10. Assignment, Introduction to Robotics WS17/18 - Ver. 1.0

Prof. Daniel Göhring, Martin Dreher, Jakob Krause
Institut für Informatik, Freie Universität Berlin
Submission: online until Tuesday, 16 Jan 2018, 11:55 a.m.

Please summarize your results (images and descriptions) in a pdf-document and name it, e.g., "RO-10-<surnames of the students - group name>.pdf".
Submit your python code and video
Only one member of the group must submit the necessary files.
Do not copy solutions to other groups.
Every group must contain two people, unless granted differently.
Only submissions via KVV will be accepted.

Path Following, Part I

https://github.com/AutoModelCar/AutoModelCarWiki/wiki/Navigation

There are two paths on the map below, one path is on the center of the first lane and second one on the center of the second lane.

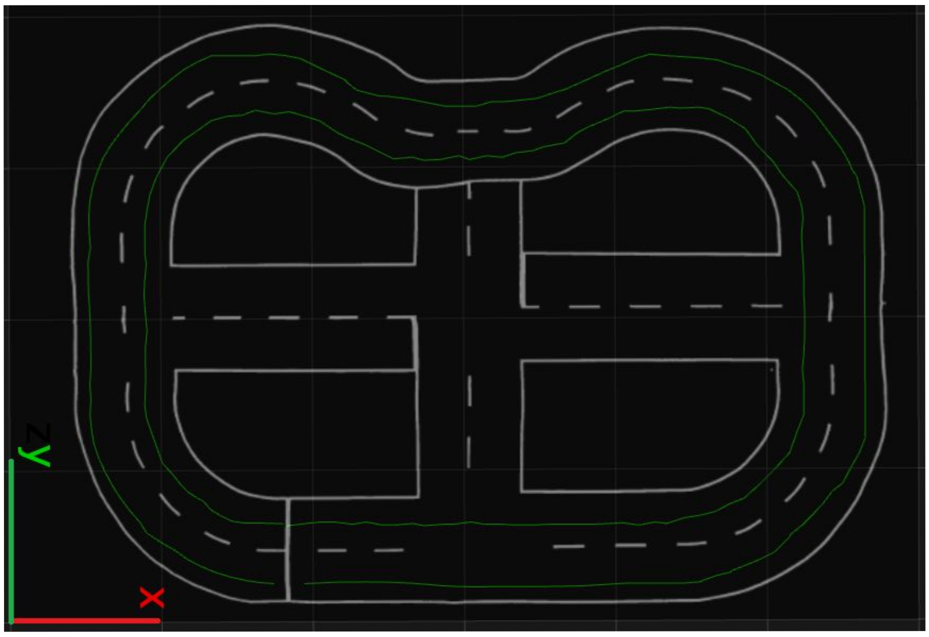


Figure 1: Two paths in the middle of the lanes

For each lane, we provide you a potential field map. (matrixDynamic_lane1.npy And matrixDynamic_lane2.npy)
A potential field map is a 2D matrix containing a force array [f_x, f_y] in map coordinates.
The matrix indexes are x and y - map coordinates with 10 cm resolution in each dimension.
For example, the force for a point (201cm,101cm), is saved in

matrix[201/10,101/10]=matrix[20,10]. A Force vector is a vector from the point to the lookahead point which is at least 20 cm ahead of the closest point on the lane.
For each point on the map, after finding the closest point on the lane, the lookahead point is calculated based on its distance to the path. When the position of the car is far away from the lane, the lookahead point is 20 cm ahead of closest point, whereas when the position of the car is on the path, the look ahead point is pushed further to 1.2 meter (for the first lane) and 0.7 meter (for the second lane)

*lookahead_offset_for_lane_1=0.2 + 1.0/(1.0*distance+1) meter*

*lookahead_offset_for_lane_2 =0.2 + 0.5/(0.5*distance+1) meter*

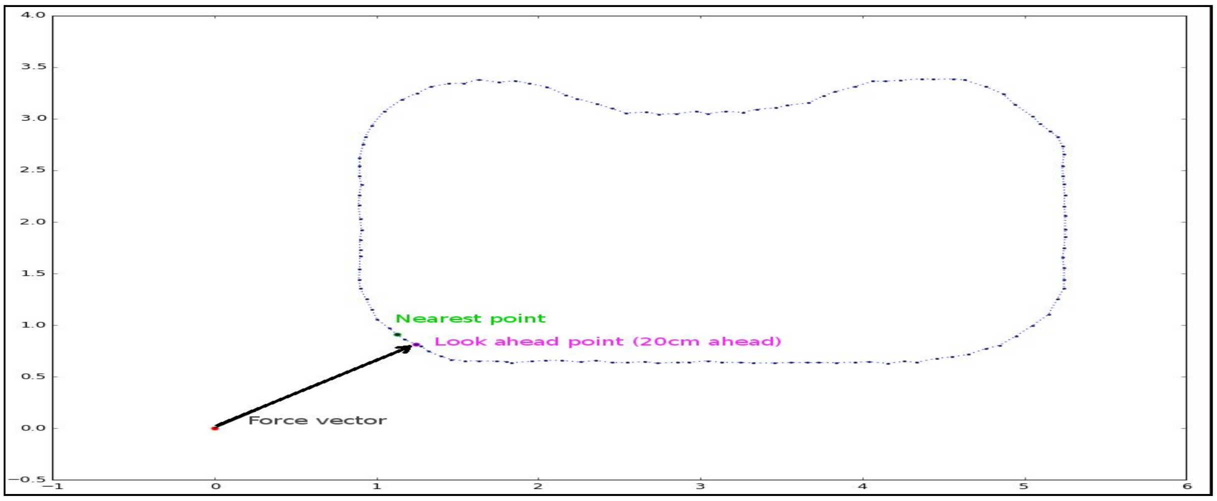
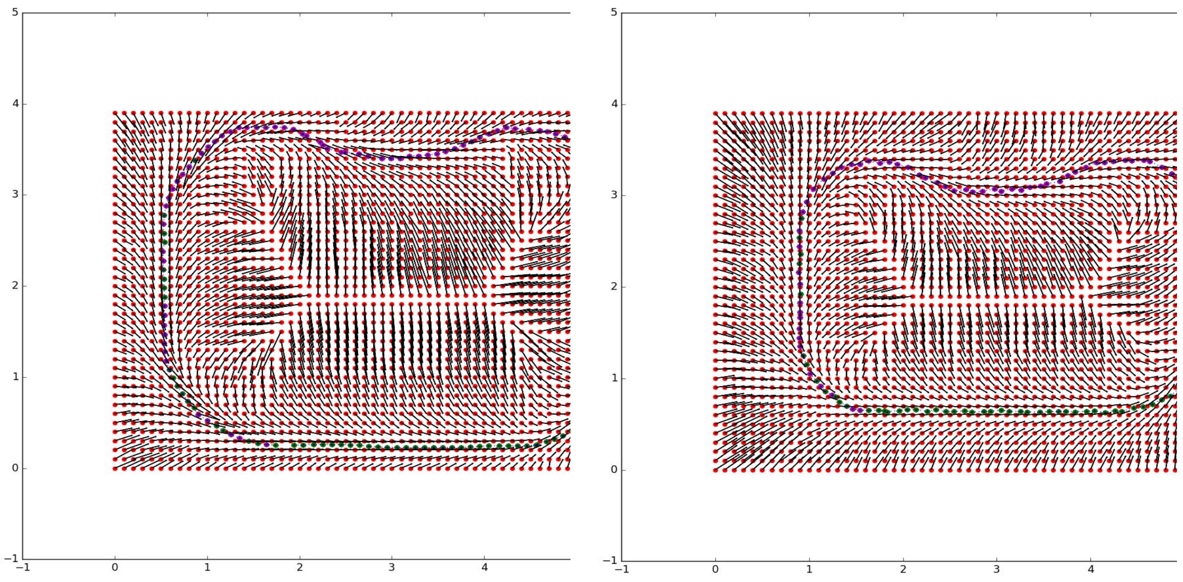


Figure 1: Force vector of the red point, Green point: Nearest point, Pink point: Look ahead point, Blue points: desired path



a. matrixDynamic_lane1.npy b) matrixDynamic_lane2.npy

Figure 2: potential field maps toward the lanes

You can modify the code below and make your own potential field map: https://github.com/AutoModelCar/catkin_ws_user/blob/master/src/fu_navigation/src/make_force_matrixs.py

1 - Load the "matrixDynamic" and find the steering angle (5 points)

Read the npy matrix and find the desired steering based on the potential field. You can use the formula below. If you find a better solution, you can use your formula, too. Normal cars drive on circles based on the steering angle (Ackermann drive). After finding the desired steering angle *(a) plot the turning circle, do this for 3 different positions*, as shown in the picture below. The steering angle of the car has a limitation, i.e., it cannot be larger than 45 degrees. Therefore, limit the desired angle. Keep in mind that the car can drive forward or backwards, depending on f_x.
f_x, f_y are just different names for F_x_car_coordinate, F_y_car_coordinate

```
F_x_map_coordinate, F_y_map_coordinate = matrix[x_index,y_index,:]
F_x_car_coordinate = cos(yaw)*F_x_map_coordinate+sin(yaw)*F_y_map_coordinate
F_y_car_coordinate = -sin(yaw)*F_x_map_coordinate+cos(yaw)*F_y_map_coordinate
Kp=4
steering=Kp*arctan(f_y/(2.5*f_x))
```

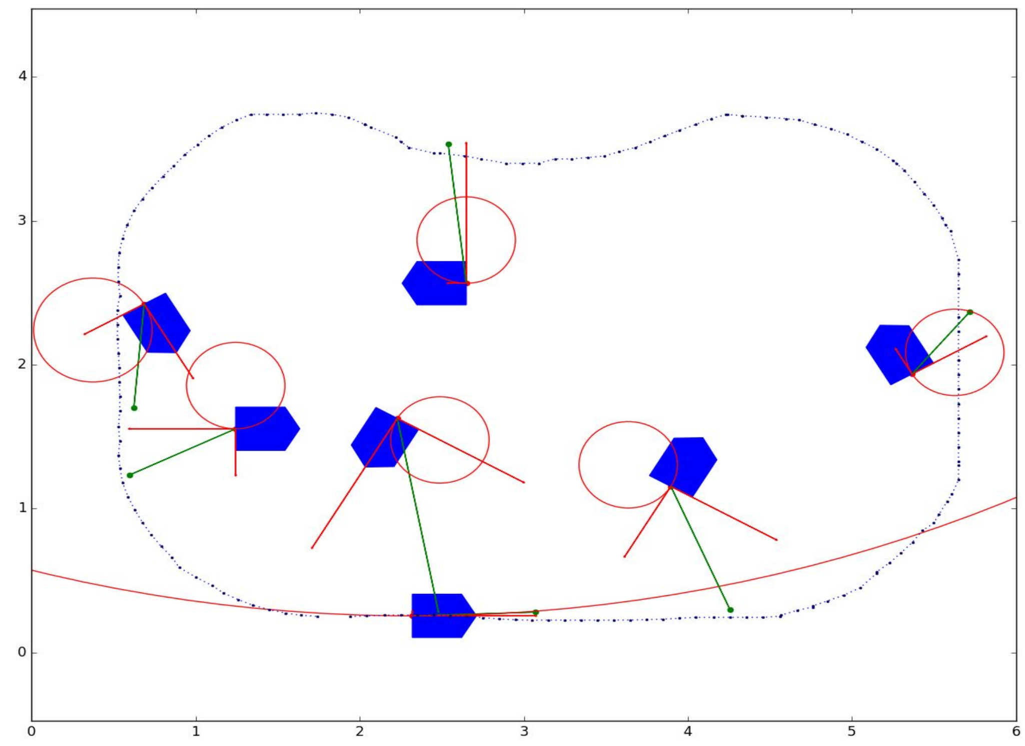


Figure 3: a Plot of the drive circles for different points

b) Plot the positions (using your Kalman-filtered GPS localization) while driving 3 circles. It will be interesting to see, whether the measured positions look like circles or if they contain any position jumps. Submit the plots in your Pdf.

2 - Navigation using the force matrix (5 points)
Design a controller using force field to follow the desired path.

- Load the matrixDynamic_lane1.npy.
- Subscribe to the car position. (x,y, yaw).
- Find the desired steering.
- Design a controller.
- Publish a desired speed and steer output of the controller.
    - Hint: When the f_x_car_coordinate is negative, then the car should move backwards, like the video below: https://www.youtube.com/watch?v=MsKjV596Fak&feature=youtu.be
- Now let the car follow the lane.
- *Plot the car's driven position (e.g. in rviz).*
- *record a video which shows how the car can follow the lane in the real world, and upload it (ogv or mp4, max 5 MB) and upload the video*

A controller code example:
https://github.com/AutoModelCar/catkin_ws_user/blob/master/src/fu_navigation/src/control.py