Translating Robotics Course Materials from Elite Research I Universities to Historically Black Colleges and Universities

Xuejun Liang
Department of Computer Science, Jackson State University
1400 J. R. Lynch Street, Jackson, MS 39217
xuejun.liang@jsums.edu

Abstract

Teaching an upper-level undergraduate robotics course at Historically Black Colleges and Universities (HBCUs) is challenging. The lack of suitable teaching materials is one of the biggest challenges, although there are many great masterpieces in developing robotics course materials, which are, however, generally developed for teaching students at elite Research I universities. This paper presents ideas and details in adopting and revising these course materials and creating upper-level undergraduate robotics course materials that are suitable for underrepresented students.

Introduction

Teaching an upper-level undergraduate robotics course at Historically Black Colleges and Universities (HBCUs) is challenging. The lack of suitable teaching materials is one of the biggest challenges, although there are many great masterpieces of robotics courses. For example, there is an online list of excellent robotics course materials (Dollar et al., 2010). But, these materials are prepared for teaching students at elite Research I institutions. Students are expected to learn and figure out many details on their own and often required to complete their robot programming projects from scratch. These expectations and requirements are too high for HBCU students. Therefore, translating these materials into HBCU courses and making them suitable for HBCU students learning is necessary.

The overall approach of the translation is breaking down big tasks into pieces and providing more detailed in-class teaching materials, including programming fundamentals, formulas and algorithms, skeleton codes of robot control programs, and step-by-step guidance for doing projects. Due to the space limit, three major robot programming projects are selected to report here. Our whole robotics course materials are available online and can be found at http://www.jsums.edu/robotics/CSC499-539-10F/.

Robot Programming Projects

This section will present some details of the selected three robot programming projects. They are waypoint following, target searching, and path planning. All three projects are adopted and revised from Prof. Parker's robotics course entitled Software for Intelligent Robots (Dollar et al., 2010). Note that the Player/Stage robot programming framework is used in these projects.

P1. Waypoints Following

The task of this project is to read a sequence of waypoints from a data file and then drive the robot to each waypoint one after another. The project is required to use the robot's odometry data and the servo-loop control approach. The following steps will guide students walking through the project.

1. Give students a skeleton world file and let them add details in the given file according to the requirements such as world size, simulation window size, etc.
2. Give students a skeleton client code which provides the program structure. The client program gets a sequence of waypoints from a file by calling the getWaypoints function and then enters an outer loop to iterate through each waypoint. The inner loop is a control loop to drive the robot moving to a waypoint. Students only need to complete three C++ functions:
   - getWaypoints. It reads a sequence of waypoints from a data file and stores them into a queue.
   - gotoWaypoint. It computes the distance and angle from the robot's current pose to the waypoint. If the distance is small enough, then return true. This will indicate that the robot has reached the waypoint. Otherwise, return false.

- translate. It applies the servo-loop control rules to compute the speed and the turn rate based on the distance and the direction of motion.

Please note that in order to help our students to complete these three functions, several points are worth to mention. First, students need to have C++ programming skills on File I/O and using queue data structure, and understand the call-by-value and call-by-reference. Second, students need to know how to obtain the robot's current pose from the robot's odometry (encoder) data. Third, students need to know how to compute the distance between two points, the slope of a straight line, and the angle between two lines, and understand the difference between degree and radian. Fourth, students need to understand the rule-based servo-loop control approach in order to determine an updated speed and turn rate to drive the robot based on the distance and angle between robot and waypoint.

P2. Targets Searching

This project will generate robot control by sequencing and combining three behaviors: wander, avoid obstacles, and go to beacons. The final robot behavior should cause the robot to wander around avoiding obstacles until it detects a previously unvisited beacon. Then, it moves to the location of the beacon. Once the beacon is reached, the robot should go back into a wander mode to search for another beacon. This will repeat infinitely. The following steps will guide students walking through the project.

1. Give students a skeleton world file and let them add details for defining and creating beacons and robot.
2. Give students a skeleton client code which provides the program structure. In the control loop, after updating the proxies, three functions: wander, avoidObstacles, and gotoBeacon are called to active the three behaviors. Then, if no beacon is found, combine (weighted vector sum) the outputs from wander and avoidObstacles to get a single output vector, otherwise, combine outputs from gotoBeacon and avoidObstacles. Finally, the translate function is called to get a speed and a turn rate to drive the robot. Students only need to complete three C++ functions:
   - wander. It generates a new random vector of distance and direction of motion every 3 seconds. (Note that the control loop runs at about 10Hz, so 30 loop iterations is about 3 second.)
   - avoidObstacles. It generates a vector of distance and direction of motion to avoid the obstacle, if there is an obstacle in front detected by using laser scanner. Otherwise, it generates a zero vector. Note that once starting avoiding, it is needed to continue avoiding for 2 seconds.
   - gotoBeacon. It returns true whenever an unvisited beacon is detected by using the Fiducial detector. Otherwise, return false. When an unvisited beacon is detected, it computes a vector of distance and angle from the robot's current pose to the beacon.

Note that the translate function has been implemented in P1. The function to combine (weighted sum) two vectors is given. In order to help our students to complete these three C++ functions, several points are worth to mention. First, students need to know how to produce a random number in a given range and how to do the coordinate transformation (rotation and shift) between the world coordinate and the robot's coordinate. Second, students need to know the implementation skills for the wander function to generate a new random vector when it is called every 30 times and for the avoidObstacle function to continue generating the same vector for 20 times once starting avoiding. Third, students need to know how to use the list data structure in C++.

P3. Path Planning

The task of this project is to implement the wavefront path planer. The path planner accepts as input a user's goal point and generates the waypoints of a path from a given starting point to the goal point. Then, the avoid obstacles behavior and the go to waypoint behavior from the previous projects are used to drive a robot to follow the path. The following steps will guide students walking through the project.

1. Give students a skeleton world file and let them fill in information for simulation window and map.
2. Give students a skeleton code of the wavefront path planner, which provides all detailed implementation of the planner except the following three functions, which are left for the student implementation.
   - grow. It grows the obstacles in the grid map for a single step, i.e. one grid cell farther. It scans the grid cells. If a cell is occupied, then mark the unoccupied neighbors of the cell as occupied.
   - propagate. It propagates the wavefront one grid cell farther. It starts from the grid cells with value i and the propagated cells get the value i+1.
   - nextWaypoint. It computes the next waypoint. The next waypoint is a neighboring cell of the current waypoint and its value is 1 less than the value of the current waypoint cell.

Note that the implemented portion of the planner for students includes the conversion between the pixel map representation and the grid map representation of an image, the conversion between the world coordinate and the image coordinate, and the waypoints relaxing. In order to help our students to complete the three functions, two points are still needed to make. First, students need to know the grid cell representation of an image, the process of scanning the grid cells, and how to process the edge cells and corner cells easily. Second, students really need to pay attention to not accessing an array element outside the array boundaries.

References

Dollar, A., Rus, D., and Fiorini, P. (2010). *Robotics Courseware*, Available at http://roboticscourseware.org