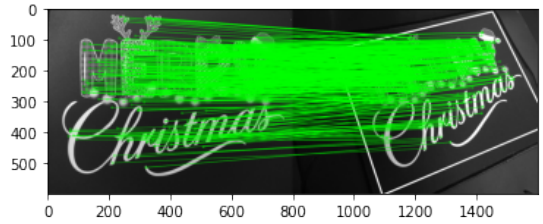
Project 5 - Vission and PDDL

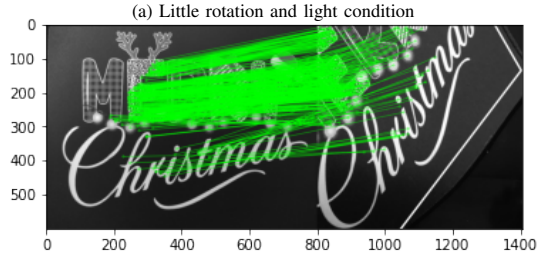
Hoang-Dung Bui,
George Mason University
Fairfax, USA
hbui20@gmu.edu

I. Feature Matching with OpenCV

We have taken 5 photos, and adjust their contrast, light's intensity, color, or crop, empty parts of the images. There are some conclusion after playing with SIFT and the images.



(a) Little rotation and light condition

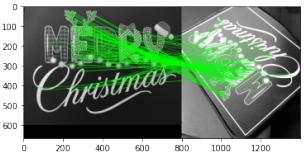


(b) Cropping more than half of the images, and leaving a small common part

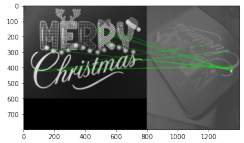
Fig. 1

As we change only little about the view angle and little condition, SIFT and OpenCV still works very well as shown in Fig. 1a. There is 686 matches between two images. Even we cropped the large part of the images, and left a small common part, SIFT still works very well (581 matchings) (Fig. 1b). It is easy for it to detect the feature matching between the two images.

As making significant adjustment on the images such as rotating 90 degree, the number of matching reduces significantly (only 188 matching) 2a. As converting the image into grayscale and opposite rotation, then SIFT's performance degrades as shown in Fig. 2b with only 33 matching.



(a) Gray-scale and opposite orientation



(b) Grayscale and rotating 180 degree

Fig. 2

SIFT's performance get worst as we do all the following: convert the images into grayscale, rotating 180 degree, reduce the contrast to extreme, crop and cover part of images. It is quite extreme as we implement all the adjustments.

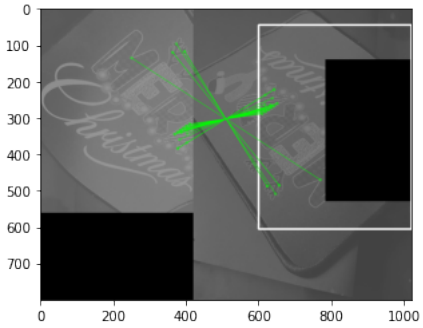


Fig. 3: Converting into Grayscale, cropping, and covering

In overall, SIFT is a powerful tool to detect feature and feature matching. Its performance only gets bad if we change a lot of factors in the images such as convert the image to gray-scale, rotate 180 degree, or change the contrast and light condition.

The code of comparing the match among the images is shown in Fig. 4. We use KDTree algorithm and use Flann matcher to process the features. The features are matched by KnnMatch.

```
# find the keypoints and descriptors with SIFT
kp1, des1 = sift.detectAndCompute(img1,None)
kp2, des2 = sift.detectAndCompute(img2,None)
FLANN_INDEX_KDTREE = 1
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks = 50)
flann = cv.FlannBasedMatcher(index_params, search_params)
matches = flann.knnMatch(des1,des2,k=2)
# store all the good matches as per Lowe's ratio test.
good = []
for m,n in matches:
    if m.distance < 0.7*n.distance:
        good.append(m)
print(len(good))
```

Fig. 4: The code to compare matching

Response: In my opinion, light condition and rotation do not affect much on the feature matching of SIFT. The features which SIFT works best are the standalone areas, intersections, and then corners.

As we take the sunflower image, and its 90 degree transform, there are 284 matches. It seems it works worse than my system. However, my system works slower than the OpenCV system.

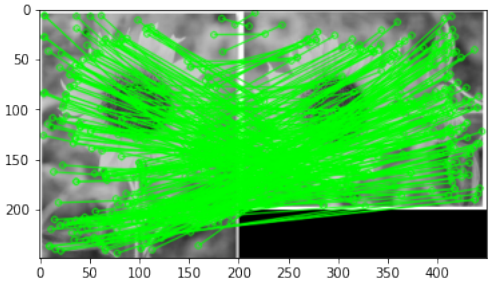


Fig. 5: Sunflower and its transformed one matching

II. Structure from Motion with COLMAP

We have problem in installing COLMAP and processing the set of images. I have problem of installing Ceres solver due to the incompatible with my GPU's version, it requires a new version of cuda. Therefore, I could not run COLMAP with GPU. As a result the program ran very slow. First, we captured a 35 second video of my dining room, and extract more than 100 frames. COLMAP took several hours to process it, and output nothing. Then I captured a short video with 6 seconds, and COLMAP can process in half hours, however, it outputted something not really meaningful. COLMAP could extract the relation between the scene and pose as well as building 3D view (shown in Fig. 6 and 7). However, the result is not as expected. It does not provide an overall scene about environment and the feature inside. It seems we need to provide more data with longer video for COLMAP to build the scene. It took a lot of my time to work with COLMAP, so I let it as uncompleted work.

(a) (b)

Fig. 6

Results/Question:

(a) (b)

Fig. 7

III. PDDL

A. Running Fast Downward

As running the first code's block, we have the result as shown in Fig. 8:

- The length of the completed plans: 34 steps with plan cost of 34.

```
[t=4.43265s, 126952 KB] Plan length: 34 step(s).
[t=4.43265s, 126952 KB] Plan cost: 34
[t=4.43265s, 126952 KB] Expanded 1385559 state(s).
[t=4.43265s, 126952 KB] Reopened 0 state(s).
[t=4.43265s, 126952 KB] Evaluated 2812744 state(s).
[t=4.43265s, 126952 KB] Evaluations: 2812744
[t=4.43265s, 126952 KB] Generated 6245225 state(s).
[t=4.43265s, 126952 KB] Dead ends: 0 state(s).
[t=4.43265s, 126952 KB] Expanded until last jump: 395214 state(s).
[t=4.43265s, 126952 KB] Reopened until last jump: 0 state(s).
[t=4.43265s, 126952 KB] Evaluated until last jump: 910585 state(s).
[t=4.43265s, 126952 KB] Generated until last jump: 1816319 state(s).
[t=4.43265s, 126952 KB] Number of registered states: 2812744
[t=4.43265s, 126952 KB] Int hash set load factor: 2812744/4194304 =
[t=4.43265s, 126952 KB] Int hash set resizes: 22
[t=4.43265s, 126952 KB] Search time: 4.41014s
[t=4.43265s, 126952 KB] Total time: 4.43265s
Solution found.
Peak memory: 126952 KB
Remove intermediate file output.sas
search exit code: 0

INFO     Planner time: 4.55s
```

Fig. 8: Output of block 1

- The number of expanded states: 1385559 states
- The total running time: 4.43 seconds

As running the second code's block, we have the result as shown in Fig. 9:

```
[t=0.00421435s, 10636 KB] Plan length: 48 step(s).
[t=0.00421435s, 10636 KB] Plan cost: 48
[t=0.00421435s, 10636 KB] Expanded 106 state(s).
[t=0.00421435s, 10636 KB] Reopened 0 state(s).
[t=0.00421435s, 10636 KB] Evaluated 107 state(s).
[t=0.00421435s, 10636 KB] Evaluations: 107
[t=0.00421435s, 10636 KB] Generated 519 state(s).
[t=0.00421435s, 10636 KB] Dead ends: 0 state(s).
[t=0.00421435s, 10636 KB] Number of registered states: 107
[t=0.00421435s, 10636 KB] Int hash set load factor: 107/128
[t=0.00421435s, 10636 KB] Int hash set resizes: 7
[t=0.00421435s, 10636 KB] Search time: 0.00157635s
[t=0.00421435s, 10636 KB] Total time: 0.00421435s
Solution found.
Peak memory: 10636 KB
Remove intermediate file output.sas
search exit code: 0

INFO     Planner time: 0.10s
```

Fig. 9: Output of block 2

- The length of the completed plans: 48 steps with plan cost of 48.
- The number of expanded states: 106 states
- The total running time: 0.0042 seconds

As comparing two motion planner, the first one seems providing the optimal solutions. It discovered and evaluated a huge number of states (more than 1 million states). The plan length and plan cost are both small (34), however its running time is more than 4 seconds (very long for a real time application).

For the second motion planner, it does not provide the optimal solution (the planning length is 48) and the number of expanded states are quite small (only 106 states). However, it can find a solution in very short running time (4 milliseconds) - 100 times faster than the first one. I think this motion planner can work well for practical application.

B. An Example PDDL Problem

Results/Question Include in your writeup the actions that minimize the total cost. What is the total cost?

Answer: To reach the goal, we should follow the actions:

1) Travel from Earth to Vulcan - the cost of 10.

2) Get the purple-key - the cost of 1.
3) Travel from Vulcan to Cardassia with the cost of 7
4) Unlock the purple vault - the cost of 11.
5) Travel from Cardassia to Betazed with the cost of 9
6) Get the blue-key with the cost of 1
7) Travel from Betazed to Ferenginar with the cost of 10
8) Get the Red-key with the cost of 1
9) Travel back to Betazed with the cost of 10
10) Unlock the red vault - the cost of 1
11) Travel from Betazed to Qonos with the cost of 10
12) Unlock the blue vault - cost of 1
13) Travel to Levinia with the cost of 500.

So, the total cost is: 10 + 1 + 7 + 11 +9 +1 +10 +1 +10 +1 +10+ 1+ 500 = 562

Question What is the goal specification (in PDDL) of the full problem?

Answer: The goal specification are shown in Fig. 10. All the keys are not on the ship, however, be at the betazed, qonos, cardassia, and the ship should be at levinia.

```
(:goal (and
    (at cerritos levinia)
    (not (on-ship redkey1 cerritos))
    (not (on-ship bluekey1 cerritos))
    (not (on-ship purplekey1 cerritos))
    (at redkey1 betazed)
    (at bluekey1 qonos)
    (at purplekey1 cardassia)
    ))
```

Fig. 10: The goal specification

There are four new actions are created. The first action is *beam-up-key* (Fig. 11), which is at the key's location and get the key on the ship. The key's status is not longer at the location but on the ship. The cost of this action is 1. Those

```
(:action beam-up-key
  :parameters (?s - ship ?l - location ?k - key)
  :precondition (and
    (at ?s ?l)
    (at ?k ?l))
  :effect (and
    (not (at ?k ?l))
    (on-ship ?k ?s)
    (increase (total-cost) 1)
    ))
```

Fig. 11: Action 1

actions *unlock-red-vault*, *unlock-blue-vault*, and *unlock-blue-vault* are similar, and shown in Fig. 12, 13, and 14. They share similar structure: the key need to be on-ship, and the current ship's location is at the corresponding vault. We remove the key and unlock the vault (key and vault are the same location). The key's state changes from on-ship to at the location.

```
(:action unlock-red-vault
  :parameters (?s - ship ?l - location ?rv - vault ?rk - red-key)
  :precondition (and
    (at ?s ?l)
    (at ?rv ?l)
    (on-ship ?rk ?s))
  :effect (and
    (at ?rk ?l)
    (not (on-ship ?rk ?s))
    (increase (total-cost) 1)
    ))
```

Fig. 12: Action 2

The adding initialization of the program is shown in Fig. 15. We need to locate where are the location of the keys and the vaults.

```
(:action unlock-blue-vault
  :parameters (?s - ship ?l - location ?bv - vault ?bk - blue-key)
  :precondition (and
    (at ?s ?l)
    (at ?bv ?l)
    (on-ship ?bk ?s))
  :effect (and
    (at ?bk ?l)
    (not (on-ship ?bk ?s))
    (increase (total-cost) 1)
    ))
```

Fig. 13: Action 3

```
(:action unlock-purple-vault
  :parameters (?s - ship ?l - location ?pv - vault ?pk - purple-key)
  :precondition (and
    (at ?s ?l)
    (at ?pv ?l)
    (on-ship ?pk ?s))
  :effect (and
    (at ?pk ?l)
    (not (on-ship ?pk ?s))
    (increase (total-cost) 1)
    ))
```

Fig. 14: Action 4

The result is shown in Fig. 16. The plan is similar to the optimal path which we discussed earlier. The ship travels to vulcan - get the purple key - travel to cardassia - unlock the purple-vault - travel to betazed - travel to ferenginar - get the red-key - travel back betazed - unlock the red-vault - get the blue-key - travel to qonos - unlock the red-vault - travel to levinia.

- The path length is 13 steps with the cost of 562
- Evaluated 40 states
- The total running time is 0.026 seconds

IV. PDDL WITH NAVIGATION

To have estimated distances among the start, keys and vaults, the package *skimage* was used with the function *skimage.graph.MCP_Geometric*. The distances are rounded as shown in Table I and added into the PDDL problem's initialization.

Between	D	Between	D
Start - red-key	959	blue-key - purple-key	1299
Start - blue-key	738	blue-key - red-vault	1325
Start - purple-key	776	blue-key - blue-vault	643
Start - red-vault	801	blue-key - purple-vault	693
Start - blue-vault	232	purple-key - red-vault	227
Start - purple-vault	616	purple-key - blue-vault	656
red-key - blue-key	1324	purple-key - purple-vault	1187
red-key - purple-key	384	red-vault - blue-vault	682
red-key - red-vault	221	red-vault - purple-vault	1212
red-key - blue-vault	839	purple-vault - blue-vault	530
red-key - purple-vault	1370	-	-

TABLE I: Estimated distance among the locations

Because I cannot find the way to define an 2D position in PDDL, thus, the @D position consists of two locations: *position-x* and *position-y*. Due to that, all the predicates, functions, and action in domain file are modified as following:

The predicate *on-ship* is the the same, but for *at* and *adjacent*, they have more parameters. The new parameters are nothing than the locations in x and y dimensions.

The new travel action is also extended with *from-x, from-y, to-x, to-y* parameters.

The new parameter's set of the *beam-up-key* action is: *?s - ship, ?l-x - location, ?l-y - location, ?k - key*.

The new parameter's set of the *unlock-red-vault* action is: *?s - ship, ?l-x - location, ?l-y - location, ?rv - vault ?k - key*.

```
(at cerritos earth)
(at purplekey1 vulcan)
(at bluekey1 betazed)
(at redkey1 ferenginar)
(at bluevault1 qonos)
(at redvault1 betazed)
(at purplevault1 cardassia)
```

Fig. 15: The initial conditions

```
travel cerritos earth vulcan (10)
beam-up-key cerritos vulcan purplekey1 (1)
travel cerritos vulcan cardassia (7)
unlock-purple-vault cerritos cardassia purplevault1 purplekey1 (1)
travel cerritos cardassia betazed (9)
travel cerritos betazed ferenginar (10)
beam-up-key cerritos ferenginar redkey1 (1)
travel cerritos ferenginar betazed (10)
unlock-red-vault cerritos betazed redvault1 redkey1 (1)
beam-up-key cerritos betazed bluekey1 (1)
travel cerritos betazed qonos (10)
unlock-blue-vault cerritos qonos bluevault1 bluekey1 (1)
travel cerritos qonos levinia (500)
```

Fig. 16: The found path

```
(:predicates
  (at ?l - locatable ?px - location ?py - location)
  (on-ship ?k - key ?s - ship)
  (adjacent ?ax - location ?ay - location ?bx - location ?by - location)
  )
```

Fig. 17: The new predicates

```
(:functions
  (distance ?ax - location ?ay - location ?bx - location ?by - location)
```

Fig. 18: The new functions

```
(:action travel
 :parameters (?s - ship ?from-x - location ?from-y - location ?to-x - location ?to-y - location)
 :precondition (and
   (at ?s ?from-x ?from-y)
   (adjacent ?from-x ?from-y ?to-x ?to-y))
 :effect (and
   (not (at ?s ?from-x ?from-y))
   (at ?s ?to-x ?to-y)
   (increase (total-cost) (distance ?from-x ?from-y ?to-x ?to-y))
   ))
```

Fig. 19: New travel action

```
(:action beam-up-key
 :parameters (?s - ship ?l-x - location ?l-y - location ?k - key)
 :precondition (and
   (at ?s ?l-x ?l-y)
   (at ?k ?l-x ?l-y))
 :effect (and
   (not (at ?k ?l-x ?l-y))
   (on-ship ?k ?s)
   (increase (total-cost) 1)
   ))
```

Fig. 20: New beam-up-key action

```
(:action unlock-red-vault
 :parameters (?s - ship ?l-x - location ?l-y - location ?rv - vault ?rk - red-key)
 :precondition (and
   (at ?s ?l-x ?l-y)
   (at ?rv ?l-x ?l-y)
   (on-ship ?rk ?s))
 :effect (and
   (at ?rk ?l-x ?l-y)
   (not (on-ship ?rk ?s))
   (increase (total-cost) 1)
   ))
```

Fig. 21: New unlock action

After changing the parameters of the PDDL, we have a plan of the robot traveling as following:

```
travel cerritos start-x start-y rkey-x rkey-y (959)
beam-up-key cerritos rkey-x rkey-y redkey1 (1)
travel cerritos rkey-x rkey-y rvault-x rvault-y (221)
unlock-red-vault cerritos rvault-x rvault-y redvault1 redkey1 (1)
travel cerritos rvault-x rvault-y pkey-x pkey-y (227)
beam-up-key cerritos pkey-x pkey-y purplekey1 (1)
travel cerritos pkey-x pkey-y bvault-x bvault-y (656)
travel cerritos bvault-x bvault-y pvault-x pvault-y (530)
unlock-purple-vault cerritos pvault-x pvault-y purplevault1 purplekey1 (1)
travel cerritos pvault-x pvault-y bkey-x bkey-y (693)
beam-up-key cerritos bkey-x bkey-y bluekey1 (1)
travel cerritos bkey-x bkey-y bvault-x bvault-y (642)
unlock-blue-vault cerritos bvault-x bvault-y bluevault1 bluekey1 (1)
travel cerritos bvault-x bvault-y start-x start-y (231)
```

Fig. 22: The plan with the occupancy map

The features of this motion planning:

- The planning length is 14 steps with the plan cost: 4165
- The expanded states: 42 and evaluate 91 states
- The running time: 1.38 seconds.

After having the plan, especially the travel path, we used the *astar-search* function (used in Project 2) to find the path. The path is a set of waypoints and we used an PD controller to move a differential drive robot follow the path. The results are shown as following:

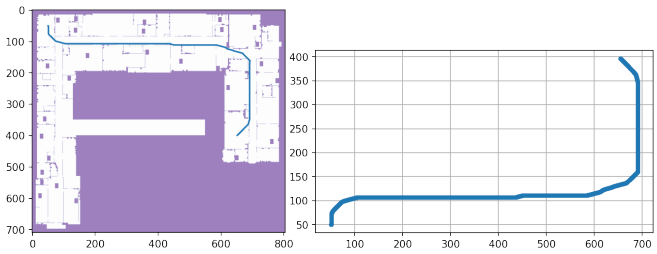


Fig. 23: Path Segment 1: From *start* to *red-keys*

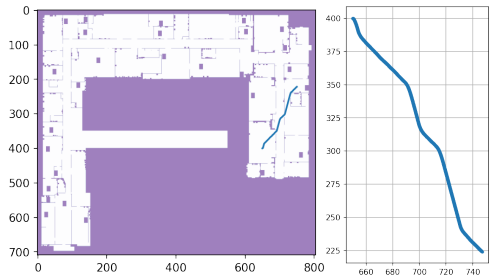


Fig. 24: Path Segment 2: From *red-key* to *red-vault*

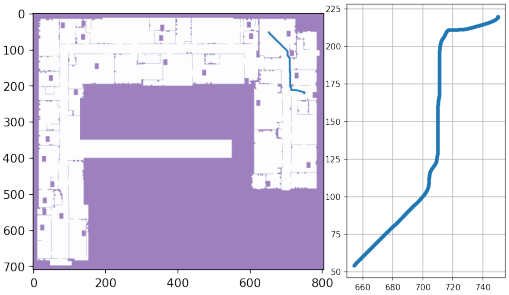


Fig. 25: Path Segment 3: From *red-vault* to *purple-key*

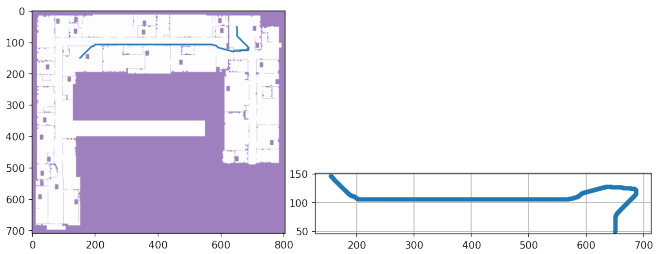


Fig. 26: Path Segment 4: From *purple-key* to *blue-vault*

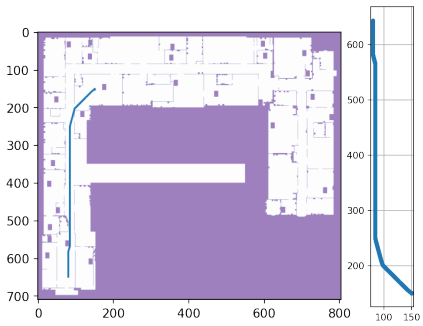


Fig. 27: Path Segment 5: From *blue-vault* to *purple-vault*

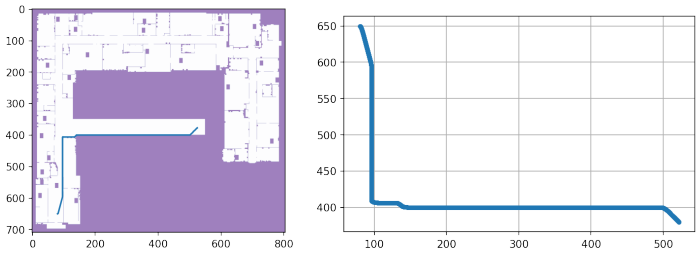


Fig. 28: Path Segment 6: From *purple-vault* to *blue-key*

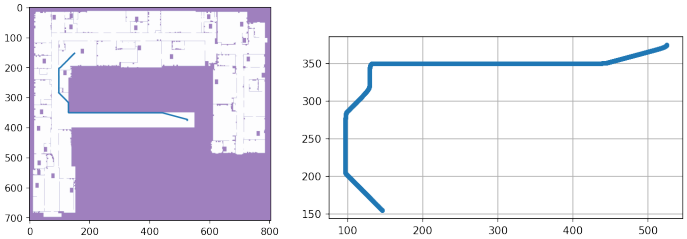


Fig. 29: Path Segment 7: From *blue-key* to *blue-vault*

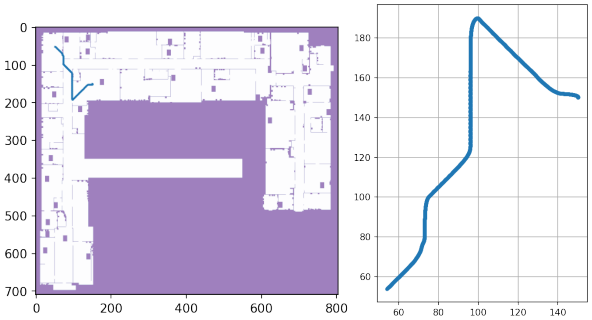


Fig. 30: Path Segment 8: From *blue-vault* back to *start*