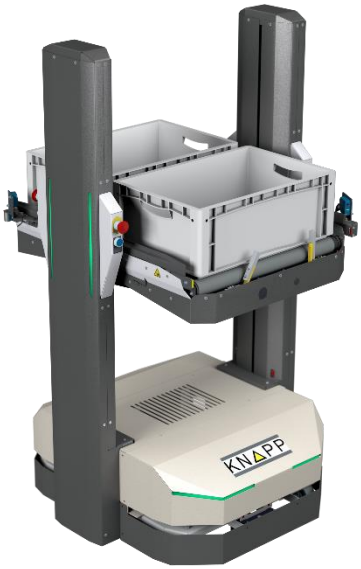
Candidate Evaluation

SW developer for autonomous mobile robot (AMR) systems



Candidate Evaluation

Candidate Evaluation

V1-00

12/2020

en

1 General

The aim of this coding example is not to find the most creative, most complex or even the single best solution but rather to get an initial understanding of the coding habits and style of the candidate under evaluation. Therefore, you should not strive to provide the most sophisticated or complex solution but rather try to solve the example the way you would normally approach a coding problem of similar complexity. However, we note that we will not only evaluate the code as is, but the overall approach that was taken and the whole project that is provided, including the required build files and possibly required included libraries. The first most important things we will check is '*does it build, does it run*' so make sure that these two requirements are met under any circumstance. Under any circumstances, the solution must be prepared by yourself and should not be copied from other sources, since we may discuss the provided solutions in a later interview phase. The example is constructed such that you should not require more than 4-8hours to implement and test everything. We encourage you to provide the actual amount of time spent to implement the example, so we can also discuss this metric with you.

2 Basic Conditions

- The code must be written using C++ programming language.
- The code must compile and run under Ubuntu 20.04 LTS but in general should written as platform independent code.
- The solution must be provided in a public Gitlab/Github repository. The link should be sent to us for validation of the results.
- The usage of widely used libraries such as Boost is not prohibited, although it should not be required to use such libraries.
- We encourage you to write unit tests and maybe even include these in Gitlab's CI pipeline to both make sure your code will perform as intended and to convince us of your correct implementation.
- In general, you can assume correct usage of the program. Meaning that no unspecified input sequence or invalid file will be provided intentionally. However, you should make sure that your program does not crash on simple user errors such as providing invalid input data such as an unspecified order_id.
- All data files required to test your program will be provided together with dis document.

3 Task

1) Create an app named ‘OrderOptimizer’.
2) The app should have an interface for being able to receive following topics: ‘*currentPosition*’ and ‘*nextOrder*’.
    a. The topic ‘*currentPosition*’ provides the AMR’s current position:

        double x
        double y
        double yaw

    b. The topic ‘*nextOrder*’ indicates an order assigned to the current AMR and is defined as follows:

        uint32 order_id
        string description

    The “interface” is of your choice, can be REST, mqtt, whatever you are comfortable with.

3) The app should take an argument that specifies an absolute path to a directory containing files.
4) The folder contains two subdirectories ‘*orders’* and ‘*configuration’* which contains the following files respectively:
    a. The ‘*orders’* folder contains several yaml files (one for each fictional day of orders) in the following format:

```
- order: 1234
  cx: 123.4
  cy: 234.5
  products:
        - 56
        - 78
- order: 2345
  cx: 456.7
  cy: 678.9
  products:
        - 99
```

    Please not that in this example, order 1234 contains 2 products, namely product 56 and 78, while order 2345 contains the single product 99. The destination to which the order should be delivered is also defined for both orders.

b. The ‘configuration’ folder contains a yaml file that provides product specific configurations stating how a product can be manufactured. For each product, a list of components needed to manufacture the product is specified. For each part, a pickup location is further specified. An example yaml file would look like the following snippet:

```
- id: 1
  product: “56”
  parts:
          - part: “a”
            cx: 120.2
            cy: 240.8
          - part: “b”
            cx: 420.8
            cy: 110.2
- id: 2
  product: “78”
  parts:
          - Part: “x”
            cx: 85.1
            cy: 10.2
```

In this example, product 56 consists of two parts, ‘a’ and ‘b’ at two locations, while product 78 consists of only one part ‘x’. Please note that the part names are arbitrarily chosen and are not relevant for this coding example.

5) On each order that is received via the ‘nextOrder’ topic, the app should parse all files in the folder ‘orders’ to find information for that given order. Due to performance reasons, parsing of multiple files must be done in separate threads, one per file. For the purpose of this fictional example, we assume that the files change regularly, so parsing them each time is required, although we are aware that this is neither a good idea nor best practise.
6) After obtaining the products required for an order, your app should check which parts are required for the respective products using the yaml file in the folder ‘configuration’. We assume that this configuration does not change regularly, so you may parse it once when starting the app.
7) Having obtained all products and necessary parts for a given order, your app should determine the geometrically shortest path that the AMR needs to travel to collect all parts. We assume a completely empty map without any obstacles for this purpose. The current position of the AMR should be considered when calculating this shortest path.
8) It should then print the calculated path in a form looking like the following example:

Working on order 1234 (<print description from message here>)
1. Fetching part ‘a’ for product ‘56’ at x: 120.2, y: 240.8

2. Fetching part ‘x’ for product ‘78’ at x: 85.1, y: 10.2
3. Fetching part ‘b’ for product ‘56’ at x: 420.8, y: 110.2
4. Delivering to destination x: 123.4, y: 234.5