# Developing Robot Programming Lab Projects

**Xuejun Liang**
Department of Computer Science, Jackson State University, Jackson, MS, USA

**Abstract** - *Robotics has played an important role in educations at different levels. But, robotics education at the college level is still ad-hoc. Many researchers have developed many great robotics course materials including lab projects. However, those materials are for teaching students at elite Research I universities rather than underrepresented students at Historically Black College and Universities (HBCUs). This paper presents ideas and details in adopting, revising, and developing robot programming lab projects that are suitable for underrepresented students at HBCUs.*

**Keywords:** Robotics Education, Robot Programming, Player/Stage, Tekkotsu

## 1 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 have been many great efforts in developing such robotics courses. For example, a Cognitive Robotics course has been developed at the Carnegie Mellon University [8, 11]. There is also an excellent list of robotics course materials [3]. But, those materials are prepared for teaching students at elite Research I institutions with a far more quick pace than in HBCU courses and were taught more on independent learning than in-class learning preferred by most HBCU students. Therefore, translating those materials into HBCU courses and making them suitable for HBCU students learning is necessary.

In addition, robot programming tasks in robotics competitions held along with the Annual ARTSI Student Research Conferences are valuable materials [1, 2]. But, it is difficult for our students to complete the whole tasks from scratch and without further detailed guidance, and therefore, there is a need to revise them in order to make them become useful and suitable robot programming lab projects.

In the rest of this paper, a brief overview of our robotics course will be given in Section 2. Robot programming with using the Player/Stage and with using the Tekkotsu will be introduced in Section 3. Three robot programming projects with using the Player/stage and two with using the Tekkotsu will be discussed in Section 4 and Section 5, respectively. The discussions for each project will focus on two aspects: 1. task and detailed steps to guide students to accomplish it, and 2. necessary knowledge for completing the projects, including programming skills, mathematical formulas, algorithms, and issues regarding to failures and uncertainty. Finally, a short discussion and conclusion is presented.

## 2 Overview of the Robotics Course

The robotics course is designed as an elective course for both senior undergraduate students and graduate students of Computer Science. It covers major topics on intelligent mobile robotics, including robot control architectures, sensing, localization, navigation, planning, and uncertainty. The course also reviews programming fundamentals in C++ language and introduces two robot programming software packages: the Player/Stage and the Tekkotsu. Students are evaluated on their homework assignments on major robotics topics, robot programming projects, midterm examination, and final examination. The course has been offered three times in the fall semesters over the last three years. The robot programming projects in each semester are updated with adding and/or removing some projects. The three course websites, one for each semester, are available to the public [5].

The robot platform used in our robotics course is the iRobot Create robot with the ASUS Eee PC on top of it [10]. As mentioned above, two robot programming software packages used in our robotics course are the Player/Stage [4, 9] and the Tekkotsu [8, 11]. In this paper, five robot programming projects will be discussed. Three of them are adopted and revised from Prof. Parker's robotics course entitled Software for Intelligent Robots [7]. They are waypoint following with using odometry data, targets searching and approaching by using behavior coordinating, and metric path planning by using wavefront algorithm. These three projects use the Player/Stage. The other two projects are developed based on the tasks in robotics competitions held along with the Annual ARTSI Student Conferences [1, 2]. In these two projects, a robot needs to navigate and localize itself within a maze, and to announce detected objects and their locations in the maze. But, the objects inside the maze and the navigation markers on the walls of the maze are different in these two projects.

## 3 Robot Programming Platforms

The iRobot Create is a popular mobile robot in robotics education. It uses differential drive and equips buttons for power, play, advance, and wheel drops (front, left, and right), bumps (left and right), IR sensor, wall sensor, cliffs sensors (left, front left, right, front right), encoders (distance, angle),

and Leds. An ASUS notebook computer sitting on top of iRobot Create is functioned as the brain of the robot. The Player/Stage and the Tekkotsu are installed on the ASUS computer.

## 3.1 Robot Programming with the Player/Stage

As shown in Figure 1, Player server provides a network interface to a variety of real or simulated robots and sensor hardware. It commands robots and gets sensor data through device-specific connections. User robot control programs (clients) communicate with Player server through the TCP connection and hence can run on any computer with a network connection to the robot (Player server).

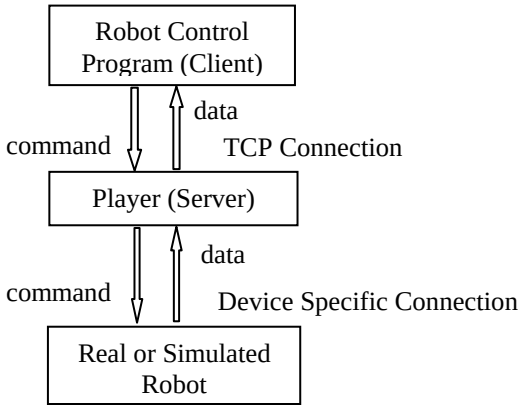


Figure 1: Player System Structure

Stage simulates a population of mobile robots moving in and sensing a two-dimensional bitmapped environment. Various sensor models are provided, including sonar, scanning laser rangefinder, pan-tilt-zoom camera with color blob detection and odometry. Stage devices present a standard Player interface. Few or no changes are required to move between simulation and hardware.

Programming with the Player/Stage, users need to provide two important files: world file and configuration file. The world (.world) file is needed when doing simulation using Stage. It describes things available in the world, including robots, items, and layout of the world. The configuration (.cfg) file contains the robot information called drivers and items in the world file if the client code interacts with them. The real robot drivers are needed to build in Player already and the simulation driver is always Stage.

The user robot control program is a client of the Player server. The client code receives inputs from sensors and controls hardware on robot through so called proxies. So learning proxies is needed to create client codes. Example 1 lists the main function of a client code in C++, which drivers the robot randomly without stop. This code shows a typical program structure of a client code, in which robots are defined and connected to the device proxies, and then in a control loop, sensing and acting interact with proxies. In the control loop of this example, the client code updates the proxies, generates a new random vector of distance and direction of motion every 3 seconds, and then translates it to a speed and a turn rate to drive the robot

```
int main(int argc, char *argv[])
{
  int randcount = 0;
  double speed, turnrate;
  Vector random(0,0);

  PlayerClient robot("localhost");
  Position2dProxy pp(&robot, 0);

  pp.SetMotorEnable (true);
  while(true)
  {
    // update the proxies
    robot.Read();

    // generate a random vector
    wander(randcount, random);

    // compute the speed and the turn rate
    translate(random, speed, turnrate);

    // command the motors
    pp.SetSpeed(speed, turnrate);
  }
}
```

Example 1: Main Function of Random Walk

Note that the control loop runs at about 10Hz, so 30 loop iterations is about 3 seconds. Hence, the wander function generates a new random vector when it is called every 30 times. This is implemented by using the function parameter randcount. Also note that the translate function applies the servo-loop control rules to compute the speed and the turn rate based on the distance and the direction of motion. In this example, it could be simple if the wander function generates a speed and a turn rate directly without using the translate function. But, when more robot behaviors, such as avoid obstacles, are needed, each behavior only needs to produce a vector of distance and direction of motion. Then, vectors from each behavior can be combined together as a single vector by using weighted vector addition.

## 3.2 Robot Programming with the Tekkotsu

Tekkotsu is an application development framework for mobile robots. It provides (1) lower level primitives for sensory processing, smooth control of effectors, and event-based communication, (2) higher level facilities, including an hierarchical state machine formalism for managing control flow in the application, a vision system, and an automatically maintained world map, (3) housekeeping and utility functions, such as timers and profilers, and (4) the newly added Tekkotsu crew [12], which enables programmers to use the built-in higher level robotic functions such as map-making, localization, and path planning.

Tekkotsu is object-oriented, making extensive use of C++ templates, multiple inheritance, and polymorphism (operator overloading). To write a robot control program, users need to define subclasses that inherit from the Tekkotsu base classes, and override any member functions requiring customization.

Two types of fundamental classes in Tekkotsu are behaviors and events. Users need to know the way to response or act when a behavior is constructed, activated, and deactivated, the way for a behavior to listen to events and to process events, and the ways to construct a state machine in Tekkotsu. Users also need to know the concepts of generator, source, and type of an event. Furthermore, when using the Tekkotsu crew, users need to know how to use different types of maps, how to localize the robot, how to detect and/or move to an object of interest, and how to get the location and shape information of objects of interest.

```
$nodeclass Randomwalk : StateNode {
  $nodeclass RandomNode: StateNode : {
    Vector random;
    virtual void doStart() {
      wander(random);
      postStateSignal<Vector>(random);
    }
  }
  $nodeclass TranslateNode: StateNode : {
    Vector random, result;
    double speed, turnrate;
    virtual void doStart() {
      random = extractSignal<Vector>(event);
      translate(random, speed, turnrate);
      result.setVector(speed, turnrate);
      postStateSignal<Vector>(result);
    }
  }
  $nodeclass DriveNode : WalkNode : {
    Vector result;
    virtual void doStart() {
      result = extractSignal<Vector>(event);
      double speed = result.getMagnitude();
      double turnrate = result.getDirection();
      setVelocity(speed, 0, turnrate);
    }
  }
  virtual void setup() {
    $statemachine {
      random: RandomNode
      translate: TranslateNode
      drive: DriveNode;

      random =S<Vector>=> translate
      translate =S<Vector>=> drive
      drive =T(3000)=> random
    }
  }
};
REGISTER_BEHAVIOR(Randomwalk);
```

Example 2: State Machine of Random Walk

In a Tekkotsu state machine, each state has an associated action: speak, move, etc. and transitions are triggered by sensory events, timers, or user's signals. State nodes are behaviors. Entering a state is activating it, and leaving a state is deactivating it. Transitions are also behaviors. A transition starts to work whenever its source state node becomes active. Transitions listen to sensors, timer, or other events, and when their conditions are met, they fire. When a transition fires, it deactivates its source node(s) and then activates its destination node(s).

A shorthand notation is used instead of C++ code to build state machines. The shorthand includes the state node definition, the transition definition, and the state node class definition. The shorthand is turned into C++ by a state machine compiler.

Example 2 lists the major part of a Tekkotsu state machine code of random walk. This application defines three state node classes and a state machine with three state nodes and three transitions. The random node generates a random vector of distance and direction of motion and then send the vector to the translate node, which computes a vector of speed and turn rate and then send the vector to the drive node, which drives the robot with the speed and turn rate for three second (3000 ms) and then transits back to the random node. Note that this application program has the same behavior as the client code in Example 1.

In addition to the concepts mentioned above, the following three basic skills of programming with the Tekkotsu are very important: (1) how to transit from one state to multiple states simultaneously so as to support parallel actions or behaviors, (2) how to transit from one state to one of multiple states based on different conditions so as to make a conditional transition, and (3) how to pass and/or share data among states so as to provide approaches of the data flow and the memory.

# 4 Projects with Using the Player/Stage

This section will present details of the three robot programming projects with using the Player/Stage. The three projects are Waypoint following, target searching, and path planning.

## 4.1 P1: Waypoint 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 to a waypoint, which is very similar to the control loop in Example 1. But, the loop will now terminate after the robot reaches to the waypoint and the wander function in Example 1 is now replaced by the gotoWaypoint function. Students only need to complete the following three C++ functions:

- getWaypoints. It reads a sequence of waypoints from a data file and store 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 is the same as the one in Example 1.

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 the 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 learn 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.

The second task of this project is, instead of simulation, to drive a real iRobot Create robot to follow a square path by using the same client program. Students are also asked to measure the odometry error and make changes to their client code to improve the performance. This exercise makes students get better understanding of uncertainty of sensory data.

## 4.2 P2: Target 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 (for graduate students, previously unvisited) beacon, then move to the location of the beacon. Once the beacon is reached, the robot should go back into 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, which is very similar to Example 1. 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 the following three C++ functions:
   - wander. It is the same as the one in Example 1.
   - 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 two seconds (the control loop runs at about 10Hz, so 20 loop iterations is about 2 second.)
   - gotoBeacon. It returns true whenever a (for graduate students, unvisited) beacon is detected by using Fiducial detector. Otherwise, return false. When a (for graduate students, 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++.

## 4.3 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. If the cell of robot's starting point has the same value i, then the wavefront propagation should be over, otherwise, try to propagate the wavefront to its neighbors. If there is no room to

propagate the wavefront, then the robot's goal point is unreachable; otherwise, the wavefront function can be called again and starts from the grid cells with 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. This function is called first with the current waypoint equal to the robot's starting point. It is called continuously until the new waypoint is the goal point.

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 need to know what is the logic that makes the statement "there is no room to propagate the wavefront" to be true. In addition, students really need to pay attention to array boundaries.

# 5 Projects with Using the Tekkotsu

This section will present details of the two robot programming projects with using the Tekkotsu. These two projects have the same task, which is locating objects inside a maze.

## 5.1 P4: Locate Objects Inside a Maze

The task of this project is taken from the robotics competition held along with the 2nd Annual ARTSI Student Research Conference [1]. It is to get an iRobot Create/ASUS robot to navigate and localize itself within a maze, to announce detected objects and their locations in the maze.

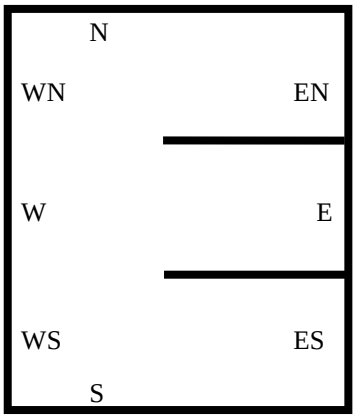


Figure 2: E-Maze with 8 Navigation Markers

The maze is shaped like the letter E as shown in Figure 2: a long corridor with three alcoves branching off from it. There is a bicolor marker at each end of each alcove or corridor, for a total of 8 markers: NE, N, NW, W, SW, S, SE, and E. Objects placed in the alcoves will be red, blue, or green balls, one per alcove. The project is graded on visiting each alcove of the maze, announcing detected objects and their locations, delivering a final report, and the overall run time.

This project is divided into three parts: (1) Travel though the maze by modifying the "following wall" behavior, (2) Travel though the maze and announce the balls the robot detects, and (3) Travel though the maze and report the balls the robot detects and their locations.

In Part 1, students need first to run a sample Tekkotsu's wall following program. Students will notice that the robot will not able to follow the wall unless its right side is placed near a wall, and the robot will not stop. Then, students are required to modify the sample code so that the robot will perform (a) go to a wall, (b) follow the wall, and (c) stop after a while. So the robot can travel through the maze along the walls regardless where the robot is placed inside the maze. Note that a state machine diagram to accomplish this task is given to students.

In Part 2, a sample Tekkotsu state machine code of looking for balls is given to students. This behavior will announce the balls in front of the robot. Students need to test the code with several runs, each run with different sets of color balls and different lighting conditions. Students may need to change the ASUS camera settings through Tekkotsu. Then, students are asked to add the looking for balls behavior in the state machine code in Part 1. So a robot can look for balls while it travels through the maze. Note that looking for balls and traveling through the maze are two parallel behaviors and the former behavior will send a signal to the latter behavior to stop the robot after the robot finds out the three balls. A state machine diagram to accomplish this task is given to students.

Please note that the robot may see the same ball several times. So the robot must have memory to remember which ball it has already seen. Meanwhile the robot should also remember the ordering in which it sees each of the three balls. To this end, a scheme of encoding the three colors is provided to students. In this scheme, Red is labeled as 1, Blue is 2, and Green is 4. A set of colors, for example, {Red, Green} will be labeled as 5 (1+4). A pair of colors, for example, (Blue, Green) will be labeled as 24 (2×10+4). A 3-tuple of colors, for example, (Blue, Green, Red) will be labeled as 241 (2×100+4×10+1).

In Part 3, a sample Tekkotsu state machine code of visiting markers is given to students. In this behavior, the robot will search for a bicolor marker by turning in its place and then move towards the marker; if the robot cannot find a marker after a while, it will move forward. In both cases, the robot will move forward and hit a wall of the maze. Students need to test this code first with the robot placed in different locations inside the maze and different lighting conditions. Students may need to change the ASUS camera settings.

Then, students are asked to modify the sample state machine code so that the robot will only find the bicolor marker at south, at southwest, at west, or at northwest. This means that the markers at the remaining four locations are ignored. Finally, students are asked to add the modified visiting marker behavior into the state machine code in Part 2. Then, students need to modify the announcement and the final report to include the location information of the detected balls. Note that the first ball the robot sees must be in the south alcove, the second must in the middle alcove, and the third must be in the north alcove, if one of the four bicolor markers at south, southwest, west, or northwest is found.

In addition to the above guidance, students must have basic object-oriented programming skills such as deriving a subclass and instantiating a class. Students must also learn Tekkotsu programming basics: behaviors, event, predefined state node classes and transition classes, and state machine formulations and semantics. Moreover, students need to know how to deal with uncertainty and failure. For example, bicolor markers are difficult to detect (false negative) and background blobs are easily recognized as a color ball (false positive).

## 5.2 P5: Another Version of P4

The task of this project is taken from the robotics competition held along with the 3nd Annual ARTSI Student Research Conference [2]. This task is the same as the one in P4 except the bicolor makers are replaced by the AprilTags [6] and the three color balls are replaced by three cubes with each covered by different AprilTags. Note that there are 20 navigation markers (AprilTags) on the walls of the E-maze as shown in Figure 3 so as to support a better result of the robot localization.

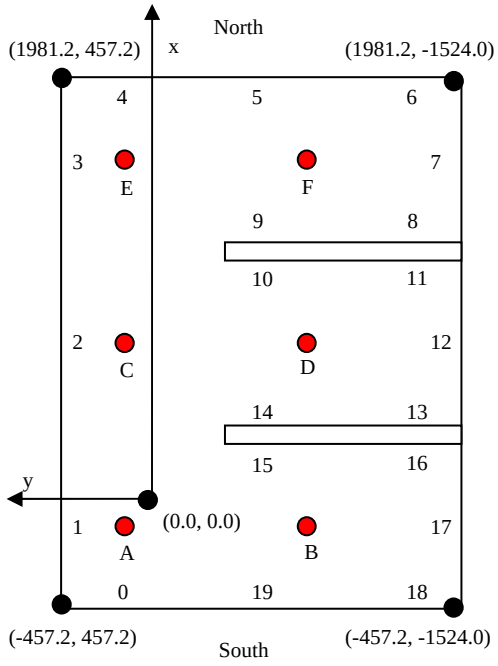


Figure 3: E-Maze, Twenty Navigation Markers (0-19), Six Waypoints (A-F), and World Coordinates of the Four Corners

In P4, each bicolor marker is treated as a topological navigation marker and the measurements of the maze are not used at all. On the other hand, in P5, a metric map of the maze as shown in Figure 3 is used in the robot localization. Meanwhile, moving robot to a particular waypoint inside the maze can be done easily by calling a pilot function that is newly added into Tekkotsu.

This project is divided into four parts: (1) Memorize and report which object the robot has observed in each alcove, (2) Look for the object in each alcove, (3) Drive robot to a waypoint in the maze, and (4) Putting it together.

In Part 1, students need first to run a sample Tekkotsu's pilot demonstration program. Students can use commands provided in this program to drive the robot forward or making a turn, check the robot location, localize the robot, and look AprilTags that are facing to the robot and report their IDs.

In this part, students are asked to add a command in this sample code to report which object the robot has observed in each alcove. Here, the robot needs to figure out which object in which alcove. We assume that when the robot is seeing an object in an alcove, it will also see at least one navigation marker on the wall of the same alcove. This may generally not be true. But, users can use commands to make the robot facing to an object and at least one marker at the same time. Note that students need to define three shared variables to remember the objects the robot has observed in each alcove.

In Part 2, students are asked to add a command to look for an object and its location. Now, we assume that the robot is inside an alcove, but it may not face an object. This new command allows the robot to look for an object for several times by making several turns until it finds one as well as its location or fails to find.

In Part 3, students are asked to add a goto <x y> command to drive the robot from its current position to location (x, y) inside the maze. Please note that the world coordinates of points in the maze should be used in the goto function. The coordinates of the four corners are shown in Figure 3. Note that the coordinates are using millimeters as measurement unit. Students can use the built-in path planning and execution function of the Tekkotsu pilot to implement the goto command. Or, students can implement the goto function on their own. In this case, we assume there is no obstacle in between the robot current location and the target location.

Due to the uncertainty, the robot may not be able to be close enough to the target location. In this case, the goto function should redo itself again until the robot is close to the target within a threshold distance (for example, 200 mm). If the robot is still not able to reach the target after redoing 3 times, the goto function should be stopped and return a failure.

Due to the same reason as above, the robot may hit walls before reaching to the target. In this case, the robot should backup a little bit and then the goto function should redo itself again. If the robot is still not able to reach to the target after

redoing 3 times, the goto function should be stopped and return a failure.

In Part 4, students are asked to put them together by creating a node class that takes a role of subtask scheduler. For example, a schedule can be goto A, goto B, look for object, goto A, goto C, goto D, look for object, … and finally report which object the robot has observed in each alcove. It is obvious that this schedule will let the robot find the object at each alcove, if each subtask is successfully executed.

# 6 Discussion and Conclusion

Our robotics course has offered three times in the fall semesters over three years starting from 2009. It covers all major topics on intelligent mobile robots, including robot control architectures, navigation, localization, planning, sensing, and uncertainty. A primary component of this course is robot programming. Over the three years of time, many robot programming projects have been adopted, revised, and developed for underrepresented students at HBCUs. In addition to the major projects presented in this paper, we have created several introductory projects in our robotics course for learning the basics of the Player/Stage and the Tekkotsu.

Teaching robot programming is challenging. More in-class teaching is always welcoming by students. Like most HBCUs, we don't have open labs and TAs for our robotics projects. Sometimes, the instructor has to spend extra hours in the lab to help students with their projects. Providing detailed and intuitive guidance to students is important and neccessary. Therefore, students will see the hope to complete the project and then spend their time doing the project.

In our robotics courses in 2009 and 2010, we used both Player/Stage and Tekkotsu. The first version of the project of localizing objects in a maze was added into our 2010 robotics course. But, we were left no time to do this project. In our 2011 robotics course, we used the Tekkotsu only. So, we got time to cover the second version of the project of localizing objects in a maze. Note that AprilTags are much easier to be detected than bicolor markers. This makes the robot localization more accurate. So, we can use the Tekkotsu built-in robot localization function in the second version. But, we can apply the method used in the first version to the second version without using the robot localization.

Compared with the Tekkotsu, the Player/Stage provides more general framework for robot programming. Users are easy to start with it and to test their own algorithms. On the other hand, the Tekkotsu builds a set of high-level interacting software components to relieve a programmer of the burden of specifying low-level robot behaviors [12]. This makes it possible to teach and practice more on robotics rather than programming details. But, there is a large learning curve to master the Tekkotsu fundamentals and its high-level software components. However, Tekkotsu is easy to use for the demonstration of robotics concepts. Meanwhile, the 3-D simulation software Mirage can be used along with the Tekkotsu for the simulation.


### Acknowledgments

This work was supported by National Science Foundation award CNS-1042326. The author thanks Dr. David Touretzky for his help on the Tekkotsu.


# 7 References


[1] ARTSI Alliance, *2010 Robotics Competition*, Available at http://artsialliance.org/2010-Robotics-Competition

[2] ARTSI Alliance. *2011 Robotics Competition*. Available at http://artsialliance.org/2011-Robotics-Competition

[3] A. Dollar, D. Rus, and P. Fiorini, *Robotics Courseware*, Available at http://roboticscourseware.org

[4] B. Gerkey, R. Vaughan, K. Støy, A. Howard, G. Sukhatme, and M. Mataric, *Most Valuable Player: A Robot Device Server for Distributed Control*, Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2001), pp. 1226-1231, Wailea, Hawaii.

[5] X. Liang, *Introduction to Robotics* (*Fall 2011, Fall 2010, Fall 2009*), Available at http://www.jsums.edu/robotics/

[6] E. Olson, *AprilTag: A robust and flexible visual fiducial system*, Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2011), Shanghai, China, May 2011.

[7] L. Parker, Software for Intelligent Robots, Available at http://roboticscourseware.org/fullcourses/mikey-test-course

[8] E. J. Tira-Thompson, G. V. Nickens, and D. S. Touretzky, *Extending Tekkotsu to new platforms for cognitive robotics*, Proceedings of the 2007 AAAI Mobile Robot Workshop, pp. 47-51, Menlo Park, CA.

[9] C. Toby, M. Bruce, and G. Brian, *Player 2.0: Toward a Practical Robot Programming Framework*, Proceedings of the Australasian Conference on Robotics and Automation (ACRA 2005), Sydney, Australia.

[10] D. S. Touretzky, *iRobot Create/ASUS Notebook Platform for Tekkotsu*. Available at http://chiara-robot.org/Create/

[11] D. S. Touretzky, and E. J. Tira-Thompson, *Tekkotsu: A framework for AIBO cognitive robotics*, Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05), volume 4, pp. 1741-1742. Menlo Park, CA, 2005.

[12] D. S. Touretzky, and E. J. Tira-Thompson, *The Tekkotsu "Crew" Teaching Robot Programming At A Higher Level*, Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10), pp. 1908-1913, Atlanta, GA, 2010.