Project description—Redundant inspection planning

Background

In this work we investigate the problem of *inspection planning*, or coverage planning [2, 7]. Here, we consider the specific setting where we are given a robot equipped with a sensor and a set of points of interest (POI) in the environment to be inspected by the sensor. The problem calls for computing a minimal-length motion plan for the robot that maximizes the number of POI inspected. This problem has a multitude of diverse applications, including industrial surface inspections in production lines [11], surveying the ocean floor by autonomous underwater vehicles [3, 8, 9, 12], structural inspection of bridges using aerial robots [4, 5], and medical applications such as inspecting patient anatomy during surgical procedures [10].

Specifically, we are interested in improving Incremental Random Inspection-roadmap Search (IRIS) [6], a new asymptotically-optimal inspection-planning algorithm. IRIS computes inspection plans whose length and set of inspected points asymptotically converge to those of an optimal inspection plan. IRIS incrementally densifies a motion-planning roadmap using sampling-based algorithms, and performs efficient near-optimal graph search over the resulting roadmap as it is generated.

Problem statement

Motivated by recent work on resilient robotic autonomy [1], in this project we consider the problem of *redundant inspection planning*. Specifically, given a set of points of interest and some parameter k, we want to compute a path such that every point of interest is seen from at least k different configurations[1]

Assume that k is small (say $k \leq 3$) and suggest a modification of the search algorithm employed by IRIS. Once you suggested such an algorithm implement it and test it on the different scenarios provided in the code base.

Code base

The code to be used is located at `https://github.com/UNC-Robotics/IRIS`. If you are using Ubuntu or macOS, then installation is straightforward following the installation instructions. If you are a Windows user, a detailed installation manual can be found in the appendix to this document.

The code is split into two applications `build_graph` and `search_graph`. The first generates the (implicitly-defined) RRG while the second runs searches over the corresponding inspection graph. This is done in order to allow for changes in the search algorithm (e.g., comparing different approximation parameters) without the need to re-generate the RRG which may be time consuming and will not allow an apple-to-apple comparison. Thus, the outcome of `build_graph` is three files that contain (i) the set of

[1]We can think of more realistic requirements where we want to compute a path such that every point of interest is seen from at least k different configurations that are at least δ apart but this is beyond the scope of the project.

configurations generated (ii) the corresponding vertices together with the set of points viewed from each vertex and (iii) the set of potential edges (some may be in collision and this is stated in the file). Each file also contains the time required to generate the data (e.g., collision detection time or time required to compute the points covered). Then, in `search_graph` these times are accounted for when searching for a solution. More details can be found in the readme page `https://github.com/UNC-Robotics/IRIS`.

There are three models that can be used (i) A planar 5-link manipulator inspecting a planar environment, (ii) a CRISP robot inspecting a collapsed lung, (iii) a quadrotor inspecting a bridge. More details can be found in the readme page `https://github.com/UNC-Robotics/IRIS`.

The code uses several external libraries (e.g., boost, eigen and OMPL). Of specific interest to us is OMPL—the Open Motion Planning Library which consists of many state-of-the-art sampling-based motion planning algorithms. In the implementation of IRIS, OMPL is used to represent the different C-spaces to implement the underlying RRG (taking into account collision detection, nearest-neighbor search, sampling and more).

References

[1] Kostas Alexis. Towards a science of resilient robotic autonomy, 2020.

[2] Randa Almadhoun, Tarek Taha, Lakmal Seneviratne, Jorge Dias, and Guowei Cai. A survey on inspecting structures using robotic systems. 13(6), 2016.

[3] Brian Bingham, Brendan Foley, Hanumant Singh, Richard Camilli, Katerina Delaporta, Ryan Eustice, Angelos Mallios, David Mindell, Christopher Roman, and Dimitris Sakellariou. Robotic Tools for Deep Water Archaeology: Surveying an Ancient Shipwreck with an Autonomous Underwater Vehicle. *J. of Field Robotics*, 27(6):702–717, 2010.

[4] Andreas Bircher, Kostas Alexis, Michael Burri, Philipp Oettershagen, Sammy Omari, Thomas Mantel, and Roland Siegwart. Structural Inspection Path Planning via Iterative Viewpoint Resampling with Application to Aerial Robotics. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 6423–6430. IEEE, 2015.

[5] Andreas Bircher, Mina Kamel, Kostas Alexis, Michael Burri, Philipp Oettershagen, Sammy Omari, Thomas Mantel, and Roland Siegwart. Three-dimensional coverage path planning via viewpoint resampling and tour optimization for aerial robots. *Rob. Res.*, 40(6):1059–1078, 2016.

[6] Mengyu Fu, Alan Kuntz, Oren Salzman, and Ron Alterovitz. Toward asymptotically-optimal inspection planning via efficient near-optimal graph search. In *Robotics: Science and Systems (RSS)*, 2019.

[7] Enric Galceran and Marc Carreras. A survey on coverage path planning for robotics. 61(12):1258–1276, 2013.

[8] Nuno Gracias, Pere Ridao, Rafael Garcia, Javier Escartín, Michel L'Hour, Franca Cibecchini, Ricard Campos, Marc Carreras, David Ribas, Narcís Palomeras, et al. Mapping the Moon: Using a lightweight AUV to survey the site of the 17th Century ship 'La Lune'. In *OCEANS-Bergen, 2013 MTS/IEEE*, pages 1–8. IEEE, 2013.

[9] Matthew Johnson-Roberson, Oscar Pizarro, Stefan B. Williams, and Ian Mahon. Generation and Visualization of Large-Scale Three-Dimensional Reconstructions from Underwater Robotic Surveys. *J. of Field Robotics*, 27(1):21–51, 2010.

[10] Alan Kuntz, Chris Bowen, Cenk Baykal, Arthur W. Mahoney, Patrick L. Anderson, Fabien Maldonado, Robert J. Webster, and Ron Alterovitz. Kinematic Design Optimization of a Parallel Surgical Robot to Maximize Anatomical Visibility via Motion Planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 926–933, 2018.

[11] Roberto Raffaeli, Maura Mengoni, Michele Germani, and Ferruccio Mandorli. Off-line view planning for the inspection of mechanical parts. *International Journal on Interactive Design and Manufacturing (IJIDeM)*, 7(1):1–12, 2013.

[12] Maurice A. Tivey, Albert Bradley, Dana Yoerger, Rodney Catanach, Alan Duester, Steve Liberatore, and Hanu Singh. Autonomous Underwater Vehicle Maps Seafloor. *Eos, Transactions American Geophysical Union*, 78(22):229–230, 1997.

Appendix—installation manual using Windows OS

1. **Installing WSL (Windows Subsystem for Linux)** If you use Windows as your OS, WSL is provides an easy way to use Windows IDEs with code built in Linux. Follow instructions described in the following link `https://docs.microsoft.com/en-us/windows/wsl/install-win10`. There, install the Ubuntu 18.04 LTS distribution.

2. **Installing relevant software on your Ubuntu distribution.** IRIS's dependencies can be found in the following link `https://github.com/UNC-Robotics/IRIS`. Here, we provide step-by-step instructions on how to install them. Launch your Ubuntu machine. First time takes a few minutes for updates and installation. You will be prompted to choose a username and password.

   (a) Install git by typing the commands:

   ```
   $ sudo apt-get update
   $ sudo apt-get install git-core
   ```

   Check that this was installed correctly by typing the command

   ```
   $ git --version
   ```

   You should get something similar to the following

   ```
   git version 2.17.1
   ```

   (b) Install gcc. General instructions can be found here:
   `https://linuxize.com/post/how-to-install-gcc-compiler-on-ubuntu-18-04/`. However, you only need to type the following commands

   ```
   $ sudo apt update
   $ sudo apt install build-essential
   $ sudo apt-get install manpages-dev
   ```

   Check that this was installed correctly by typing the command

   ```
   $ gcc --version
   ```

   You should get something similar to the following

   ```
   gcc (Ubuntu 7.4.0-1ubuntu1̃8.04.1) 7.4.0
   ```

   (c) Install boost. General instructions can be found here:
   `https://www.osetc.com/en/how-to-install-boost-on-ubuntu-16-04-18-04-linux.html`.
   However, you only need to type the following commands

   ```
   $ sudo apt install libboost-dev
   $ sudo apt install libboost-all-dev
   ```

   Check that this was installed correctly by typing the command

   ```
   $ dpkg -s libboost-dev — grep 'Version'
   ```

You should get something similar to the following

```
Version: 1.65.1.0ubuntu1
```

(d) Install cmake. General instructions can be found here: `https://vitux.com/how-to-install-cmake-on-ubuntu-18-04/`. However, you only need to type the following commands

```
$ sudo apt-get install cmake
```

Check that this was installed correctly by typing the command

```
$ cmake --version
```

You should get something similar to the following

```
cmake version 3.10.2
```

(e) Install Eigen3. To instal Eigen3, type the following command

```
$ sudo apt-get install libeigen3-dev
```

(f) Install OMPL. General instructions can be found here: `https://ompl.kavrakilab.org/installation.html`. However, you can follow these directions:

Download instalation file from `https://ompl.kavrakilab.org/installation.html` and place it in `c:/tmp` (create folder first). Now, go back to your Ubuntu machine and type the following commands (this may take some time):

```
$ mkdrir Project
$ cd Project
$ mv /mnt/c/tmp/install-ompl-ubuntu.sh ./
$ chmod u+x install-ompl-ubuntu.sh
$ ./install-ompl-ubuntu.sh --python
$ cd ompl-1.4.2-Source/
$ cd build
$ cmake ..
$ make
```

3. **Install** IRIS. We first want to install the code (after going back to the Project directory):

```
$ cd ../../
$ git clone https://github.com/UNC-Robotics/IRIS.git
$ cd IRIS
$ git submodule update --init --recursive
```

Now we download the data used by the algorithm. From your Windows machine, download the data from this link - `https://drive.google.com/file/d/19DGtog4D4hAgwFu1bV_ct0h_n-G4BR1Z/view` and locate is at `c:/tmp`.

In your Ubuntu machine, type the commands

```
$ mkdir data
$ cd data
$ mv /mnt/c/tmp/data.tar .
$ tar -xvf data.tar
$ cd ..
```

We are now ready to build the code

```
$ mkdir build
$ cd build
```

We now need to run cmake. Unfortunately, some times your computer cannot find the ompl files we installed so we need to tell it where ompl is installed. Ideally, the following lines should work

```
$ cmake ..
$ make
```

If you get an error, then this should work.

```
$ cmake -DOMPL_INCLUDE_DIRS=/usr/local/include/ -DOMPL_LIBRARIES=/usr/local/lib/libompl.so
  ..
$ make
```

Here "/usr/local/include/" may be replaced by specific path in your computer, but probably this is the path. Similarly, "/usr/local/lib/libompl.so" may be replaced by specific path in your computer, but probably this is the path.

You can now run the code directly from your Ubuntu distro (make sure you are in the build directory).

For example, to build a graph type

```
$ ./app/build_graph seed num_vertex file_to_write
```

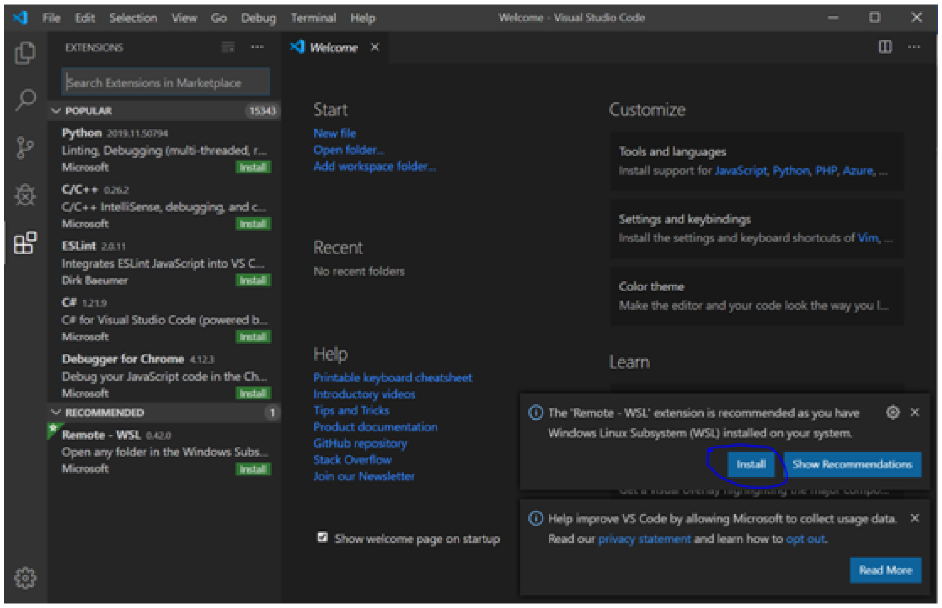
To search the graph type

```
$ ./app/search_graph file_to_read initial_p initial_eps tightening_rate method file_to_write
```

4. **Using the code through Visual Studio Code**

   (a) On your Windows machine, download and install Visual Studio Code from this link `https://code.visualstudio.com/download`.

   (b) Open VS Code and it will offer you to install WSL extension



Linting, Debugging (multi-threaded, r...

C/C++ IntelliSense, debugging, and c...

Integrates ESLint JavaScript into VS C...

C# for Visual Studio Code (powered b...

Debug your JavaScript code in the Ch...

Open any folder in the Windows Subs...

Install support for JavaScript, Python, PHP, Azure, ...

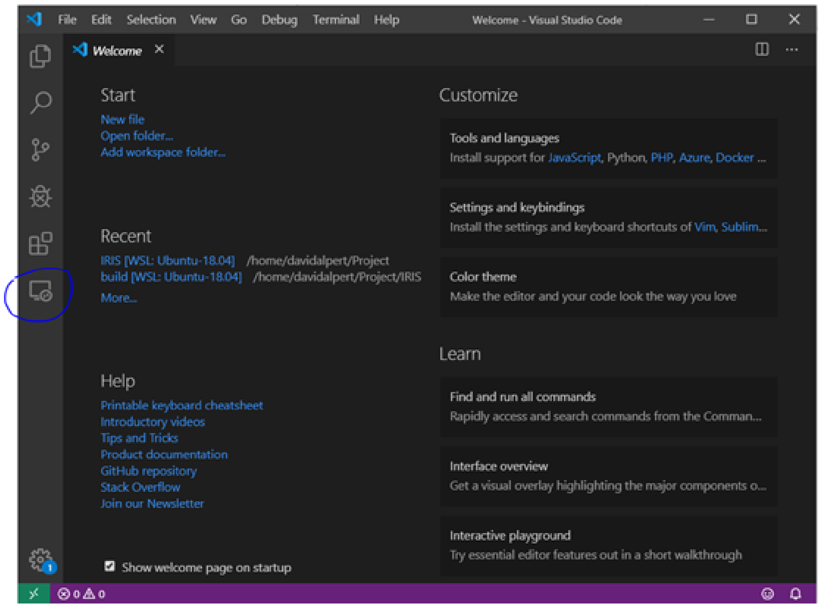
Install the settings and keyboard shortcuts of Vim, ...

Make the editor and your code look the way you l...

The 'Remote - WSL' extension is recommended as you have Windows Linux Subsystem (WSL) installed on your system.

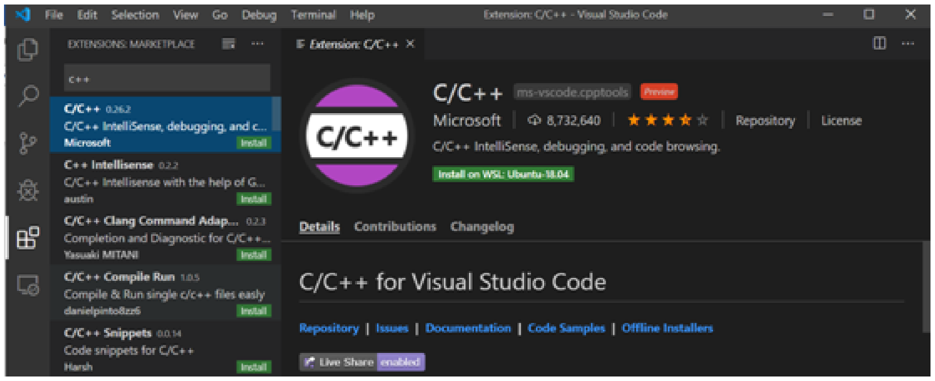
Help improve VS Code by allowing Microsoft to collect usage data. Read our privacy statement and learn how to opt out.

   (c) Push the marked button


File Edit Selection View Go Debug Terminal Help
Welcome - Visual Studio Code
Welcome
Start
New file
Open folder...
Add workspace folder...
Recent
IRIS [WSL: Ubuntu-18.04] /home/davidalpert/Project
build [WSL: Ubuntu-18.04] /home/davidalpert/Project/IRIS
More...
Help
Printable keyboard cheatsheet
Introductory videos
Tips and Tricks
Product documentation
GitHub repository
Stack Overflow
Join our Newsletter
Show welcome page on startup
Customize
Tools and languages
Install support for JavaScript, Python, PHP, Azure, Docker ...
Settings and keybindings
Install the settings and keyboard shortcuts of Vim, Sublim...
Color theme
Make the editor and your code look the way you love
Learn
Find and run all commands
Rapidly access and search commands from the Comman...
Interface overview
Get a visual overlay highlighting the major components o...
Interactive playground
Try essential editor features out in a short walkthrough

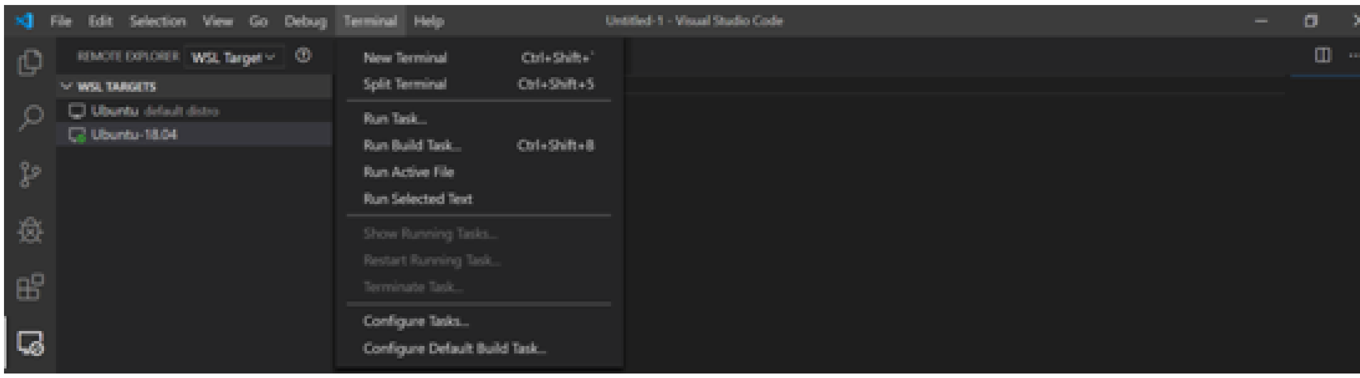
(d) Now visual Studio code will offer you to remote Ubuntu from VSCODE through WSL.

(e) Install extensions for C++


File Edit Selection View Go Debug Terminal Help
Extension: C/C++ - Visual Studio Code
EXTENSIONS: MARKETPLACE
c++
C/C++ 0.26.2
C/C++ IntelliSense, debugging, and c...
Microsoft
Install
C++ Intellisense 0.2.2
C/C++ Intellisense with the help of G...
austin
Install
C/C++ Clang Command Adap... 0.2.3
Completion and Diagnostic for C/C++...
Yasuaki MITANI
Install
C/C++ Compile Run 1.0.5
Compile & Run single c/c++ files easly
danielpinto8zz6
Install
C/C++ Snippets 0.0.14
Code snippets for C/C++
Harsh
Install
Extension: C/C++
C/C++
C/C++
ms-vscode.cpptools
Preview
Microsoft
8,732,640
Repository
License
C/C++ IntelliSense, debugging, and code browsing.
Install on WSL: Ubuntu-18.04
Details
Contributions
Changelog
C/C++ for Visual Studio Code
Repository | Issues | Documentation | Code Samples | Offline Installers
Live Share enabled

(f) You may want to consider the “Pretty” and “Bracket pair colorizer” packages. They are very helpful for writing code in Visual studio code.

(g) Now you can open new terminal of ubuntu throughout VSCode


File Edit Selection View Go Debug Terminal Help
Untitled-1 - Visual Studio Code
REMOTE EXPLORER WSL Target
WSL TARGETS
Ubuntu default distro
Ubuntu-18.04
New Terminal Ctrl+Shift+`
Split Terminal Ctrl+Shift+5
Run Task...
Run Build Task... Ctrl+Shift+B
Run Active File
Run Selected Text
Show Running Tasks...
Restart Running Task...
Terminate Task...
Configure Tasks...
Configure Default Build Task...

(h) From the Terminal Open IRIS's folder.

(i) If you want to run in debug mode with breakpoint you have to install gdb and compile in debug mode by the following command:

```
$ sudo apt install gdb
$ mkdir debug
$ cd debug
$ cmake -DCMAKE_BUILD_TYPE=Debug ..
$ make
```

(j) The last step is to press F5 and choose gdb compiler, and create the relevant `launch.json` file. Lines that should be changed have black background. Remember to change update file to use your username. Here is the file relevant for `build_graph`

```
”version”: ”0.2.0”,
    ”configurations” : [
        {
            ”name” : ”(gdb) launch”,
            ”type” : ”cppdbg”,
            ”request” : ”launch”,
            ”program” : ”$workspaceFolder/debug/app/build_graph” ,
            ”args” : [ ”(1)” , ”(2)” , ”control_file1” ],
            ”stopAtEntry” : ”false”,
            ”cwd” : ”/home/username/Project/IRIS/debug” ,
            ”environment” :[]
            ”externalConsole” : ”false”,
            ”MIMode” : ”gdb”,
            ”setupCommands” :[
                {
                    "description" : "Enable pretty-printing for gdb",
                    "text" : "-enable-pretty-printing",
                    "ignoreFailures" : "false"
                }
            ]
        },
    ]
}
```

Here is the file for `search_graph`:

```
”version”: ”0.2.0”,
   ”configurations” : [
      {
         ”name” : ”(gdb) launch”,
         ”type” : ”cppdbg”,
         ”request” : ”launch”,
          ”program” : ”$workspaceFolder/debug/app/search_graph” ,
          ”args” : [ ”control_file” , ”0.8” , ”0.8” , ”0” , ”0” , ”control_file” ],
         ”stopAtEntry” : ”false”,
          ”cwd” : ”/home/username/Project/IRIS/debug” ,
         ”environment” :[]
         ”externalConsole” : ”false”,
         ”MIMode” : ”gdb”,
         ”setupCommands” :[
            {
               ”description” : ”Enable pretty-printing for gdb”,
               ”text” : ”-enable-pretty-printing”,
               ”ignoreFailures” : ”false”
            }
         ]
      },
   ]
}
```