REPORT
SEMESTRAL PROJECT

Oleh Rybkin and Hanna Chaika

Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Cybernetics
B3M33ARO

May, 2017

1 Introduction

The aim of the semestral project was to implement a program for the robot, so that it autonomously goes through the race course. The course is given by checkpoints that are denoted by QR code marker on a vertical stand and a circle in given distance in front of the marked stand. The robot needs to pass through the circle of each checkpoint, avoiding possible obstacles along the way. The project was implemented the Python programming language.

2 Algorithm

2.1 Generating environment map

We store a simple 2D map of the environment consisting of obstacles and checkpoints. The map is created from 3D points gathered by the depth sensor, and projected to 2D "bird view" (see Fig. 1).

Markers are projected to 3D world space using camera matrices [5] obtained from the sensor automatically. For speed, we, however, assume the same camera center and the same camera coordinate frames. We found that this approach works perfectly unless the marker is very (5+ meters) far or its image very close (30 pixels) to the camera image edge. 3D points of the marker are fitted to a plane using singular value decomposition (SVD), and a checkpoint is computed from that plane and the position of the robot.

We found that this simple checkpoint estimation works with the precision of a few centimeters, thus we don't update the checkpoint positions.

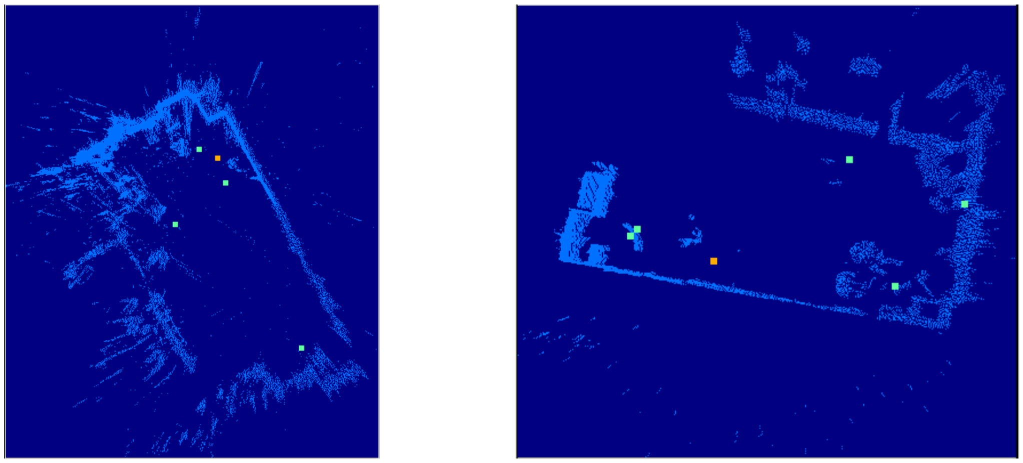

Figure 1: Examples of environment map that we store. Obstacles are colored in light blue. Orange square is the position of the robot, cyan squares are the checkpoints. This map was generated with a simple movement and 360° rotation. Checkpoints are 40 cm away from actual markers.

Robot position is estimated with odometry information. We considered using iterative point cloud (ICP) algorithm on 2D environment maps, but it turns out that odometry works very well, at least for short paths under 100 meters.

”Bird views” captured in consecutive frames are integrated into a single map using odometry. Processing point clouds turns out to be a computationally expensive procedure, so we capture the ”bird views” only each second. We further reduce the computational burden by storing only one 2D point per 2 cm × 2 cm square in the environment map.

2.2 Planning

Path planning is implemented via rapidly exploring random trees (RRT) [3] in 2D space of the stored environment map (see Fig. 2). We found that if we leave only one 2D point per 2 cm × 2 cm square, the number of points required to store the map can be compressed from hundreds of thousands down to few thousands. More importantly, the number of stored points does not grow up too much during exploration.

We firstly considered using graph search (Dijsktra or A*) on a grid-like discretized map, but RRT offers natural thinking in a continuous space, which allows e.g. to turn on an arbitrary angle. While the planned path is almost always suboptimal, this could be alleviated with implementing RRT* [2], for which we unfortunately didn't have time.

The planning routine is reexecuted every few seconds so that new information about obstacles can be integrated.

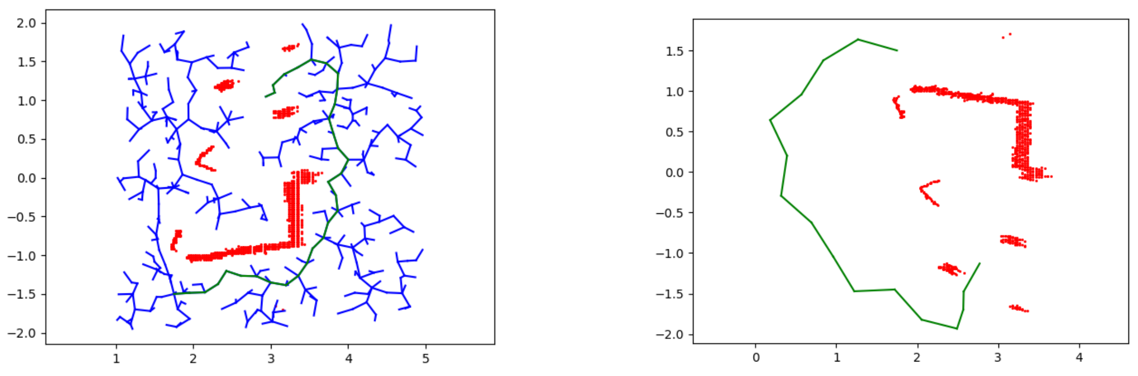

Figure 2: Examples of trajectories planned with RRT (left - all nodes, right - only planned path). Planned path shown in green, obstacles in red. It can be seen that the method tends to go off the scene, which means that scene dimensions need to be carefully determined.

2.3 Following the path

The planned path is represented by a sequence of goal points. For the current goal point, we compute the angle between it and our direction α_{goal}, and the distance to the points d_{goal}. Then a simple rotational motional followed by a translational motion is executed. The robot rotates until the desired angle α_{goal} is reached within precision of 2 degrees, and then moves forward until it covers the desired distance d_{goal}.

We found out that these simple heuristics work, but sometimes give noisy movement. The biggest problem was that while moving (i. e. executing a purely translational motion), robot tended to turn to one side a little. This problem has been fixed by using a PID controller (see e.g. [1]):

$$u(t_k) = K_p e(t_k) + K_i \sum_{i=1}^{k} e(t_i)\Delta t + K_d \frac{e(t_k) - e(t_{k-1})}{\Delta t} ,$$

where $\Delta t = t_{current} - t_{previous}$ is the sample time, deviation $e(t_k) = \alpha_{current} - \alpha_{start}$ is the difference between the angle at which the robot have started the moving and the actual angle taken from the odometry sensor.

We found that using just the proportional term is already enough to correct the deviation. We set K_p to 0.5. Adjusting terms K_i and K_d could further improve the behaviour of the controller.

2.4 High-level control

We tried to keep the high level reasoning of the robot simple. The control architecture is behaviour-based. The order of actions is as follows:

1. Rotate left by 45° degrees and then right by 90° to create a map of the scene.
2. Estimate the dimensions of the scene from the map. This is needed for RRT.
3. Until all the checkpoints are visited:
   (a) Find a path to each unvisited checkpoint. The path is computed 10 times using RRT and the shortest one is chosen.
   (a) Execute movement to the checkpoint to which the shortest path corresponds. During the movement, environment map and the path itself are updated each few seconds.
   (a) Play the audio and mark the checkpoint as visited.

3 Testing and Conclusions

The algorithm was tested in a simulation of the environment described in the assignment [4]. We used three markers and two obstacles, placed in the front part of

E-209 room. The robot was able to navigate the scene correctly in almost all cases. The time needed for the task was few minutes at worst. We hope that the current algorithm is capable of solving the task under a minute when moving at maximal velocity.

References

[1] PID controller. `https://en.wikipedia.org/wiki/PID_controller`.

[2] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *Int. J. Rob. Res.*, 30(7):846–894, June 2011.

[3] Steven M. Lavalle. Rapidly-exploring random trees: A new tool for path planning. Technical report, 1998.

[4] V. Kubelka, M. Pecka, K. Zimmermann, L. Wagner. Repository with material for the final project. `https://gitlab.fel.cvut.cz/kubelvla/b3m33aro_semestral`, 2017.

[5] K. Zimmerman,. Camera model and calibration. `http://cw.fel.cvut.cz/wiki/_media/courses/b3m33aro/pinhole.pdf`.