Robotics HW1 Report

B03902062 資工三 董文捷

1. **Part C**
   I have read some examples using `addKeyHandler` in documents. In `gyroExample.cpp`, it chooses `ArFunctor1C` as second parameter. By this way, one can easily access simple function like `setVel`, but can not customize for complex requirements. I therefore followed `actsColorFollowingExample.cpp` to create particular callback function and pass needed parameters by `ArGlobalFunctor1` and `ArGlobalFunctor2`. In the callback function, it will print out the key pressed, also adjust `Vel` and `RotVel` according to its original value. At first, I tried to make acceleration and deceleration by `setTransAccel` and `setRotAccel`, but it does not work as expected. I then decided to use `getVel` and `setVel`, `getRotVel` and `setRotVel`. Every time when a key is pressed, get current value of corresponding velocity first, if it does not achieve defined maximum velocity, acceleration or deceleration is made by increasing or decreasing velocity and pass it as parameter to `setVel` and `setRotVel`. Otherwise, it can only maintain current velocity, and a message of achieve maximum velocity will be printed.

2. **Part D**
   To avoid collision, I added `currentReadingPolar` in callback function of UP and DOWN. Whenever UP or DOWN is pressed, the sonar will check whether there is an obstacle on the moving direction. If returned distance is less than defined safety distance, the callback function will make acceleration or deceleration, trying to stop the robot. Furthermore, in some circumstance, the user does not press any key to change the velocity of robot, but the robot is still moving, so the same collision avoidance part should also be added in the while loop in main.

3. **Part E**

   - Because it is hard to control moving time and distance of robot through `setVel`, I used `move` instead. I also noticed the document mentions that it is best to restrict the distance to the range (5000mm, 5000mm] if possible. A while loop is used to reach the goal, everytime a movement is done, a caculation of distance between current position and goal is made. If the distance is within the tolerance, the robot will stop; otherwise, it will rotate to new direction according to arctan value of delta y to delta x, and a move command is

sent. However, something confused me is that although the reach criterion is met, the final output position seems to exceed the goal for a long distance that can not be neglected. At first, I thought it was because the robot does not stop immediately, so I use `stop` whenever `isMoveDone` returns true. However, I found the situation not improved. I then realized that even in the simulator, there exist some kinodynamic constraints, that is to say, due to the constraint on deceleration, the robot can not stop right after the move is done, it needs some time to slow down. Some solutions is tried, for example, multiply the expected moving distance with a coefficient < 1 as parameter to `move`, taking the extra distance for stopping into consideration, but the extra distance is not linear and dependent of moving distance. Decrease the moving distance slice a time can make maximum velocity lower, but will not meet the time requirement. `setAbsoluteMaxTransVel` did not work, directly adjust velocity by `setVel` when it is too high will influence `move` because it is implemented by VEL command. The best solution I found is to use a while loop waiting until `getVel` equals to zero, although there is still extra movement made when decelerating, at least we can get precise position and the next command will not be affected. If the error is too large, the stop criterion will not pass and next movement is planned until the robot is near enough to goal.

- Odometric pose starts at (0, 0, 0) when the simulation begins, and changes as the robot moves forward, backward, or rotates, based on that motion plus some simulated error. On the other hand, true pose means the absolute position of the robot within the the simulator. The true pose of start position may not be (0, 0, 0). Approximately, we can say that *true pose = true pose of start position + odometric pose*. In simulation with pioneer and Mobilesim, it is not exactly equal, possibly due to sensor error for odometric pose.
- Performance (use `time` in ubuntu with stdin and stdout redirection)
  - Test case $(5, 7, 1.57)$
    $23.100s, 23.166s, 23.066s, 22.758s, 23.275s$ Average : $23.073s$
  - Test case $(-3, 4, -4)$
    $33.092s, 33.250s, 34.191s, 33.647s, 32.990s$ Average : $33.434s$

4. **Bonus** : To find a suitable algorithm for bonus task, I have read papers about path planning with unknown obstacle. At first, I read some part of D* Lite, but I found it complicated to implement. Then I read some simple papers about Bug algorithms, including IMPLEMENTATION OF TAN-

GENT BUG ALGORITHM WITH A NON-HOLONOMIC WHEELED MOBILE ROBOT PIONEER 3-DX and PERFORMANCE COMPARISON OF BUG ALGORITHMS FOR MOBILE ROBOTS. Both of them simulated their results with pioneer and Mobilesim, and Bug algorithm seems to be an appropriate solution to this task, so I started to focus on Bug2 algorithm. The algorithm Bug2 is a greedy algorithm that the mobile robot follows a constant slope computed initially between the positions of start position and goal. The mobile robot maintains its motion to goal unless the path on the slope is interrupted by an obstacle. If the robot faces it, the robot follows edges of the obstacle by using its sonar sensors in the clockwise sense until it finds its initial slope again. However, both of papers do not mention details on how to implement obstacle following. I then read the third part, *From Theory to Implementation* in Performance Comparison of Bug Navigation Algorithms, which mentions PD controller for wall following. Unfortunately, I do not have enough time to get deeper into it. I tried to implement Bug2 according to my own understanding, but the method I used for obstacle following does not work when the path need to have a turn due to the corner of obstacles. For now, my program can not finish the bonus task. I will try to improve my method or find some other references for obstacle following to complete Bug2 algorithm when available.