Room Segmentation

in

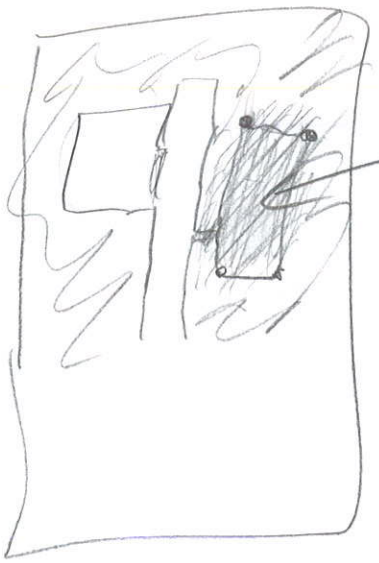
I. • for (i=0; i < (1,5/map_resolution); i++)

1. erode image further by 1
2. find contours
3. for each contour
   i) check if contour fulfils criteria of a room *
   → yes: • save contour somewhere
   • set this contour to black in image (since it has already been stored)
4. exit loop if there are no more contours (= black image)

• store the remaining contours (left over rooms)

* criteria for checking that a contour is a room:
- area (in m^2) is bigger than a small number and smaller than ~50 m^2 or so → this will not hold for big rooms, so this must be a rather soft criterium
(- area has to be compact, i.e. there should not be too many black areas within the room)

II. copy original map (white = accessible, black = walls and unknown areas)
copy the stored contours into the map and fill each with a unique id numbe
repeat until convergence (i.e. there are no more white pixels)
- a white pixel with a labeled neighbor pixel receives the label of the neighbor



fill contour cv::drawContours

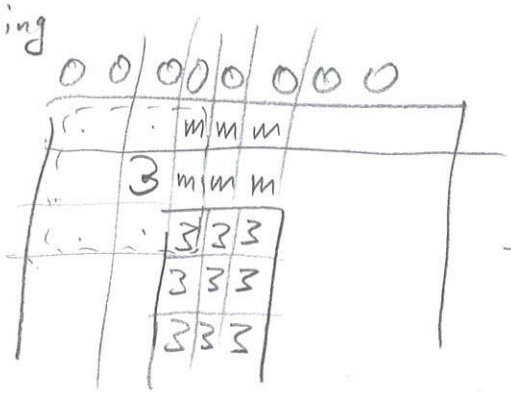
original + labeled areas

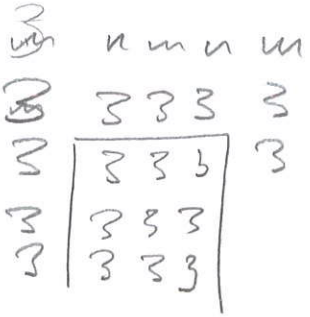
drawContours
- index
- color = index + 1

← cv::Mat with type CV_32SC1
int one channel

white color = maximum integers

growing





~~Edge~~ while (have pixels in white)

for (v=0; v < image.rows; v++)

for (u=0; u < image.cols; u++)

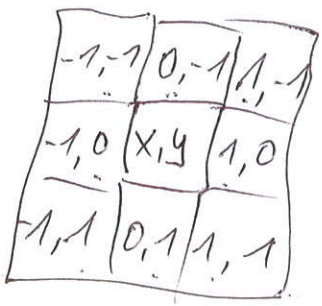
vector neighborsX = {-1, 0, 1, -1, 1, -1, 0, 1}
neighborsY = {-1, -1, -1, 0, 0, 1, 1, 1}

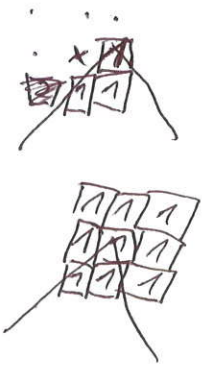
```
for (y...)
    for (x...)
        for (i=0, i<8;i++)
            newy = y + neighborsY[i]
            new_x = x + neighborsX[i]
```

read original
→ if white & label as neighbor
write label copy

original = copy

```
image_original
image_copy = image_original.clone();
```





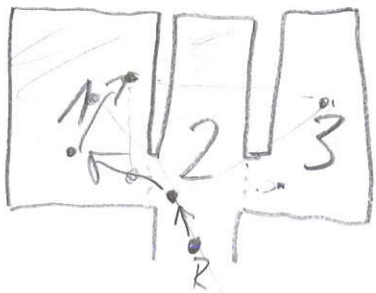
1. autonomous visiting of all detected rooms (e.g. travelling salesman problem)
original
2nd task, page: 0

2. inspection of each room

3. learn about state machines with SMACH

4. learn about action servers

1. go to next unprocessed room



a) check if room was already processed
↳ not → inspect room (2.)
↳ yes → b)

b) check which is the closest unprocessed room

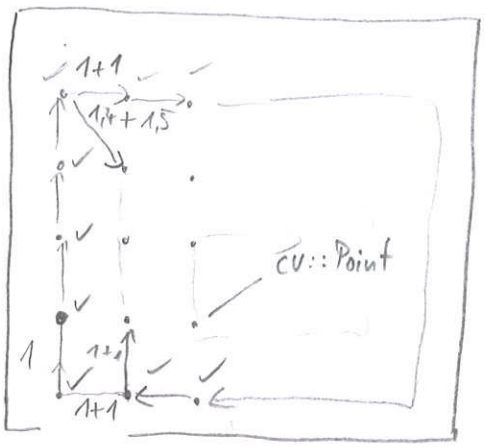
c) → return the next room

state

pose

move_base

2. inspect room

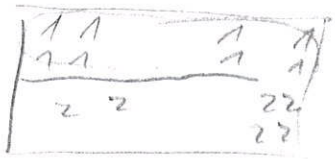


std::map < cv::Point, bool > pointsToVisit

= point 1, false
point 2, false
⋮
point n, false

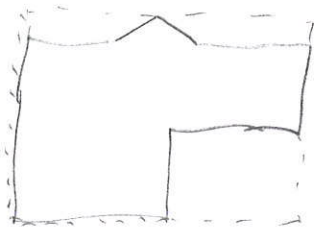
↳ once you arrived or could not arrive at point x
set map: point x, true

2. a) you know the shape* of room because you can read all the labels from room map

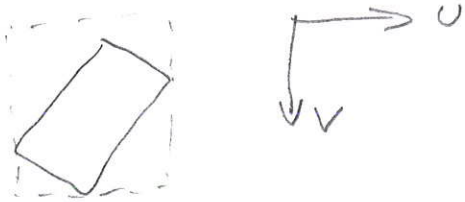


* which pixels are part of the room

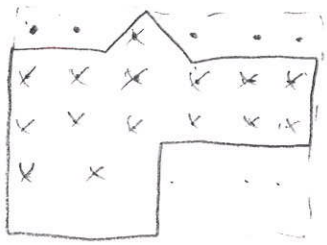
b)



find bounding box of room



c) sample a grid of points to visit from the bounding box



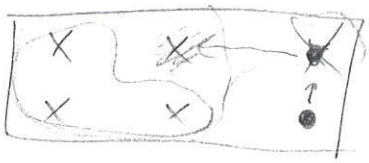
i) for each sampled point
check that this point is inside the room (= point label = room label)
↳ yes, inside the room:
put point into the pointsToVisit vector <cv::Point>

d) (as long as points left in pointsToVisit) (find next best point)
for each point in pointsToVisit
compute costs to go to that point
remember goal point with lowest costs → nextPoint

e) move robot (or try to move it) to the nextPoint

f) once you arrive there or if navigating there fails
→ remove nextPoint from pointsToVisit

g) go to d) until pointsToVisit is empty



h) you finished the room

Room inspection depends on Clearence_factor and Step_size!

Loop through all pixels of image

std::vector<int> minU (numberLabels, 1000000)
maxU (— " —, 0)
minV
maxV

↳ each vector has one element per label

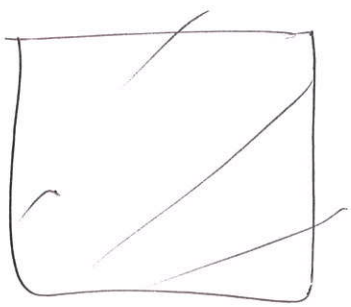
for each pixel
  check label
    check if u and v coordinates of
    this pixel are the smallest or biggest
    for this label

Example: Pixel u=110, v=180, label=3

minU[3] = min(u, minU[label])
↑ label

minV[3] = std::min(v, minV[3])
↑ 180 ↑ ~~0~~ 1000000
↖ 180

Boundingbox for area with label
is
minU[label]
maxU[label]
minV["]
maxV["]



end task, page:-04

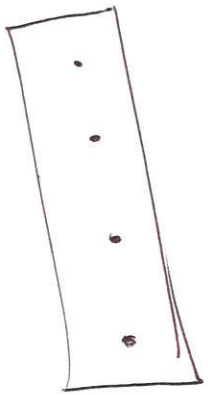
contours

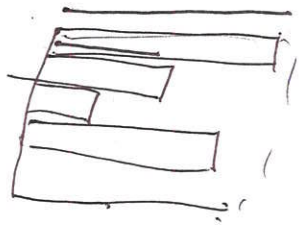
vector < vector < cv::Points > >

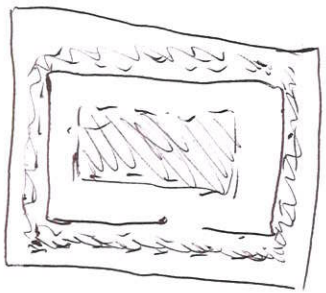
0 [8 Points]
1 [4 Points]
2 [16 Points]

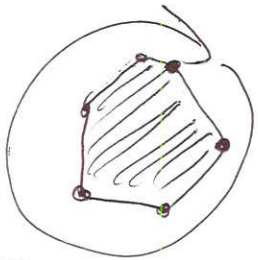
[0]
[1]
[2]

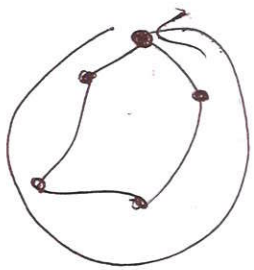
0 1 2

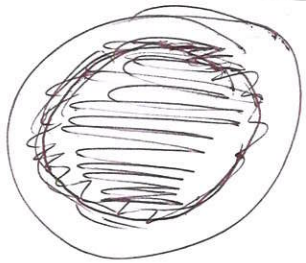




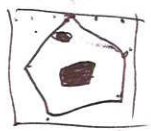








1. pointPolygonTest

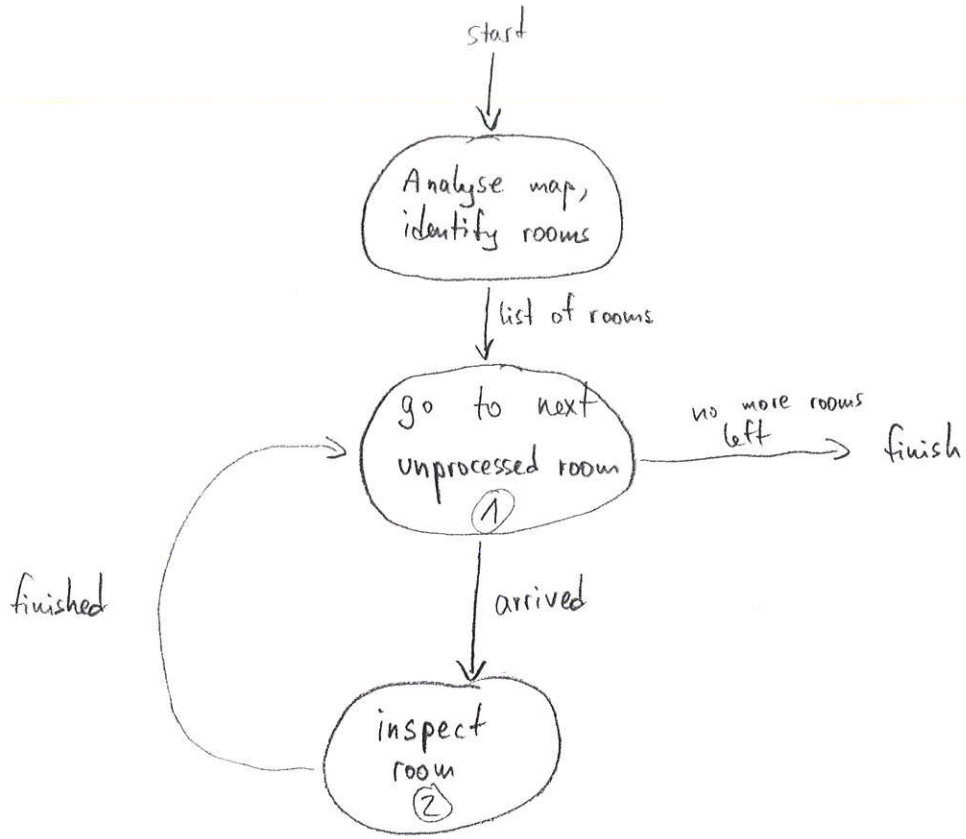


→ count Pixels inside
black Pixels inside

2. clockwise / counter clockwise test

3. hierarchy

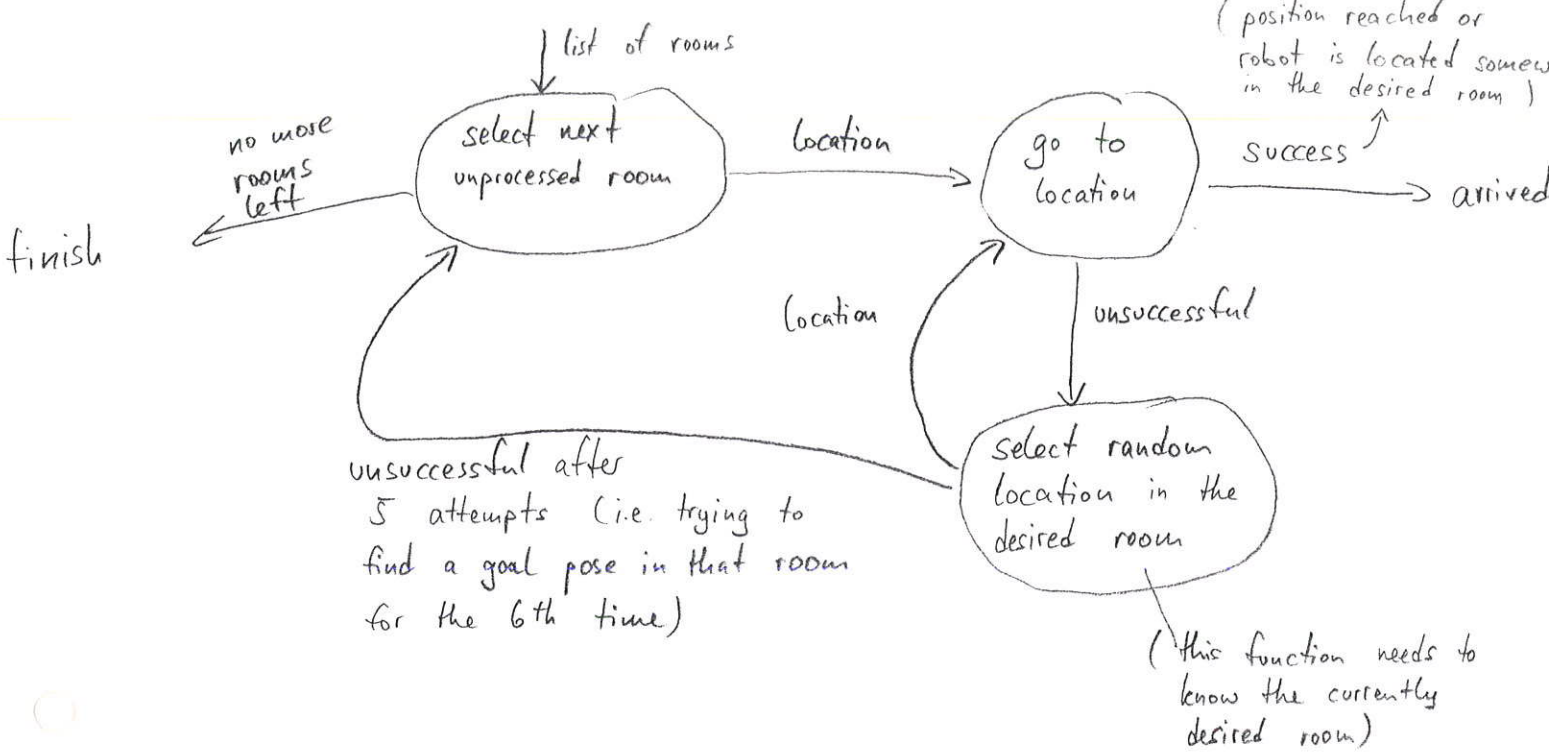
Task: Cleaning



(1) try centre point first, if not successful → already in desired room → fine, proceed
↘ not in desired room → sample random location in desired room and try again
↳ after 5 attempts without success, mark room as unreachable and go to next room

(2) drive to all computed poses in the room, skip those which you cannot reach

(1) Task: go to next unprocessed room



a) init

/srv receive static map

orig. inflated map from static map

b) callback for obst. + infl. obst.

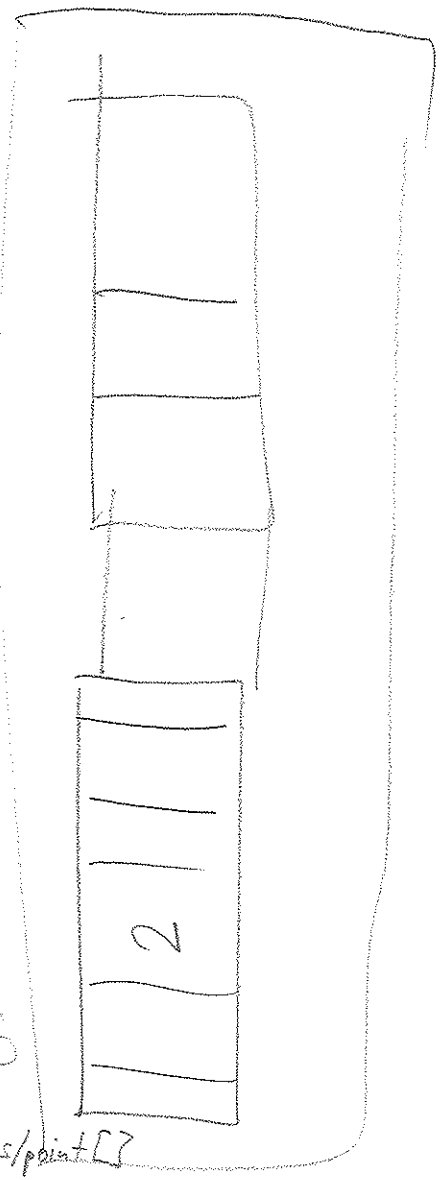
inflated map ← original infl. map + copy + obst. + infl obst.

c) execute

read from inflated map

→ check whether a query point is 0 in inflated map

action



service: 1. deliver the whole inflated_map

2. check points

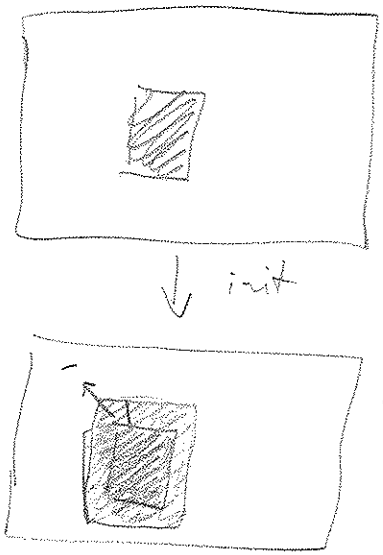
→ array of points geometry_msgs/point[]

→ check their states

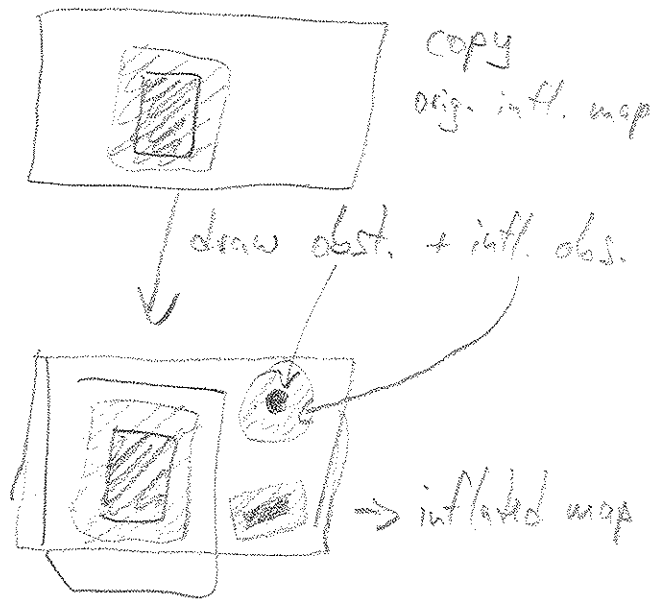
→ return array of free/occupied states

bool[]

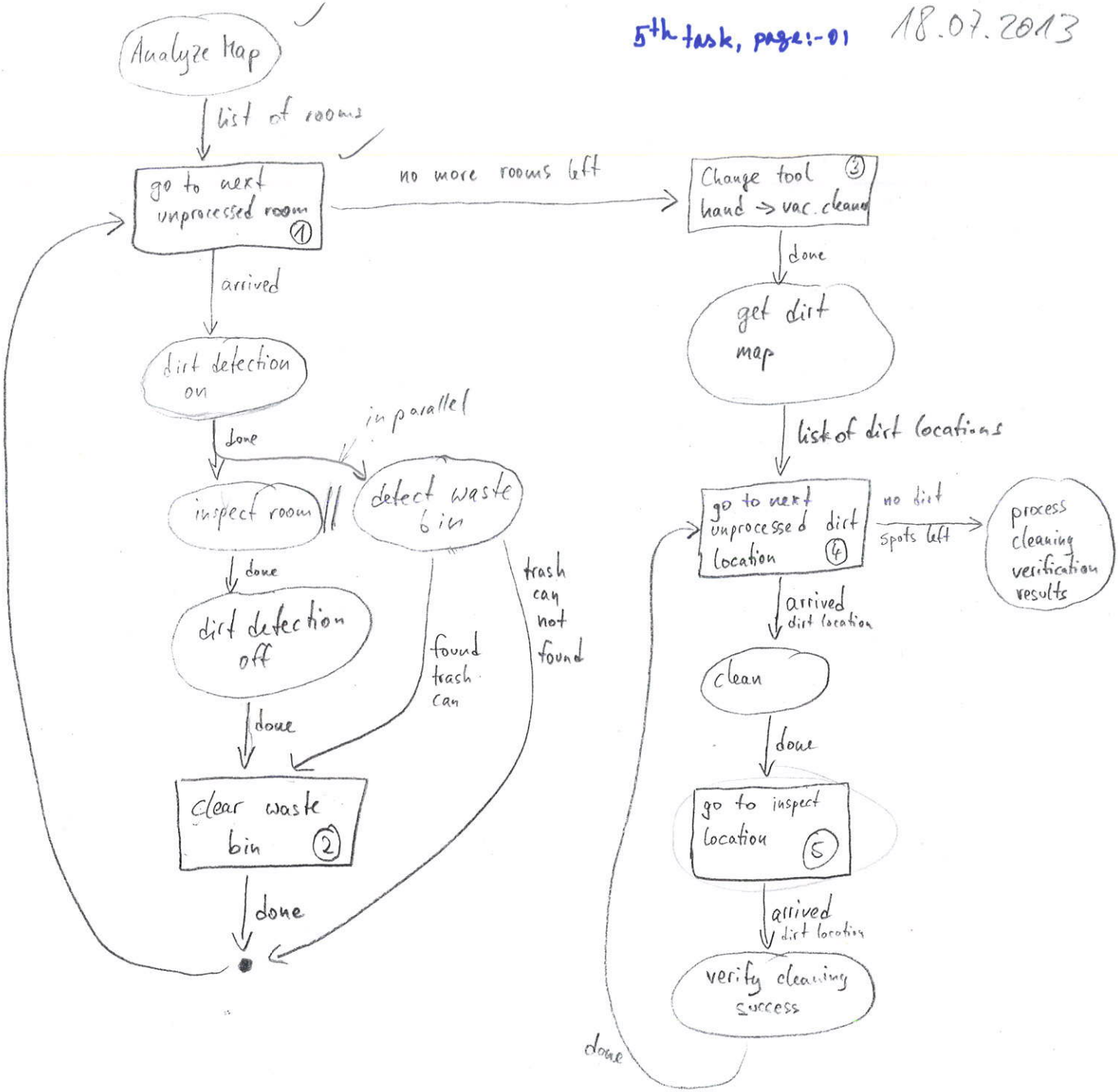
a)

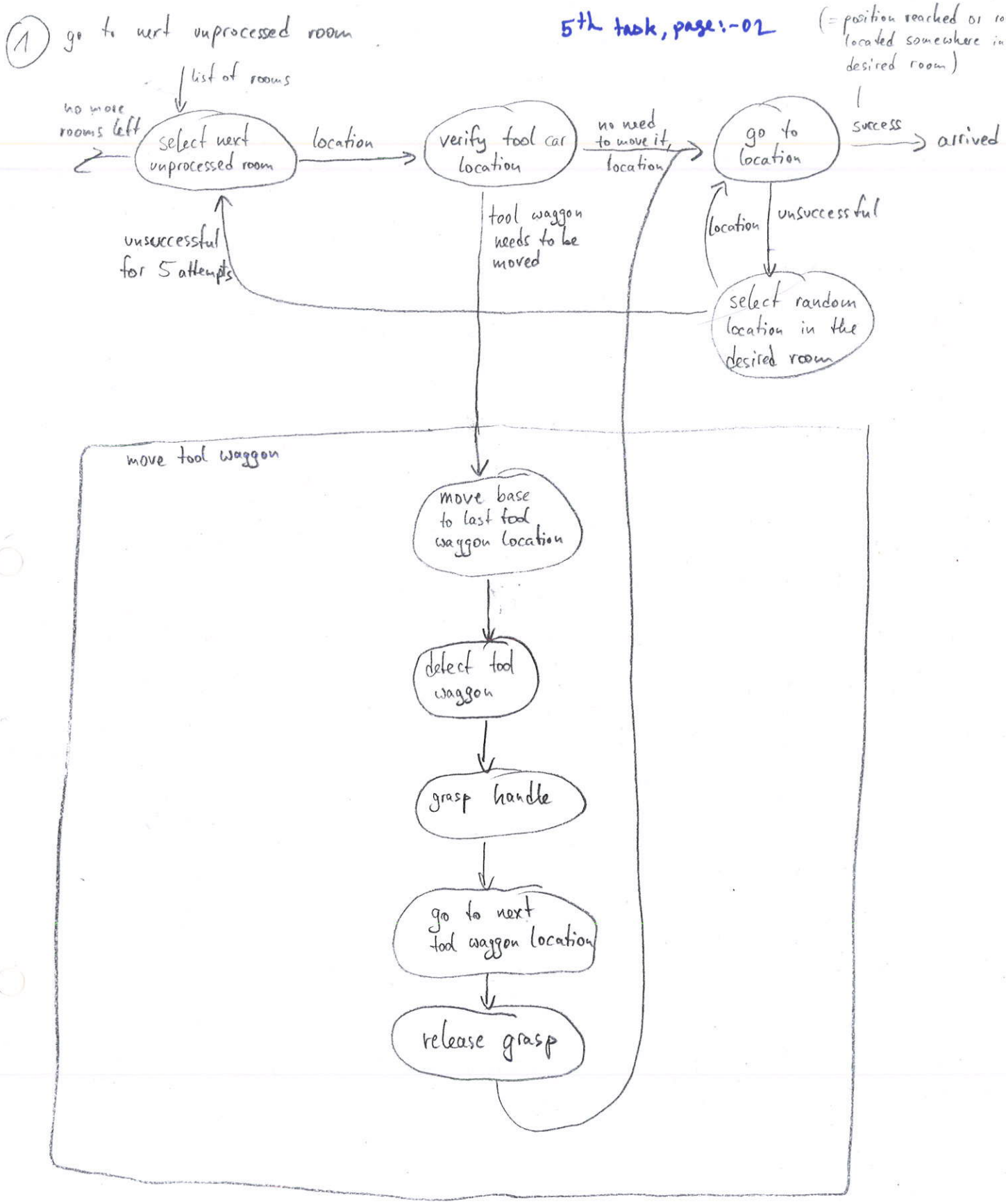


b) callback with obs. + infl. obs.

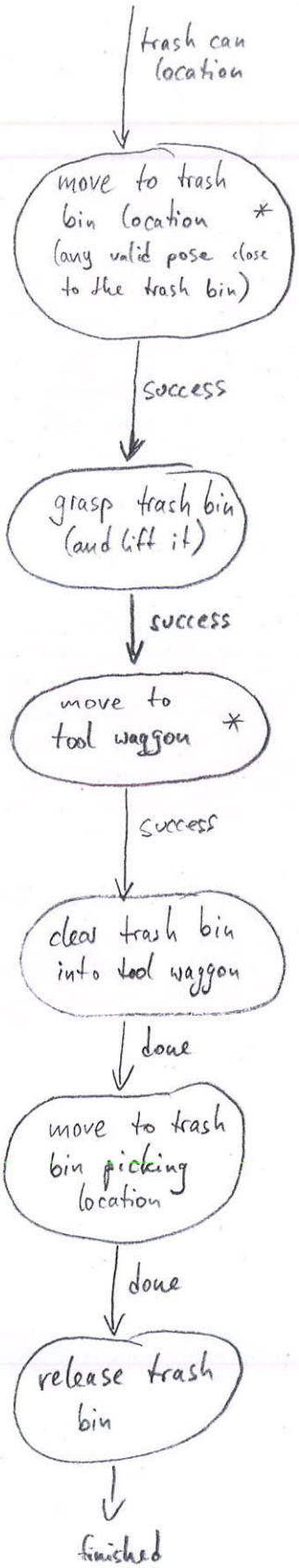


5th task, page:-01 18.07.2013



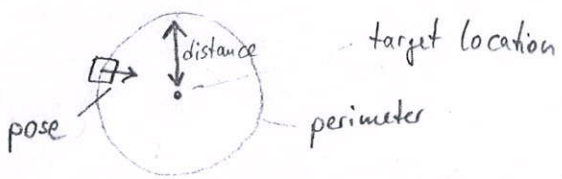

1
go to next unprocessed room
list of rooms
no more rooms left
select next unprocessed room
location
verify tool car location
no need to move it, location
go to location
success
arrived
(= position reached or robot located somewhere in the desired room)
unsuccessful for 5 attempts
tool waggon needs to be moved
location
unsuccessful
select random location in the desired room
move tool waggon
move base to last tool waggon location
detect tool waggon
grasp handle
go to next tool waggon location
release grasp

(2) clear waste bin

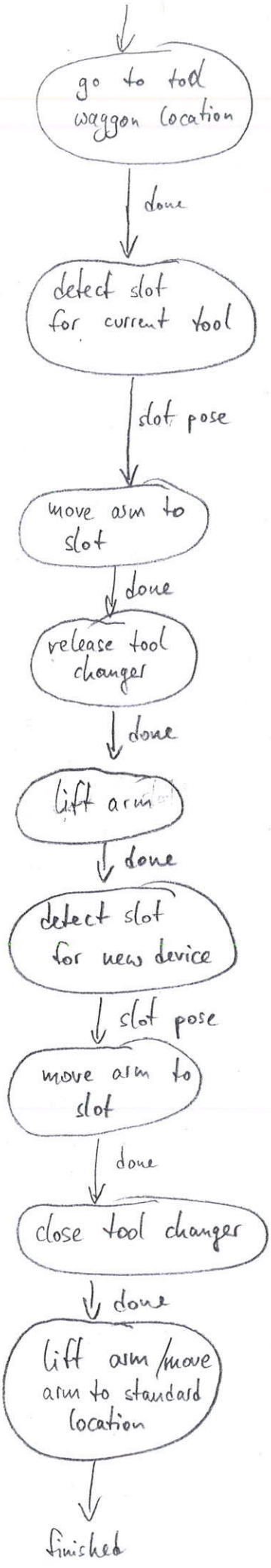


* use a generic move function, e.g. move_base_perimeter, which allows to specify: - distance
- pose relative to target location

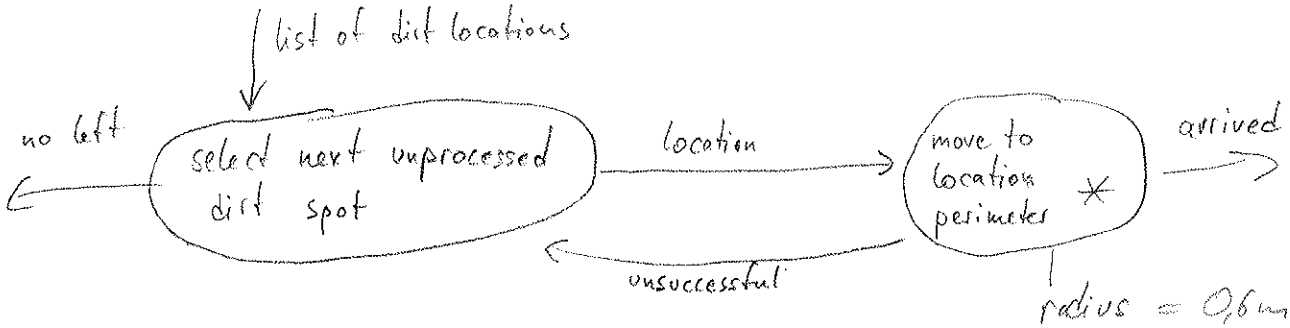
e.g.



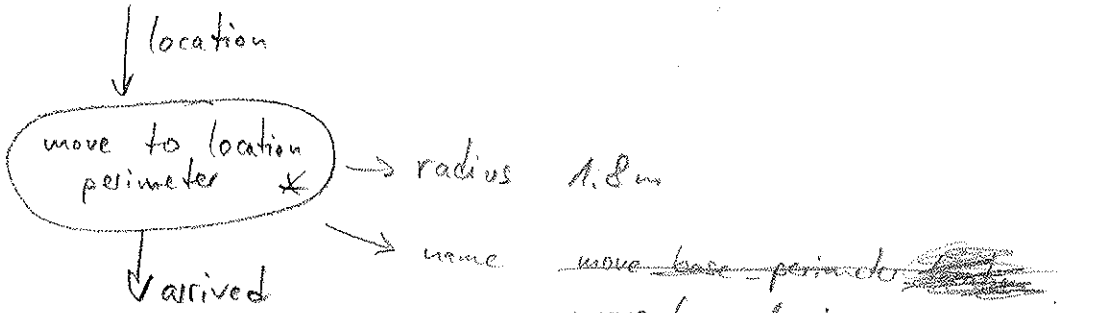
(3) change tool



(4) go to next unprocessed dirt location



(5) go to inspect location

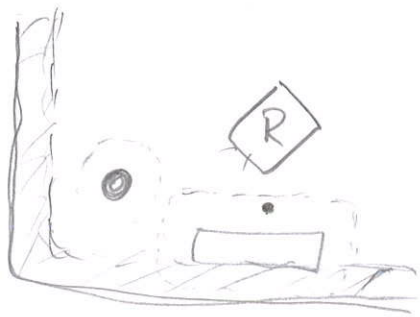


* as at (2) this function computes a perimeter around the goal location and tries to move somewhere on this perimeter in a defined pose to the goal location

/map → walls, enviroment

/local_cost_map (or similar)

/move_base/local_costmap/inflated_obstacles



in inspect room

1. check cost map
2. verify that your next goal is not inside an inflated obstacle
3. skip goals that are not reachable

/rmb-ou/autopup/autopup_tod_clou

cob_perception_common

cob_object_perception

(cob_fiducials)

/fiducials/detected_fiducials

on/off

on/off

trash bin locations

(PoseStamped)

service → internal variable on/off

trash_bin_detection

Subscriber * → Pose Pose ...

if turned on

*. read the detections — From whose ?

. if detection.label.compare("2") == 0

→ store location internally in a vector ↑

↓

transform location (Pose) to /map coordinate system

Pose type PoseStamped

tf::transformPose

or tf::lookupTransform from detection.header.frame_id to "/map"

↓

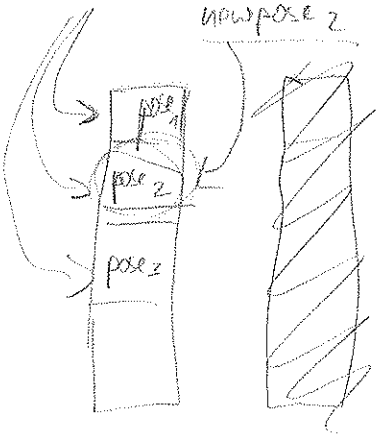
this T with Pose

↑ check if there is a detection in the vector with similar pose

→ yes: average both and store at vector

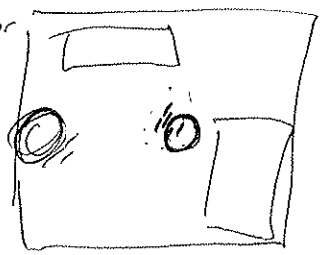
→ no: just put it in vector

new_pose



~~** for turning off~~

- ~~take the vector of poses~~
- ~~average all poses that are close to each other~~



average Pose:

$$\frac{\begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} + \begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix}}{2} \quad \text{position}$$

$$\frac{\begin{pmatrix} x_1 \\ y_1 \\ z_1 \\ w_1 \end{pmatrix} + \begin{pmatrix} x_2 \\ y_2 \\ z_2 \\ w_2 \end{pmatrix}}{2} \quad \text{orientation}$$