GUI programming (III): A GUI for a robot path planner

Abstract
Now that we have a good background with PyQt, we will implement the GUI for a simple robot motion planner based on artificial potential fields. The idea is that the graphical interface might assist a robotics researcher or developer to explore, refine or improve the method. We assume we have already carried out an analysis of user needs as part of the process of discovering requirements, and have therefore the list of user tasks our interface must support. Certainly, this still leaves us a lot of freedom in design and implementation terms, and we should take a pragmatic approach to balance usability and development effort.

Keywords
PyQt • GUI • user tasks • task priorisation • interface design • wireframes/mockups • prototypes • usability • evaluation

Contents

1. Artificial potential fields

Path planning refers to finding intermediate positions from a start robot configuration/position to a goal one, while avoiding collisions with obstacles. To address this problem, we will make a lot of simplifications since it is not the purpose of this lab to learn about the problem and methods, but rather to practice interface design and programming —while using a hopefully motivating example.

Both our robot and our obstacles will be circles. As the path planner, we will use a simple method based on the technique of Artificial Potential Fields (APFs). In essence, with APFs the robot moves under the influence of an attractive force towards the goal and repulsive forces exerted by obstacles for the robot to stay away from them. The resulting force dictates the direction of motion.

We have a basic implementation of the method in `potentialFieldPathPlanner.py` that we can use as our "backend" solution so that we can focus on the "frontend", our GUI-based application. Look at the `PotentialFieldPathPlanner` class and learn how it can be used. It is easy to see how we can provide the planner with the robot and its desired goal position, manage obstacles, and run the planner so as to get a path. We have some parameters to configure, such as the maximum number of iterations, the area of influence of obstacles, and the attraction and repulsion factors. Run the program to get a gist of how it works; the program displays the path using *matplotlib*, but not the robot or the obstacles. We will do that in our PyQt program.

2. User tasks

As you know, when we want to design and build any program or product and its interface, it is crucially important to have a clear idea of its end users (sometimes other stakeholders as well) and their true needs. In our case, the users would be roboticists, either researchers or developers, who are interested in testing path planners to help them fine tune their algorithms or implementations. This scenario might be different to, for instance, a high-school student who is just exploring Robotics

problems at a general level and mainly for leisure purposes. Certainly, different user profiles (may) have different needs, and we should design our interface accordingly.

Let's assume we have already performed a careful process of discovering user needs and system requirements and, as a result, we have come up with the following user tasks:

T_1: (a) Define and (b) edit a robot

T_2: (a) Define and (b) edit a start position

T_3: (a) Define and (b) edit a goal position

T_4: (a) Create, (b) remove, and (c) edit obstacles

T_5: Set and change parameters of the planner

T_6: Run the planner

T_7: Visualize (a) the initial position, (b) the goal position, (c) the robot at current position, (d) the obstacles, (e) the resulting path and the robot at intermediate positions along that path

Remember, these are the set of user activities our interface should support. There are many ways for such support, and carefully deciding about how to do this the best is what interaction design is essentially all about. Note that different sets of tasks may result in different designs. For instance, we do not have a task for saving or loading environments, or for displaying several paths obtained with different settings. These other tasks are likely to be useful in practice as well, but we omit them for simplicity.

3. Design

Prioritize the tasks. As a next step, it can be useful to use some guideline, such as the "frequency $\times$ popularity" heuristic, to decide how efficient and visible the corresponding interface aspects should have to properly support the set of user tasks. Note that in a real project, we might need to conduct further investigation to find out the actual frequency and popularity of the tasks. Instead, for this lab project, use your background and intuition to motivate a reasonable and sensible choice.

Interaction diagram. Map the frequency and popularity of the user tasks into design constraints such as efficiency and visibility (from the user's perspective). Accordingly, outline which "screens" or *display units* (main window, dialog window, etc.) we might need. If possible, do not think yet of particular widgets but just the diagram of transitions between these display units. You can annotate these units with the tasks they relate to; this might help us make sure all tasks are properly covered. Draw arrows between the display units to indicate the possible navigation possibilities (transitions).

Paper prototypes. After you are happy with this interaction diagram, you can think of particular interface elements and interaction possibilities. For instance, what is a better widget for setting the attraction factor, an entry box, a slider or other? Why? Remember: your best guess is not enough; design is iterative by nature.

Drafting, refining, evolving and documenting. Draft one (or multiple!) prototype(s), in the form of wireframes or mockups. It is good that you think of the possibilities of Qt, as well as your own abilities, the difficulty of one solution over the other, and so on. In the end, we should choose a smart balance between an ideal design and the practical constraints we have such as technology limitations, time pressure, or lack of knowledge/skills. But it is good that you do not discard your good design ideas simply because you find they are too costly or difficult to implement for the time budget (or any other constraints in all kinds of resources). Then, you can device a compromise solution between the ideal design you initially considered and a more realistic solution. All the thought and effort involved in all these steps are well worth it, both in our academic learning context and in real-world projects. Remember: design consists of taking (informed) decisions all the time; the process is as important (or more!) than the end product, and more so for learning purposes.

4. Implement

Now implement a PyQt application based on your prototype(s) and your design knowledge. You are provided with a basic `main.py` in the folder *GUI* that you *may* use as a starting point (but you do not have to). This program has some import statements (including the classes related to path planning) and creates and displays an empty square window.

Although it is important to stick to your interface design, you still have some flexibility. For instance, if you find some unexpeced difficulties during the implementation, you may prefer an alternative solution, but document the rationale behind your

decision and pros and cons of the different approaches. Note also that a full implementation of the GUI aspects of all the tasks might be too costly for our scheduled time. Again: keep iterating your designs as your understanding and resources demand. Therefore, focus your effort on learning and doing a quality job even if you do not implement everything. But the program should implement the full path planning procedure; if some aspects do not have an interactive part, they should be present anyway, even if hard coded.. Use tradeoffs wisely and do not forget to document your the process, the ideas and decisions. They can be helpful later to understand why you built the interface as you did, and can provide you with suggestions for future improvements or products. Besides that, in our academic context the lecturer would like (or rather, need) to understand what is behind your design and implementation.

Regarding the software development, try to follow the good practices you may know regarding object orientation, decomposition, factorisation, modularity, code reuse, no magic values, etc.

5. Evaluate

During the implementation you will have gained insights and you may have a better idea of the pros and cons of your design. This may reminds us the task-artifact cycle. It does not matter if your interface is not perfect; it does not have to. We are learning. But it is *of paramount* importance that you are aware of the good and not so good aspects. As you interact with your own implementation, you will still find more and more ideas for improvement. Write down all usability issues you find and how alternative designs might correct them. Bear always in mind the usability principles and evaluation methods we have been studying.

The paragraph above is implicitly considering a final evaluation of your final prototpe or evaluation. This is not bad, but remember that evaluation is part of the iterative user-centered design process. This means that we can perform different types of evaluations at different stages using prototypes of different fidelities with different purposes in mind. Certainly, our time budget is limited; so make a wise choice of which evaluation technique you decide to use, when, and why. Do not understimate the *why*: each evaluation should have a main purpose in mind.

6. Additional activities/thoughts

1. We assumed (actually, pretended) that a proper analysis of user needs have been previously performed. What activities might have we carried out for such an analysis in the context of this particular problem? What people might you consider as possible users or as experts?

2. The listed tasks above are like “essential” (core) tasks. (By the way, do you feel we missed any *core* task?). Our users could not work effectively (or not at all) with an interface which does not support these core tasks. However, we might have discovered other tasks which are not critical, but can also be “important”, and even other tasks which might be just “nice to have”. Can you think of examples of tasks in these two other categories (*important* and *nice-to-have*)? In real-world projects, it can be important to document them so that we might include support for some of these tasks in the future if we have time and resources to build an improved version of our software or product. So, if we do not keep them, we might have forgotten about them (a pity after the requirement discovering and design efforts we made!). It can be argued that our prototypes might even be prepared to already support (some of) those other tasks, even if they are not implemented as of now.

3. Did you have to change some aspect of the “backend” code to facilitate some aspects of your application? If so, which ones and why? Note that an Application Programming Interface (API) is also a form of interface, of a different nature, but an interface after all. If the software programmers who designed and built the API did not think of their target users (here, other programmers), then usability problems are possible too. Interfaces are everywhere, and learning about usability can help us be better professionals (and persons), because we care about other people, whether users of our technology, work mates, and others we interact with one way or another in our training, professional and personal lives.