Exercises for the BeeWi Camera Buggy SDK

Here are some exercises that will work through using the BeeWi Camera Buggy SDK. You'll need to set up the *buggy.py* and the *video_thread.py* as shown in the documentation.

Good luck with them, and don't worry if you find them hard – they're meant to be a challenge! The exercises are just guidelines – feel free to tailor them to your own needs. It will help a lot if you do them all in the right order – as the levels get harder, they will use key features from previous levels.

There are hints and tips on the final page. Try not to use them unless you're really struggling!

Exercises

Beginner - Moving the Buggy around

Level 1

Write a program that moves the Buggy forwards turning left, and then backwards turning right.

Level 2

Write a program that moves the Buggy so that it ends up facing in the opposite direction. You can use as much space as you need to turn it around, but the smaller space you use the better.

Level 3

Imagine the Buggy has a pen attached underneath it (or better yet, connect a pen to it), and write a program to make the Buggy write out your first name.

Intermediate – Using the built-in functions

Level 1

Make the Buggy slowly creep forward until it is stuck, and then stop.

Level 2

Write a program that makes the Buggy dance – make it move in random directions for random periods of time. *Be careful – it will probably hit into walls a lot, so don't make it go too fast or you'll break it!*

Level 3

Make the Buggy slowly creep forward until it is stuck, then reverse in a random direction and carry on creeping forward.

Advanced – Doing it yourself

Note: These are much more difficult than the previous tasks. You'll need to make use of the Python OpenCV distribution to help you with these tasks!

Level 1

Implement the Random Search algorithm, giving the program an image to search for, and stop when it finds it. If it helps, display the Buggy's camera in a small window.

Level 2

Give the Buggy two images, perhaps of simple objects or signs – for example, a circle and a cross. If the Buggy sees one image, it should drive towards it – if it sees the other, it should drive away from it. Make your program more advanced by including steering towards / away from the images as well.

Level 3

Make the Buggy follow large red objects, while reversing away from large green objects.

Hints and Tips

Beginner

Level 1

To do this you'll need to make use of the following functions: *forward(), backward(), turnLeft() and turnRight().*

Level 2

The program you made for Level 1 will help a lot for this task. What direction did the Buggy end up facing after you ran that program? Using loops will make life easy for everyone here.

Level 3

You won't need many loops (if any) for this task. Simply use the same movement functions as before, and make the Buggy move to follow the letters in your name. Always test your program – enough trial and error will eventually result in success!

Intermediate

Level 1

This may sound hard, but if you use the built in method *isStuck()*, this task is a doddle. You'll need a loop to keep moving forward, and then you'll need to check whether the Buggy is stuck – if it is, you'll want to stop looping and stop.

Level 2

Python has a library called *random* which will do most of the work for you in this one. Within a loop you'll need to randomise the speed, direction and value of the direction. Maybe you could wait for a random amount of time, using the *wait()* function.

Level 3

Similar program to the one you wrote for Level 1, but using the random movement from Level 2. The Buggy has a built in function called *randomisedRecovery()*, but try to write your own!

Advanced

Level 1

This will be quite hard, but you've done most of the work in the Intermediate section. First, you'll need to supply the program with an image, using *cv2.imread()*. You'll then want to look up Template Matching, to check if the image exists within the Buggy's vision, to a reasonable degree of accuracy. Aside from that, the program will be very similar to the Intermediate Level 3.

A working Random Search algorithm using the SDK has been implemented in randomSearch.py, but only have a look if you're really struggling.

Level 2

You'll need to use the image reading function twice for both images, and you'll need to use template matching again. To include steering, you'll need to take the x position of the found image, and convert this into a value you can plug into one of the turning functions. You will also need to handle what the Buggy does if it sees both images at the same time – how about taking the clearer image? Or make the Buggy act differently?

Level 3

Perhaps the hardest challenge! Very similar to Level 2, but you'll need to research Colour Spaces and possibly Contour Features. Separate the Buggy's images into different colour channels, and look for the largest contour. If over a certain size, act accordingly.