ECE 232 Lab 5/6

Matthew Dombroski

April 27, 2016

I affirm that I have not given or received any unauthorized help on this assignment, and that all work is my own.

Abstract

The purpose of this lab was to implement an autonomous navigation system for navigating the robot through an obstacle course. The core concepts of the class such as localization, optimal path planning, and path following were implemented in one integrated system that successfully navigated the robot through the course in the simulator.

1 Code Implementation

There were several overarching tasks needed in order to implement this lab. First the localizer needed to be updated so that it could handle multiple features within the map. Second a path planning algorithm needed to be implemented such that a path correctly navigating the course would be computed for each waypoint. An executive, or a program to monitor the overall state of the robot and control the high level objectives for the robot and direct it to each successive waypoint implemented. Finally a path following algorithm needed to be implemented to command the robot to follow the path created by the planner within acceptable margins

Localization

The final implementation for the localization algorithm is extremely simple and basic, but shown in the simulator to be reliable.

A matrix of z_{est} stores all of the predicted cone locations in relation to the robot. These values are updated for every cycle of localization whether or not a measurement for that cone was received. For every laser scan read all the returns are filtered and the shortest overall return is selected. This return is then compared with all of the expected cone positions in the matrix z_{est} and the cone with the highest correlation to the beam return is then selected as the beam feature correspondence. The beam return is then extended by the cone radius and the bearing, location, and correspondence are stored in the measurement vector z. The EKF algorithm will then update the robot pos and the expected cone positions based on

this measurement.
Some additional details for the localization are that the shortest beam return is correlated based upon the lest difference in distance between the beam return and cones - whichever cone is closest to the beam return is tentatively set as the correspondence for that beam. If the closest beam return is further than 2.5m away the beam return is rejected. If this test passes the bearing of the beam is then checked against that expected cone bearing and if it is within a certain tolerance based on the variance for that cone bearing, the correspondence is firmly set for that beam measurement. If at any time any of these tests fails, the measurements are ignored and the EKF updates without a measurement.

In addition it was found that updating the EKF algorithm faster caused it to be more accurate. With more updates per time interval more data is processed but the algorithm and more potential beam returns are integrated into the pos estimation. The overall result was that even given certain drawbacks in the implementation of the EKF the results of the localization were still reliable and accurate.

1.1 Path Planner

For planning the path of the robot A* with an 8 neighborhood connected grid was implemented. The main A* graph search was left unaltered and works as shown in class. The grid A* searches is created as the robot goes along using knowledge of the map and the current goal to determine the cost of motion from one node to the next. The discretization of the grid was chosen to be 0.25m as that was found to be a good balance between enough resolution to create a path to go around the cones without going unduly out of the way and course enough for A* to run quickly. In addition during testing it was found that with a discretization of 0.1m A* would never converge on the goal and would run in an infinite loop. It was never sufficiently determined why this was the case, as it would be expected that 0.1m would not greatly increase runtime overall.

Th base cost of motion to a node is dependent on the motion to get to that node. In an 8 connected grid some motions travel along a diagonal while others travel along an axis. Given that a diagonal motion travels a slightly longer distance it is weighted more than a motion along an axis. In the case of this implementation a movement along an axis is weighted at 1 and a motion along a diagonal is weighted at $\sqrt{2}$. This prevents zig-zagging paths that are in reality sub-optimal for reaching the goal. All other costs for traveling to a node are based on this initial cost.

Computing the cost between nodes was computed based on the particular motion, the distance of the destination node from a cone, and the orientation constraint for each waypoint. Given the grid is 8 connected the robot could move in any of 8 directions for each node being expanded. Reverse motion is suppressed implicitly by the fact that there will never be an optimal path to any waypoint that incorporates a reverse motion. If the robot became extremely unlocalized and somehow wound up ahead of the waypoint there could be a potential for a reverse path planned. However the localizer in this case would almost certainly never be able to re-localize in the event of such extreme uncertainty and so the course

traversal would result in failure anyway. Paths coming close to the cones are suppressed by taking the euclidean distance between the destination node and every cone in the map. If the distance falls bellow a certain threshold the cost to move to that node is increased by 50. The threshold was determined by adding the robots radius with the cone radius, plus a certain amount of expansion to give the robot some play to make up for any inaccuracies in the localization. The resulting paths generally go travel directly between the cones in a straight line due to the inflated dimensions of the cones. Paths that would result in an incorrect orientation at the waypoint are weighted out by testing for what waypoint is being traveled to. For example, when traveling to waypoint 1 the robot needs to pass through the cones facing the negative y direction. Any motion resulting in the robot having a pos with a negative y component would thus be sub-optimal, as it would have to backtrack a long way to pass through the cones in the correct direction. Thus for waypoint 1 any node with a negative y component that is negative has 50 added to its cost.

1.2 Path Follower

The path following algorithm implemented was pure pursuit. The overall path following controller receives the path from the planner and commands the robot to follow that path as closely as possible. To compute the lookahead distance first the projection point of the robot onto the path is computed and used as the origin point to start searching for the lookahead. This projection is useful because even if the robot is farther off the path than the lookahead distance this projection point can be used as the lookahead point to guide the robot back to the path. Otherwise the lookahead would come off the path and create nondeterminate behavior in the velocity commands.

After finding the closest closest point the path from that point on is discretized very finely. Each point along this discretized path is examined and checked if its euclidean distance is almost equal to the lookahead distance, within some threshold. Once a point is found to be within acceptable margins it is returned as the lookahead point and used to compute the curvature and angular velocity given the deviation of the robots y coordinate from the lookahead. In the case of this project a linear velocity of 0.1m/s was chosen to minimize noise and uncertainty as much as possible. Given the paths produced by the planner and the velocity chosen a lookahead distance of 0.15m was chosen. Such a short lookahead was used as it was found that any greater of a distance caused the corners to be rounded too greatly, and the robot could potentially run into a cone.

The PFC also tracks each waypoint it has passed and can detect when it has arrived at the goal point. Around each waypoint is a radial threshold which if the robot passes it is considered to have passed that waypoint. The specific implementation I have made uses waypoint data to facilitate the computation of the closest path point to the robot. If the PFC detects that the next waypoint is the last waypoint of the path and the threshold has been passed it commands the robot to stop until the path is updated.

Finally the angular velocity is bounded to be at max π radians per second, which is the max rotational velocity of the Turtlebot.

1.3 Executive

The executive for this project was integrated into the same file as the planner. This simplified the code a fair amount as there were many problems with getting ROS to communicate data in the correct sequence. The executive is preloaded with a list of goal points, the same points listed in the lab guide. The executive listens to the
pos estimates from the localizer and uses the predicted robot position to decide if a goal point is almost reached. If the robot is closer to the current goal than a certain threshold the executive pops off the next goal point and commands the planner to compute a path from the current goal to the next goal. This threshold for the executive is larger than the waypoint threshold in the PFC so that by time the robot reaches that waypoint the path has already been updated and the robot does not need to stop. As a consequence this goal threshold is determined by the expected runtime for A*. If the A* were to take a longer time to compute a path this threshold would need to be expanded out to give the algorithm time to get the path ready by time the robot almost exhausted the old path. In the case of this lab A* runs almost immediately so the threshold did not need to be expanded much.

2 Data and Analysis

3 Laboratory Questions