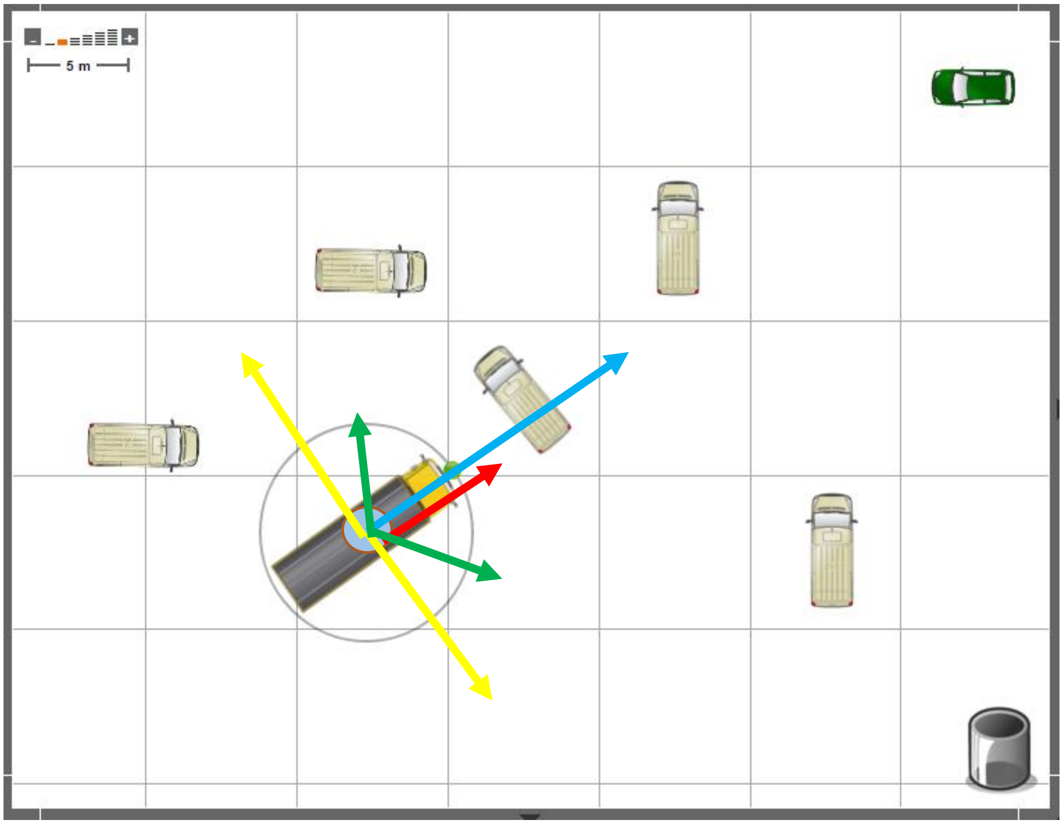
Appendix B

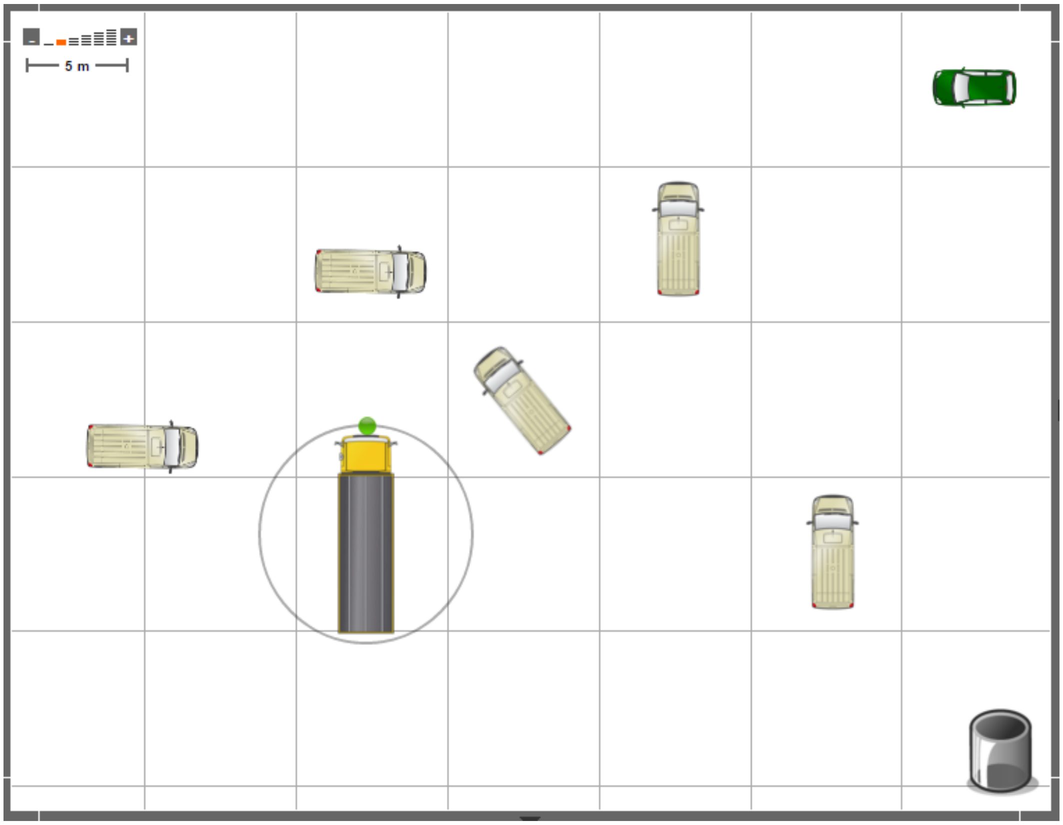
Trajectory Simulation

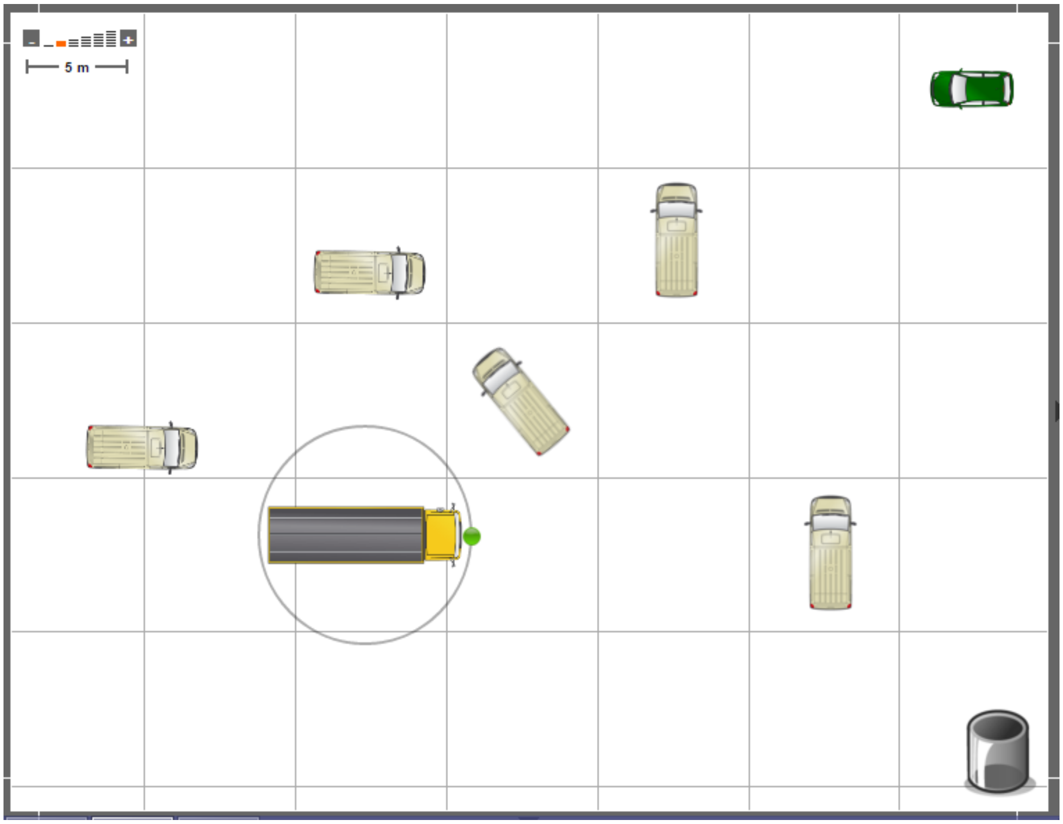


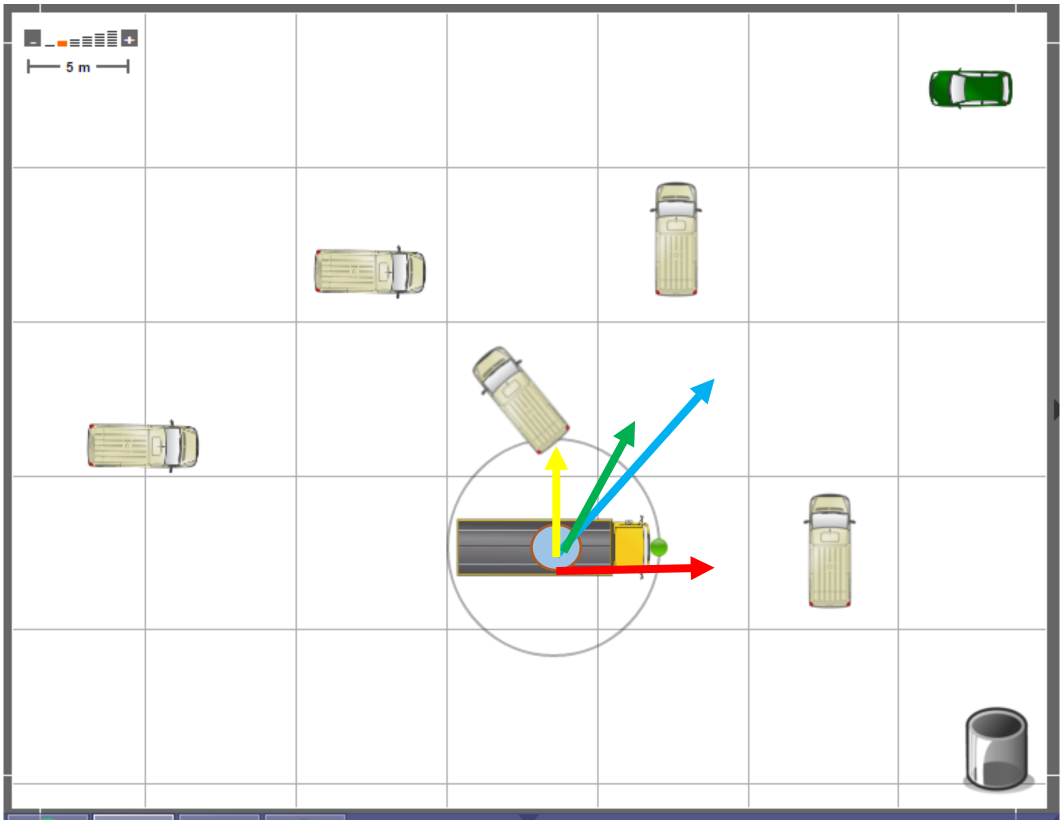
●	Origin	-
→	Vector 1	Heading
→	Vector 2	Path
→	Vector 3	Ortho.
→	Vector 4	Sum

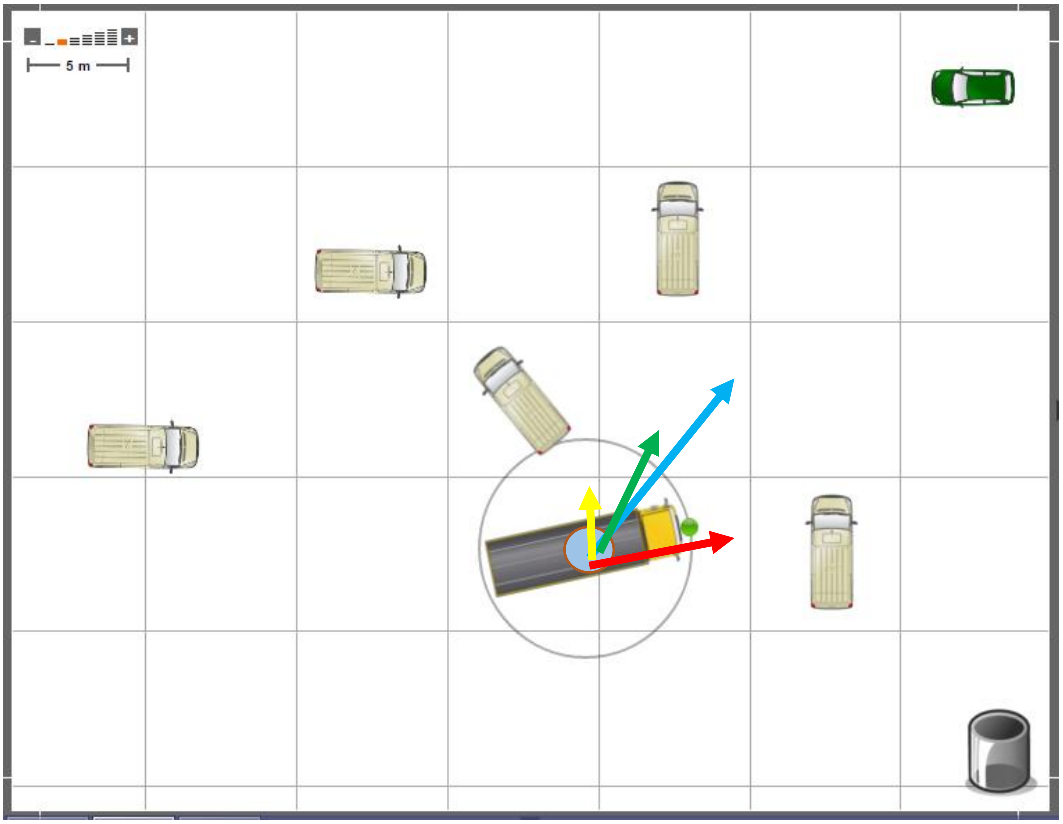


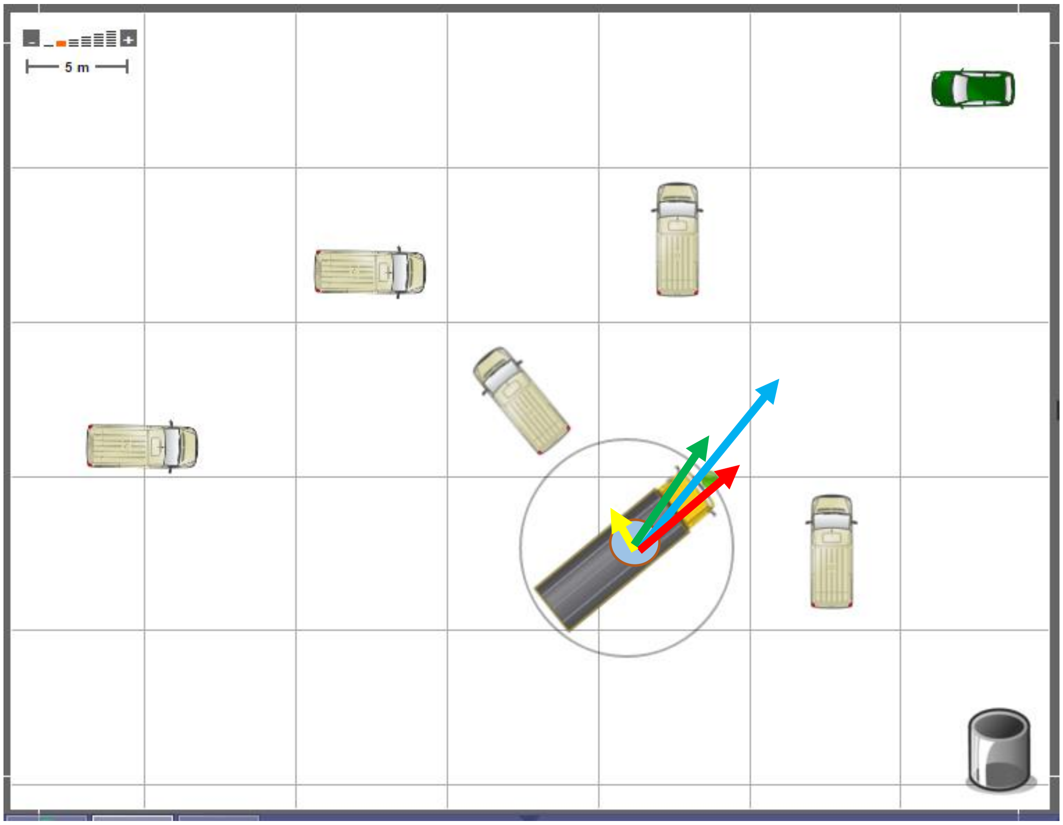

5 m

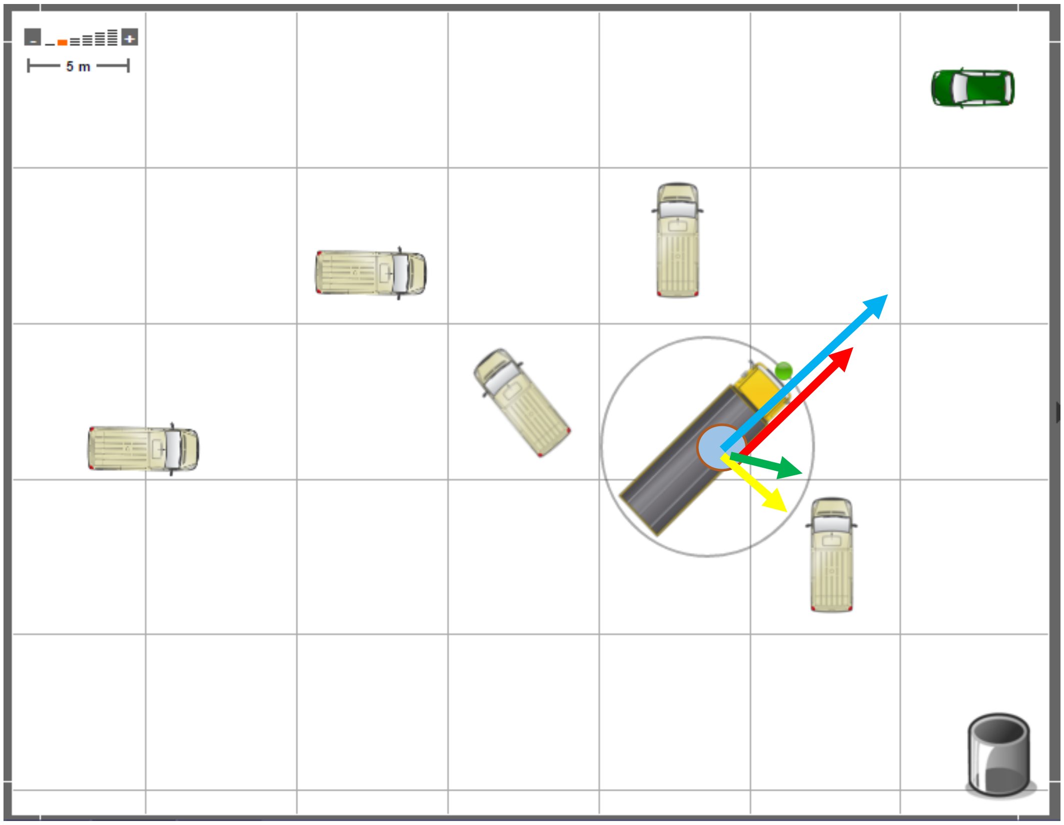

5 m

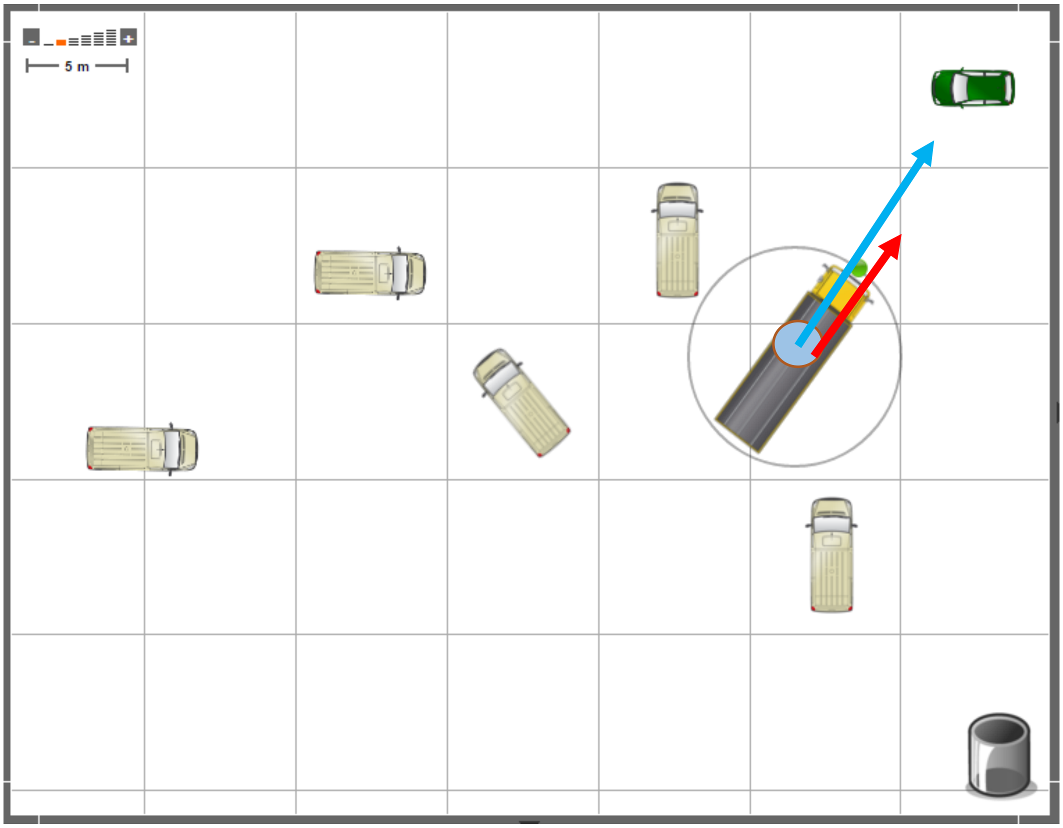

5 m

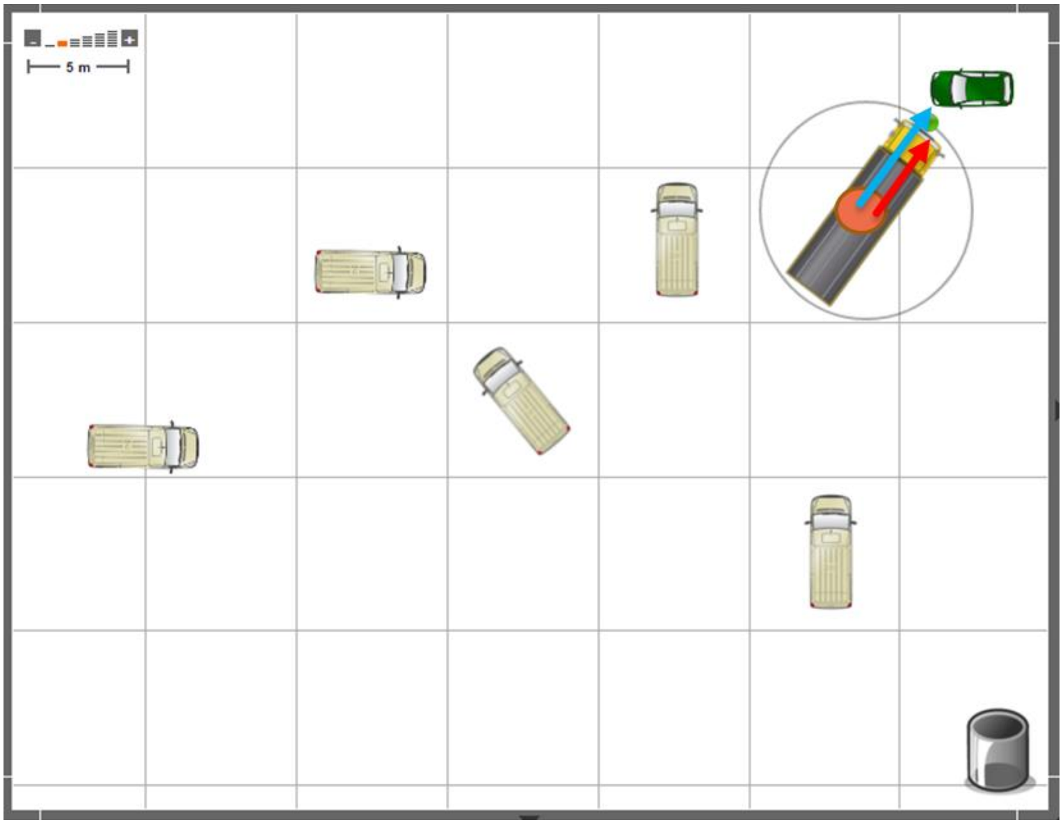

5 m


-
+
5 m


-
+
5 m


5 m


5 m


5 m