Assignment 1: Bug Algorithms

Trevor Stephens

10/25/2016

Overview

This report briefly outlines the implementation of two different bug algorithms (Bug 1 and Bug 2) implemented in ROS. Each bug algorithm will be described with special attention to the necessary decisions in implementing the algorithms. Additionally, results will be included to depict the efficacy of each algorithm.

Bug 1 Algorithm

The basic workflow for the Bug 1 algorithm is the following:

- Head towards goal
- If an obstacle is encountered, circumnavigate the entire obstacle and remember the point along the obstacle closest to the goal
- Return to the point along the obstacle closest to the goal and repeat until target is reached

For each of these items in the workflow certain decisions needed to be made to accomplish a successful algorithm.

Head Towards Goal

The task of heading towards the goal was accomplished in a two-step approach: orient the robot's bearing towards the goal and proceed forward along this bearing line (called s-line). Orienting the robot towards the goal was accomplished via a proportional controller controlling about θ_{error} to

set angular velocity commands; linear velocity commands were set to zero. This is depicted in equation 1.

$$\dot{\theta}_{z,command} = k_{p,\theta}(\theta_{robot} - \theta_{goal}) \tag{1}$$

Once the bearing was found, heading towards the goal was accomplished via constant linear velocity commands while the angular velocity was set to zero.

Follow a Wall

A necessary component of this algorithm is the ability of the robot to follow a wall. Once the laser scanner returned a range less than a pre-determined safe wall distance (0.5 meters) then the robot entered the wall-following state. This threshold was selected based on the robot's footprint combined with worst-case scenario turning and an added safety factor built in. This limits the robot to only being able to traverse narrow alleys which are greater than approximately 1 meter wide, but it was a design choice implemented to guarantee no wall collisions. Once in the wall-following state, a proportional controller convexly blended both $d_{wall,error}$ ($d_{w,e}$) and $\theta_{wall,error}$ ($\theta_{w,e}$) as depicted in equations 2 and 3.

$$\dot{\theta}_{z,command} = \begin{cases} \alpha k_{p,\theta}\theta_{w,e} + (1-\alpha)k_{p,d}d_{w,e} & \text{if } \theta_{w,e} \text{ and/or } d_{w,e} \text{ are large} \\ 0 & \text{if } \theta_{w,e} \text{ and } d_{w,e} \text{ are small} \end{cases} \tag{2}$$

$$v_{x,command} = \begin{cases} v_{x,min} & \text{if } \theta_{w,e} \text{ and/or } d_{w,e} \text{ are large} \\ v_{x,max} & \text{if } \theta_{w,e} \text{ and } d_{w,e} \text{ are small} \end{cases} \tag{3}$$

In the actual implementation α was set to $\frac{1}{7}$ as to emphasize moving away from walls more heavily. The values for $v_{x,min}$ and $v_{x,max}$ were set to $0.25\frac{m}{s}$ and $1.5\frac{m}{s}$, respectively. The thresholds for "large" angular error measurements was set to 0.1 radians, and the threshold for "large" distance error measurements was set to 1 millimeter. This error threshold for distance works in simulation, but in practice it may be too small for a noisy depth sensor such as a Kinect® or even a laser scan.

Circumnavigate an Obstacle

The circumnavigation of an obstacle utilized the previously explained wall following control algorithm. As the robot circumnavigated the obstacle, the

distance to the target was calculated at each time step, with the location of the shortest distance to the target stored. After circumnavigation of the entire obstacle the robot then proceeded to circumnavigate the object (in the same direction as the first time) until it reached the location which was shortest to the target. To determine if the robot had circumnavigated as well as reached the stored location, small squares were placed around each point of interest. These small square were set at 0.6 meters per side and 0.2 meters per side for circumnavigating and finding the stored location, respectively. Once the robot entered these squares the objective was considered complete. The slightly larger choice of 0.6 meters per side for circumnavigation was selected because it was paramount not to miss a circumnavigation, especially when the robot was following a curved wall where repeating an exact trajectory the second time around is not guaranteed.

Reaching Target

Once the robot has circumnavigated the obstacle and returned to the location shortest to the target, the algorithm starts over. These steps are repeated until the target is reached. Determining whether or not the target has been reached is done in a similar manner to determining circumnavigation as previously described. A small threshold square is placed around the target with side length 0.65 meters; if the robot enters the square the target is assumed to be reached. This threshold was set to make sure the robot did not collide with the target irrespective of the angle of approach to the target.

Bug 2 Algorithm

The second bug algorithm utilizes many of the same functionality programmed in the first bug algorithm. The overall workflow is as follows:

- Head towards goal
- If an obstacle is encountered, begin circumnavigating the obstacle and continue until the the s-line has been reached
- Leave the obstacle and head towards goal on the s-line

Head Towards Goal

The method for orienting towards the goal and heading towards the goal is identical to the method described previously for the Bug 1 algorithm.

Follow a Wall

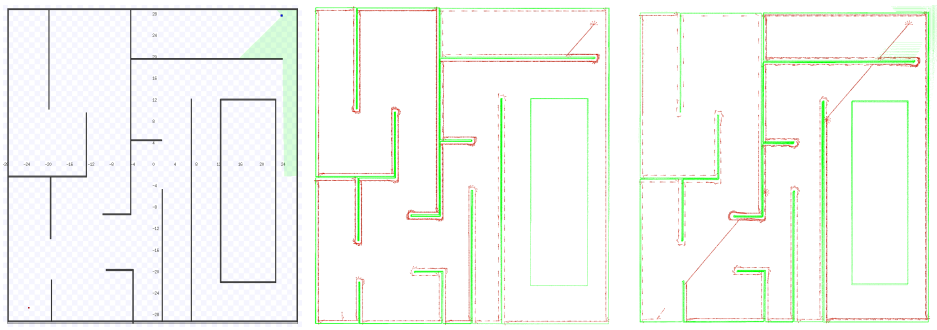
The method wall following is identical to the method described previously for the Bug 1 algorithm.

Reaching the S-Line

A major difference between Bug 1 algorithm and Bug 2 algorithm is that the robot may leave the obstacle prior to full circumnavigation if it finds a spot that is closer to the target and on the s-line. This requires an algorithm to determine if the robot is back on the s-line. This was accomplished by continually monitoring the sign of the difference between the bearing to the target at the beginning of the algorithm and the current bearing to the target. When this sign changes it means that the s-line has been crossed, which is the immediate point that the robot has gotten back to the s-line. However, this is not the only trigger in the algorithm. At any crossing of an s-line, the distance to the target is checked. If the distance to the target is shorter than any other previous point along the s-line then robot follows the s-line. However, if the distance to the target is longer than a previously encountered point along the s-line then that s-line crossing is ignored and the algorithm continues in wall following mode.

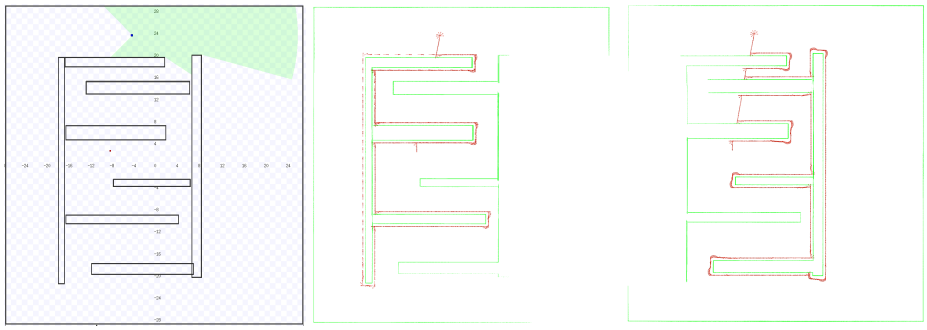
Results

The bug algorithms were tested across 9 different maps across a variety of start and goal locations. The map and results for each bug algorithm are included in figures 1-2 for two representative courses.



(a) Map (b) Bug 1 algorithm (c) Bug 2 algorithm

Figure 1: Successful goal finding for each bug algorithm in a maze



(a) Map (b) Bug 1 algorithm (c) Bug 2 algorithm

Figure 2: Successful goal finding for each bug algorithm in an "E"-shaped course

For all courses there was 100% accuracy in finding the goal for each of the bug algorithms in the finite number of starting and goal locations selected. During testing, there were no collisions with any walls or other obstacles. The algorithm does emphasize avoiding collisions by maintaining a safe wall distance of 0.5 meters; the trade-off of this design choice is that the robot has limited access to small alleys measuring approximately 1 meter or less in width. The control algorithm is also designed for smooth operation as a jerky ride is not optimal when transitioning to sensors in an non-simulated environment.