Homework 3

ME 500 A4 - Prof. Tron

Saturday 28th April, 2018

The goal of this homework is to implement a vision-based line tracker for your robot. You will first build a small classifier to classify pixels as line/background according to their color. Then, you will use this classifier to detect the center of a line in the image. Finally, you will implement a simple PD controller to make your robot follow the line while moving forward.

General instructions

The assignment is intended to be completed in pairs using *pair programming* (remember, two people working on *one* laptop). See the syllabus for details, and see Blackboard for the group assignments.

Homework report (required). Solve each problem, and then prepare a small PDF report containing comments on what you did for each problem, including whether you tackled the **optional** questions.

Demo video (required). You need to prepare a one to three minutes video with the following elements:

- A foreground framing of yourself (both members of the group) saying your names and the name(s) of the robot(s).
- For each question marked as **video**, a short segment showing the laptop and robot demonstrating that you completed the requested work. If you could not complete a question, you must explain what were the problems that you encountered.
- A final conclusion where you explain what you learned with the activity.

Submitting multiple video segments instead of a single video is allowed but not encouraged (in this case, all the segments must be recorded in the same session, e.g., not on different days or hours apart). **Please have one person run the demonstration, and the other narrating what is happening in the video.** Alternatively, you can insert explanatory text in post-production.

Analytical derivations. To include the analytical derivations in your report you can type them in any equation editor (the LaTeX typesetting system is recommended) or clearly and neatly write them on paper and use a scanner (least preferred method).

Submission. Compress all code you wrote for the assignment into a single ZIP archive with name `<Last1><First1>-<Last2><First1>-hw<Number>.zip`, where `<First1>`, `<First2>` and `<Last1>` , `<Last2>` are the first and last names of the group, and `<Number>` is the

homework number (for instance, `TronRoberto-BeeVang-hw1.zip`). Submit the report, the video and the ZIP archive through Blackboard. Both members of the group need to submit a copy of this material. Please refer to the Syllabus on Blackboard for late homework policies.

Grading. Each question is worth 1 point unless otherwise noted. Questions marked as **optional** are provided just to further your understanding of the subject, and not for credit. If you submit an answer I will correct it, but it will not count toward your grade. See the Syllabus on Blackboard for more detailed grading criteria.

Maximum possible score. The total points available for this assignment are 15.5 (15.0 from questions, plus 0.5 that you can only obtain with beauty points). Points for the video questions are assigned separately on Blackboard.

Hints Some hints are available for some questions, and can be found at the end of the assignment (you are encouraged to try to solve the questions without looking at the hints first). If you use these hints, please state so in your report (your grading will not change based on this information, but it is a useful feedback for me).

Use of external libraries and toolboxes All the problems can be solved using the basic ROS Python facilities. You are **not allowed** to use functions or scripts from external libraries or packages unless explicitly allowed.

Update the software for your robot

Before starting the assignment, follow the directions on the Blackboard Wiki page *"ROSBot/Updating the provided software"*.

General programming approach

For this homework, you will be asked to follow a modular programming approach. Instead of directly building ROS nodes, you will build modules with functions that implement specific tasks, and test functions that check their functionalities. You will then import these module into ROS nodes. The amount of programming in this approach is somewhat higher, but you will be more confident that each part of your code works as expected before combining all the parts together.

Using the camera

The nodes for the camera can be started using the provided `camera.launch` launch file. You will need to open also a VNC connection to see images (without a valid VNC session, the camera nodes will fail to launch). See the ROSBot Wiki for details on how to setup the VNC connection.

Problem 1: Color-based image segmentation

In this problem you will first build a training/testing set from an image, manually create a decision tree classifier, and apply the classifier on new images. Some of the functions for this problem have been provided to allow you to focus on the most important parts of the homework. The corresponding questions are marked with the label **provided**, and the functions can be found in the provided file `image_processing.py`. You can try to

re-implement the provided functions as an exercise, but you will not receive credit during grading.

Test track For this homework you will have free access to the lab in B14. You can use the already made tracks for your work. Alternatively, there is green tape available to make your own track. The tape is in limited quantity, so use it judiciously, and be respectful of the room and the needs of your classmates.

Question 1.1 (provided). The following function is a utility to display patches of an image

> File name: `image_processing.py`
> Header: `img_patch_show(img,box,window_name)`
> Input arguments
>
> - `img` (type `NumPy array`): a color image.
> - `box` (dim. $[1 \times 4]$, type `List`): a vector specifying a box in `img` with upper left corner at `box[0]`, `box[1]` and bottom right corner (included) at `box[2]-1`, `box[3]-1` (the negative offset is to be consistent with the general Python subscript convention).
> - `window_name` (type `string`): the name of the window to use to show the patch.
>
> Description: Shows the image patch specificied by the argument `box` in a window.

Question 1.2. In this question you will make a script that saves all the pixels of a given patch of an image to a CSV file. You will then use this function to build a training and a test dataset, each one with positive and negative examples.

> File name: `image_processing.py`
> Header: `img_patch_save_csv(img,box,file_name)`
> Input arguments
>
> - `img` (type `NumPy array`): a color image.
> - `box` (dim. $[1 \times 4]$, type `List`): a vector specifying a box in `img` with upper left corner at `box[0]`, `box[1]` and bottom right corner at `box2-1`, `box3-1` (the negative offset is to be consistent with the general Python subscript convention).
> - `file_name` (type `string`): the name of the CSV file to which to save the data.
>
> Description: Saves the pixels in the image patch specificied by the argument `box` in a CSV file having three columns (one for each channel of the image) and one row for each pixel in the box.

Question 1.3. Place the robot on the test track, so that the line is visible in the middle of the camera frame. Take a picture using the `hello_camera.py` script that you used at the beginning of the semester.

Create a function as follows

> File name: `image_processing.py`
> Header: `segmentation_prepare_dataset()`
> Input arguments

- None.

Description: This function first loads the image you captured from the camera, then it uses `img_patch_save_csv()` to save four files that you will use to setup and test your classifier:

- `training_positive.csv`, `test_positive.csv` that contain pixels only from the line.
- `training_negative.csv`, `test_negative.csv` that contain pixels only from the background.

You need to pick boxes to pass to `img_patch_save_csv()` so that the conditions above are verified, and the boxes for the four files should not have any overlap.

Add a call to `segmentation_prepare_dataset()` under the main section of `image_processing.py` so that the function gets executed when `image_processing.py` is executed as a script. Run the script.

Question 1.4. **optional** To make the classification easier, transform the image to the HSV colorspace before saving the values in the CSV files.

Question 1.5. Load the files `training_positive.csv` and `training_negative.csv` in Matlab or other software, and make a 3-D scatter plot of the two classes. Devise a set of rules that separates the two classes (e.g., values of the green channel should be greater than a threshold), and state these rules in your report.

Question 1.6. In this question you will implement the rules you devised into a decision tree.

File name: `image_processing.py`
Header: `pixel_classify(p)`
Input arguments

- `p`: A one-dimensional NumPy array or list with the three color values for a pixel.

Returns: `score`: A numerical value containing the result of the classification.
Description: The function returns, according to the set of rules you defined, `score=-1.0` if the pixel is part of the background, and `score=1.0` if it is part of the line. Use nested if-then conditionals to implement your decision rules.

Question 1.7. After building the classifier, you need to test its performance.

File name: `image_processing.py`
Header: `pixel_classify_testing()`
Description: The function loads the content of the files `testing_positive.csv` and `testing_negative.csv`, calls the function `pixel_classify()` on each point, and finally displays on screen the following statistics:

- Number of points in each file.
- Number (or percentage) of false positives.
- Number (or percentage) of false negative.

Add a call to `pixel_classify_testing()` under the main section of `image_processing.py` so that the function gets executed when `image_processing.py` is executed as a script. Run the script, and include the statistics that you obtain in your report.

Question 1.8. optional Run the previous question on testing sets from novel images with different illumination conditions.

Question 1.9 (provided). The following function extends `pixel_classify()` to work on entire images

> File name: `image_processing.py`
> Header: `img_classify(img)`
> Input arguments
>
> - `img` (type `NumPy array`): A color image.
>
> Returns: `img_segmented`: A **color** image that contains the results of the segmentation.
> Description: The function runs `pixel_classify()` on each pixel of `img`, and sets the corresponding pixel in `img_segmented` to white if it is part of the line, and black if it is part of the background.

Question 1.10 (provided). The following utilty function draws a vertical line across the image

> File name: `image_processing.py`
> Header: `img_line_vertical(img,x)`
> Input arguments
>
> - `img` (type `NumPy array`): A color image.
> - `x`: The x coordinate of the line.
>
> Returns: `img_line`: A color image that contains the original image with the line overimposed.
> Description: Use `cv2.line()` to draw the line. The returned image is a copy to avoid modifying the original.

Question 1.11. Make a function that computes, in a robust way, the horizontal centroid of all the white pixels.

> File name: `image_processing.py`
> Header: `img_centroid_horizontal(img)`
> Input arguments
>
> - `img` (type `NumPy array`): A color image.
>
> Returns: `x_centroid`: The median of the x coordinates of all white pixels.
> Description: This function should accumulate the x coordinates of all white pixels in the image, and then return the median of this list (see `numpy.median` for this last operation). You may assume that the image contains only pixels that are either white or black.

Question 1.12. Modify the main section of `image_processing.py` to include the following sequence of operations on the sample image that you already used to build your dataset:

1) Run `img_classify()` to segment the line from the background.

2) Run `img_centroid_horizontal()` to compute the horizontal centroid of the line.

3) Run `img_line_vertical()` to add a line on the segmented image at the computed centroid.

Display the results (both the original image and the segmented image with the centroid line). Run the script, and include a screenshot of the result in your report.

Question 1.13. You are now ready to make a ROS node to perform real-time extraction of the centroid of the line from the images acquired by the camera of your robot.

> File name: `image_segment_node.py`
> Subscribes to topics: `raspicam_node/image/compressed` (type `sensor_msgs/CompressedImage`)
>
> Publishes to topics:
>
> - `/image/segmented` (type `sensor_msgs/Image`)
> - `/image/centroid` (type `geometry_msgs/PointStamped`)
>
> Description: This node should import the file `image_processing.py` to use all the functions that you have developed so far. For each new image received from the camera, the node should perform the same operations described in Question 1.12, but without visualization of the results. Instead, the node should:
>
> *1)* Publish the segmented image with the line on the topic `/image/segmented`.
>
> *2)* Insert the computed horizontal centroid in the `x` field of a `PointStamped` message, add the current time to the `header.stamp` field, and then publish the message on the topic `/image/centroid`

You can use the provided file `image_repeater.py` as a starting template for the node.

Note: this node should produce a new centroid at least a few times a seconds. The faster the processing, the better the closed loop control (Question 2.5) will probably work. You can check the frequency of the messages published on `image/centroid` using `rostopic hz` from the command line.

You can speed up the centroid computation by down-sizing the image and by cropping it only to a narrow band at the top of the image (see the message on the Discussion board for details). This should be performed in the callback for receiving the images.

Question 1.14 (video). Using multiple terminals, run the `image_segment_node.py` node, together with a `image_view` node to see the contents of the topic `/image/segmented`, and a `rostopic echo /image/centroid` command to see the computed centroid. Move the robot and check that the node is able to track the line robustly and without too much delay.

Problem 2: Proportional-derivative control for line following

In this problem you will create a proportional-derivative (PD) controller that will be used to guide the robot along the line. The function specifications below will use global variables to maintain memory of past values, if necessary. However, you can optionally put all the functions inside a class.

Question 2.1. The first step is to build a proportional controller. It is a rather trivial function, but we will build it to make our code organization consistent.

> File name: `controller.py`
> Header: `proportional(x,r,kp)`
> Input arguments
> - `x`: The current state of the system.
> - `r`: The reference value.
> - `kp`: The gain.
>
> Returns: `u`: The control $u = -k_p(x - r)$.

Question 2.2. The second step is to build a derivative controller. With respect to the proportional controller, it requires the absolute time values to be able to numerically compute the derivative of the signals.

> File name: `controller.py`
> Header: `derivative(x,r,kd,time)`
> Input arguments
> - `x`: The current state of the system.
> - `r`: The reference value.
> - `kp`: The gain, a scalar number.
> - `time`: The time, in seconds, at which the values of `x` and `r` refer to.
>
> Returns: `u`: The control produced by the derivative controller, approximating $u = -k_d \frac{\mathrm{d}}{\mathrm{d}t}(x - r)$.
> Description: First, we define the error variable `e=x-r`. The function should use variables `e_previous` and `time_previous` to store the values of the error and time from the previous call. Then, the function should compute an approximation to $\frac{\mathrm{d}e}{\mathrm{d}t}$ as `de=(e-e_previous)/(time-time_previous)`. The control is then computed as `u=-kd*de`.

Question 2.3. Add the following tests to the main section of the file `controller.py`, so that they are run when `controller.py` is run as a script. For each test, print both the expected and actual result on screen. In the tests, you can use arbitrary values for `x`, `r`, etc.

1) `proportional(x,x,kp)` should return zero for any value of `x` and `kp`.

2) `proportional(x,r,kp)` should return a negative number when `x` > `r` and `kp` > 0.

3) Calling `derivative(x,r,kd,0)` followed by `derivative(x+a,r+a,kd,1)`, should return zero for the second call, independently of the value of `a`.

4) Calling `derivative(x,r,1,0)` followed by `derivative(x,r+a,1,1)`, should return `a`.

Run the script and include a copy of the output in your report.

Question 2.4 (optional). Make the test values used in the previous question random (i.e., change at every run).

Question 2.5. We can now use the previous functions to build a node that aims to command the speed of the robot to make sure that the line is always at the center of the image.

File name: `pd_line_control_node.py`
Subscribes to topics: `/image/centroid` (type `geometry_msgs/PointStamped`)
Publishes to topics: `motor_twist` (type `geometry_msgs/Twist`)
Description: As initialization, import the file `controller.py`, and create a variable with the reference value `r` set to half the size of your image. The goal of the controller is to maintain the position of the values of the line centroid (given by the field `x` of the `PointStamped` messages from `image/centroid`) close to the reference `r`. Each time a new centroid value is received, the node should compute a new value of the control `u` by summing the two contributions of the proportional and derivative controller. The time information for the derivative controller can be extracted using the `.to_secs()` method of the `header.stamp` field of the `PointStamped` message. After computing the control, prepare a `Twist` message where the linear speed is set to a constant value `lin_speed`, and the angular speed is set to the computed control `u`. Publish this message to the topic `motor_twist`.

For this question, you should start by setting `kp=1e-3`, `kd=0`, and `lin_speed=0.5`. You will tune these values in one of the next questions.

Question 2.6 (video). Holding the robot in your hand or on an raised object (so that the wheels do not touch the ground), run the `pd_line_control_node.py` node together with the `motor_command.py` node from previous homework assignments. In your video, show that when the line of the test track appears in the left side of the image, the right wheel turns faster than the left wheel (so that robot is trying to turn left), and that the opposite is true if the line appears in the right side of the image. While you demonstrate this, show also the images from the topic `image/segmented` and the values published on `motor_twist`

Question 2.7. Point the camera as low as possible without having any occlusion from the chassis of the robot. Then, you can tune the parameters of the controller on a test track with the following procedure:

1) Start the robot on a straight part of the track, and experiment by changing the value of `kp`. If `kp` is too high, you will see instabilities in the form of oscillations even on a straight segment of the track (these are due to an overcompensation by the controller coupled with the delays introduced by the vision system). If `kp` is too low, the robot will not be able to make the turns. Choose the highest value of `kp` that does not cause instabilities.

2) Experiment with different values of `lin_speed`. The robot should be able to make almost any turn if `lin_speed` is low enough. Choose the highest value of `lin_speed` that allows you to perform most turns, but fails in the sharpest, most challenging turns.

3) Try to increment the value of `kd` (e.g., with increments around `1e-4`) . This should allow the robot to be more successful at sharp turns (the robot should "anticipate" the turns as soon as the centroid of the segmented line moves from the middle). This, in turn, should allow you to increase also `lin_speed`. However, values to high of `kd` will make the robot oscillate and loose the track.

4) Try to further tune all parameters to maximize the value of `lin_speed` while being successful on all turns.

Question 2.8 (video). Show the robot performing at least one lap of a test tracks.

Hint for question 1.1: The function

Hint for question 1.5: You will probably need to specify six rules, i.e., upper and lower bounds on the values for each channel.