Coaching Robotics Competitions with Tekkotsu

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

Abstract - *The robotics competitions held along with the Annual ARTSI Students Research Conferences were designed to encourage undergraduate students from non-traditional backgrounds in the study of robotics in areas that are relevant to society. The tasks in these competitions were challenging, including robot searching and localizing itself inside a maze, finding canisters and pushing them into pen with paddles, and finding canisters, pickup them up with grippers, and transport and drop them to a goal location. These tasks are involved in robot sensing, navigation, manipulation, localization, and etc. This paper will breakdown these tasks into pieces and present detailed steps to complete each task so that making these tasks accomplishable by our robotics team students and suitable for being major robot programming lab projects in an upper-level undergraduate robotics course for underrepresented students. Meanwhile, the Tekkotsu robot programming tool will be also introduced for programming the robot to accomplish these competition tasks.*

Keywords: Robotics Education, Robotics Competition, Robot Programming, Tekkotsu

1 Introduction

Teaching an upper-level undergraduate robotics course at Historically Black Colleges and Universities (HBCUs) is challenging because of the lack of suitable teaching materials, especially the robot programming lab projects. Thanks to the ARTSI (Advancing Robotics Technology for Societal Impact) Alliance [1]. It provided a family of unified robot platforms called Calliope [2-4] to each HBCU schools in the alliance, and the technical support to the robot programming tool called Tekkotsu [5-6]. More important and helpful are robotics competitions held each year along with the Annual ARTSI Students Research Conferences since 2010 [7-10]. The tasks in these competitions include robot searching and localizing itself inside a maze, finding canisters and pushing them into pen with paddles, and finding canisters, pickup them up with grippers, and transport and drop them to a goal location. These tasks are involved in robot sensing, navigation, manipulation, localization, and so on. These tasks can be excellent hands-on learning materials for our upper-level undergraduate robotics course. However, these tasks are quite challenging to our students. A detailed guidance to lead to solve the problem is desirable.

This paper will breakdown these tasks into pieces and present detailed steps to complete each task so that making these tasks accomplishable by our robotics team students and suitable for being major robot programming lab projects in an upper-level undergraduate robotics course for underrepresented students. In fact, three of the four competition tasks have been already transformed to our robot programming lab projects [13].

It should be pointed out that all the tasks mentioned early are related to the robot vision in a sense that robots need to use their visions (cameras) to sense their environment to act to achieve their goals. These robotics applications require that a robot can be programmed to be able to know what it wants to see, to recognize what it observed, to compute the orientation, the size, and the location of objects it detected, to reason about the spatial relationship among observed objects, to figure out the geographical layout of the objects, and to build the map. These applications are apparently very interesting to computer science students, and able to broaden their understanding of computer science concept and encourage them to learn more. However, programming a robot in these applications from scratch can be very difficult. Fortunately, the Tekkotsu robot programming development environment provides users with a set of high level supports, including vision processing for shape extraction, shape predicates, and mapmaking. Hence, the Tekkotsu robot programming tool will be introduced for programming the robot to accomplish these competition tasks in this paper.

The Calliope robot family used in our robotics course and competitions include Calliope basic model, formerly called the Create/ASUS robot, and Calliope2SP which added 2-DOF arm, Sony Eye webcan, and pan/tilt on the basic model [2-4]. The iRobot Create 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 netbook computer sitting on top of iRobot Create is functioned as the brain of the Calliope robot. The Tekkotsu are installed on the ASUS computer.

In the rest of this paper, the robot programming with using the Tekkotsu will be introduced first in Section 2. Four robotics competition tasks will be detailed and discussed from Section 3 to Section 6, each with one section, respectively. In the first two tasks, 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 tasks. In the last two tasks, a robot needs to locate canisters and move them to a goal location. But, one task will use two attached paddles to push canisters and the goal location is a square region marked with colored tapes on the ground. The other task will use the gripper to pickup canisters and the goal location is marked by an AprilTag on the wall. Note that the discussions for each task will focus on two aspects: 1. task and detailed steps to guide students to accomplish it, and 2. necessary knowledge for completing the tasks, including programming skills, mathematical formulas, algorithms, and issues regarding to failures and uncertainty. Finally, a short discussion and conclusion is presented in Section 7.

2 Introduction to 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], including Lookout, MapBuilder, Pilot, and Grasper, 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.

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 notation includes the state node definition, the transition definition, and the state node class definition. The shorthand is turned into C++ code by a state machine compiler.

Example 1 lists a Tekkotsu state machine code that describes a behavior similar to a 2-bit free-running counter circuit. It says zero, one, two, and three and then backs to zero again. This application defines a state machine with four state nodes and four timer transitions, each with 1 second period. SpeechNode is a built-in node class in Tekkotsu. Note that there are many built-in node classes and transitions in Tekkotsu. Users can also define their own node classes, but rarely need to define their own transition classes.

```
#include "Behaviors/StateMachine.h
$nodeclass Counter4 : StateNode {
  $setupmachine {
    zero: SpeechNode("zero")
    one: SpeechNode("One")
    two: SpeechNode("two")
    three: SpeechNode("three")

    zero =T(1000)=> one =T(1000)=> two
    =T(1000)=> three =T(1000)=> zero
  }
}
REGISTER_BEHAVIOR(Counter4);
```

Example 1: State Machine Code of a 2-bit Counter

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.

```
$nodeclass Counter4R : StateNode {
  $provide int b(1);
  enum Next {
    forward,
    backward
  };
  $nodeclass ButtonPressNode : StateNode {
    virtual void doStart() {
      erouter->addListener(this,EventBase::button);
    }

    virtual void doEvent() {
      $reference Counter4R::b;
      b = 1-b;
    }
  }
  $nodeclass NextNode() : StateNode : doStart {
      $reference Counter4R::b;
      if (b==1)
         postStateSignal<Next>(forward);
      else if (b==0)
         postStateSignal<Next>(backward);
  }
```

Example 2: Partial Code of 2-bit Counter with Reverse

Example 2 lists a part of state machine code that describes a behavior of 2-bit counter with a reverse button. The counter will count in a forward order 0, 1, 2, 3 initially. Whenever the button is pressed, the counting order will be reversed. In this example, the ButtonPressNode node class is defined to keep track the button event. Whenever the button is pressed, the shared variable b will change its value. Note that the button press node will be always active. Thus, this node is concurrent with other state nodes. To this end, a parallel transition from the start node like startn =N=> {zero, button} is needed. The NextNode node class is defined to decide the next state based on the current order (forward or backward) via the value of the shared variable b. Thus, one of the two state transition signals is posed according to the value of b.

3 Locate Color Balls Inside a Maze

The project is taken from the robotics competition task held along with the 2nd Annual ARTSI Student Research Conference [7]. This task is to get an iRobot Create/ASUS robot [2] to navigate and localize itself within a maze, to announce detected objects and their locations in the maze.

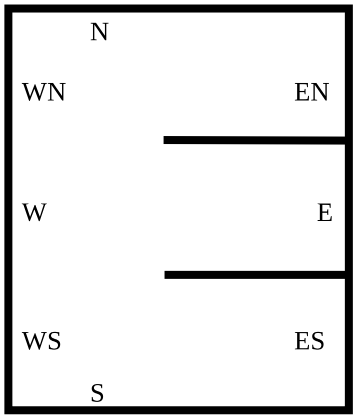

Figure 1: E-Maze with 8 Navigation Markers

The maze is shaped like the letter E as shown in Figure 1: 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.

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 all 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 ($2\times10+4$). A 3-tuple of colors, for example, (Blue, Green, Red) will be labeled as 241 ($2\times100+4\times10+1$).

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 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).

4 Locate AprilTags Inside a Maze

The project is taken from the robotics competition held along with the 3nd Annual ARTSI Student Research Conference [8]. This task is the same as the one in Second 3 except the bicolor makers are replaced by the AprilTags [11] 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 2 so as to support a better result of the robot localization.

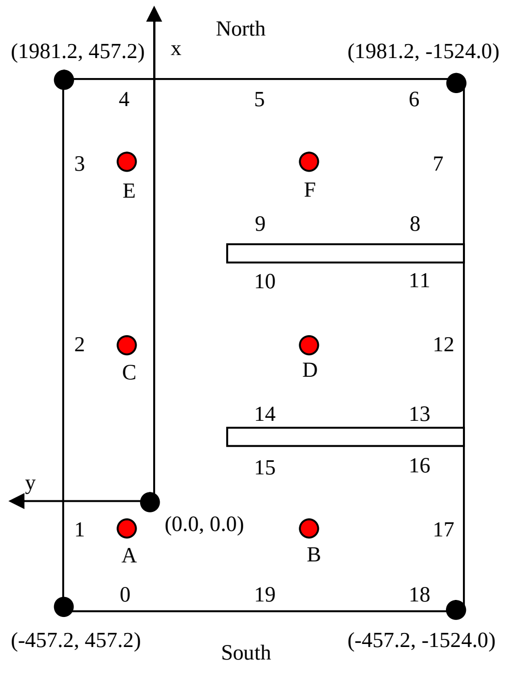

Figure 2: E-Maze, Twenty Navigation Markers (0-19), Six Waypoints (A-F), and World Coordinates of the Four Corners

In Section 3, 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 this project, a metric map of the maze as shown in Figure 2 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 be facing 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 2. Note that the coordinates are using millimeters as measurement unit. Students can use the built-in path planning and execution function through using 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 location. 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.

5 Push Canisters into Pen

The project is taken from the robotics competition held along with the 4th Annual ARTSI Student Research Conference [9]. As shown in Figure 3, five red canisters will be scattered around a 2-by-2 meter space. Near the center of the space is a 0.75-by-0.75 meter "pen" drawn with blue masking tape. The task is to get a robot with two added aluminum paddles to locate the canisters and push all of them into the pen as quickly as possible.

This project is divided into five parts: (1) Locate and move to the center of the pen, (2) Locate a canister and move to the canister, (3) Push one canister into the pen when the robot is facing a canister and a corner of the pen, (4) Push all canisters into pen, assuming that the robot is able to see a pen corner and a canister outside the pen at the same time, (5) Push all canisters into the pen with no assumption in (4).

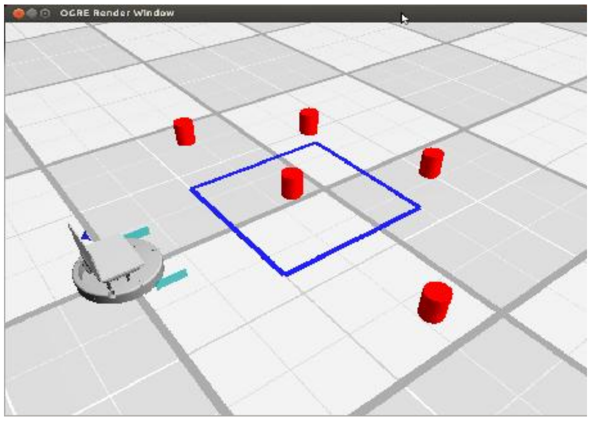

Figure 3: Push Canisters into Pen

In part 1, a sample state machine code is given to students. This code will compute the center of the pen based on a corner of the pen and then drive the robot to the center of the pen, when the robot is facing a pen corner. Students are asked to study the code and run the code in the following settings: (1) the robot facing more than one pen corner, (2) the robot facing only one pen corner, and (3) the robot facing no corner. Then, students are asked to modify the code so that the robot can search for a pen corner if no pen corner is visible to the robot, and then compute the center of the pen and drive the robot to the pen center. In case that there is no pen at all, the robot should report "no pen".

Part 2 is very similar to Part 1. A sample state machine code that drives the robot to a canister that the robot is looking at is given to students. Then, students will add a search function so that the robot will search a canister first and then drive to it.

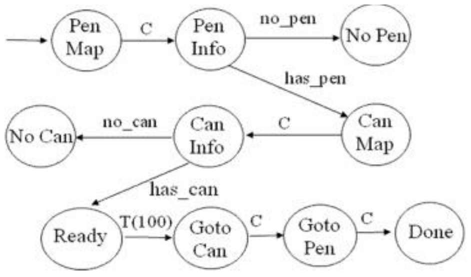

Figure 4: State Machine Diagram in Part 3

In Part 3, students are asked to write a state machine code to push one canister into the pen center according to a given state machine diagram as shown in Figure 4. If the robot can see a pen corner and a canister outside the pen simultaneously, then the robot will move to the canister first and then push it into the pen. If the robot cannot see either any pen corner or any canister, then it reports “No Pen” or “No Can". Meanwhile, students are asked to figure out the formulae to compute the angle and distance to the canister and the angle and distance to the pen center in the robot's local shape space (local map) as shown in Figure 5.

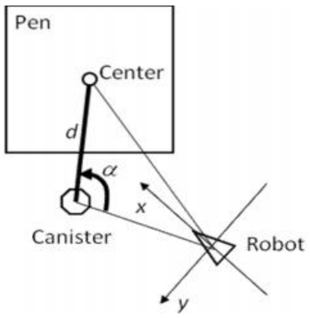

Figure 5: Center of Pen and Canister in Local Map

In Part 4, the code in Part 3 is to add a search function first according to a given state transition diagram shown in Figure 6. If the robot can see a corner of the pen and a canister simultaneously, then the robot will move to the canister first and then push it into the pen and say “Done”. If the robot does not see either a corner or a canister, then the robot will makes a turn a little bit (say, 30 degree) and then continue to look for a corner and a canister. If the robot is still unable to find a corner and a canister simultaneously after making several turns (say, 12), then the robot should stop searching and report “Fail”.

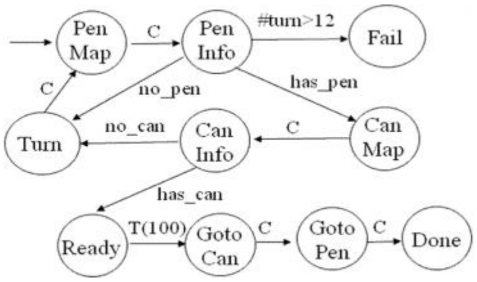

Figure 6: State Machine Diagram in Part 4

Next in Part 4, students are asked to extend the above code further to push all canisters into pen. After one canister is pushed into the pen, the robot should backup and get out of the pen, and then repeat for a next canister until all canisters are inside the pen. Note that the distance between a canister and the pen center can be used to judge if the canister is inside the pen or not.

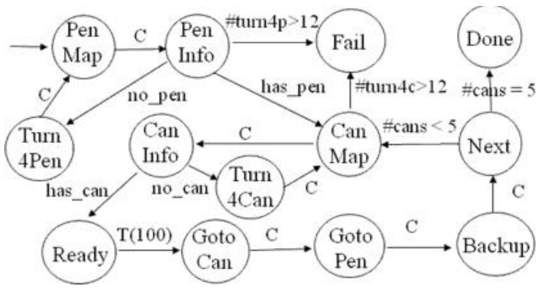

Figure 7: State Machine Diagram in Part 5

In Part 5, the assumption that the robot is able to see a canister and a pen corner simultaneously is removed. Thus, the robot should detect a pen corner and compute and memorize the pen center first. Then, whenever the robot moves, the coordinate transform of the pen center should be performed so as to keep the pen center always inside the current local shape space. Figure 7 shows the state transition diagram that students should implement for their final version of the code.

6 Pickup Canisters

The project is taken from the 5th robotics competition held along with the Fifth Annual ARTSI Student Research Conference [10]. As shown in Figure 8, this task will be done using a Calliope2SP robot in a 1 meter by 2 meter rectangular arena with white walls and an AprilTag on one short wall marking a goal location. Three colored cylinders will be scattered around the arena, far from the goal. The robot must locate each cylinder, pick it up using its gripper, and transport it to the goal location, defined as by a 2x2 foot box below the AprilTag.

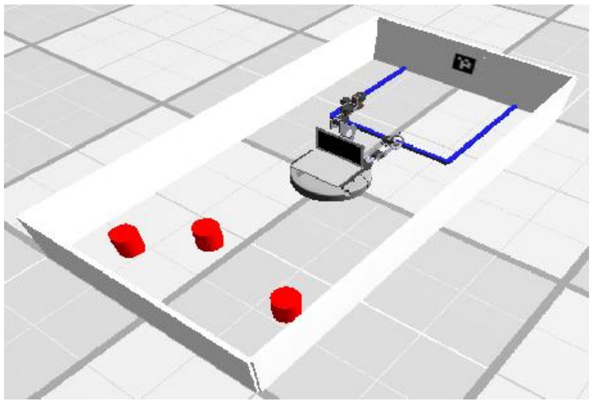
Figure 8: Pickup and Putdown Canisters

This project is divided into five parts: (1) Locate an AprilTag and move close to it, (2) Pickup a canister when it is placed in between the two fingers of the gripper, (3) Locate a canister, move to the canister, and pickup the canister, (4) Locate a canister, move to it, pickup it, transport it to the goal location, and drop it, and (5) Add memory to the robot.

In Part 1, a sample state machine code is given to students. This code will obtain the location of AprilTag in front of the robot and drive the robot to this location. Students are asked to test this code and then modify the code so that the robot can search for an AprilTag, if no tag is visible to the robot. It is also possible that there is no AprilTag at all. In this case, the robot should report that there is no tag.

In Part 2, a state machine code is given to students. This code will make the robot in a getting ready posture by using the Tekkotsu's Dynamic Motion Sequence Node. In this posture, the robot's head looks ahead, the robot's arm points ahead, and the robot's gripper is open and in a low position. Students are asked to study the code and then add two more motion sequences: Pickup and Putdown so that if a canister is placed in between the gripper's two fingers, the robot should be able to pick the canister up. Note that students may need to test the code several times to adjust the distance in between two fingers until the robot can pick the canister up.

In Part 3, a sample state machine code is given to students. This code will detect canisters and drive the robot to the canister that is closest to the robot. If there is no canister, the robot will report there is no can. As shown in Figure 9, the grippers is northeast to the robot center. In order to make the two fingers of the gripper close enough to the canister after the robot moves to the canister, the robot should follow the route with red color in Figure 9. To this end, students need to give the formulae to calculate D, β, and θ. Note that (xg, yg) can be obtained by measuring the Calliope2SP robot, d is a constant which is a little longer than the gripper's finger, and (xc, yc) can be obtained by using Tekkotsu. Note that students need to test this code and their formulae multiple times to adjust their formulae. Finally, students are asked to write a new state machine code so that the robot can locate a canister, move to the canister, and pickup the canister.

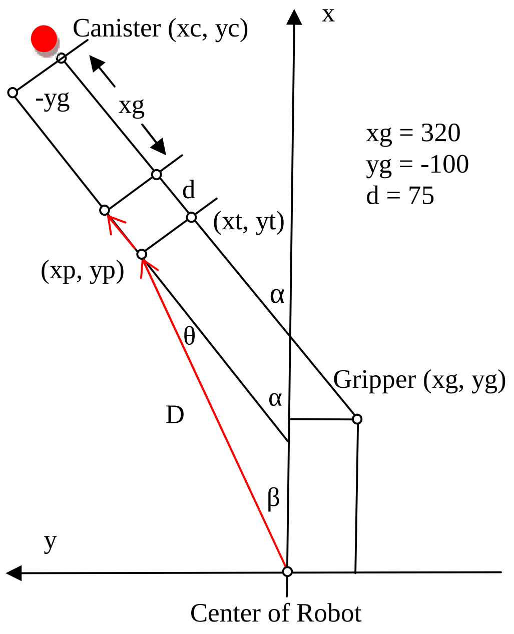

Figure 9: Center of Robot, Gripper, and Canister

In Part 4, students are asked to write a state machine code that simply combine their codes in Part 3 and Part 1 together so that the robot can locate a canister, move to it, pick it up, and transport it to the goal location that is close to the AprilTag and drop it. After testing this code, students will extend it so that the robot will continue its work until it finishes to pickup, transport, and drop all three canisters. Note that students are asked to make sure the robot will not pick up a canister again that has been already transported to the goal location. This may need some assumptions if the robot has no memory.

In Part 5, students are asked to add a shared variable to memorize the AprileTag location. Therefore, the robot just needs to search for the AprilTag once and it will know a canister has been already transported to the goal location, if the canister is close enough to the AprileTag. In this case, whenever the robot moves, the coordinate transform of the AprilTag location should be performed so as to keep it always inside the current local shape space. Figure 10 shows the state transition diagram that students should implement for their final version of the code.

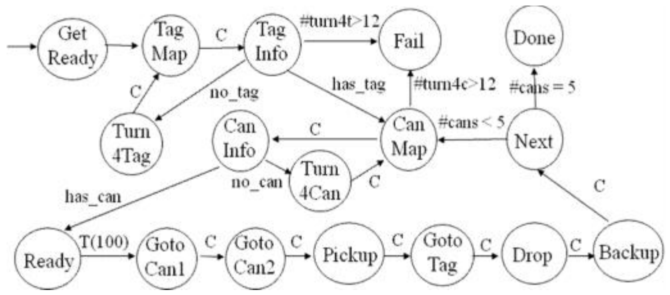

Figure 10: State Machine Diagram in Part 5

7 Discussion and Conclusion

Our robotics team at JSU has participated in the robotics competitions held along with the ARTSI Annual Student Research Conferences four times since 2010. Three of the four competition tasks have already been transformed into our major robot programming lab projects. Our robotics course has been offered in the fall semesters four times starting from 2009 [13]. It covers all major topics on intelligent mobile robots, including robot control architectures, navigation, localization, planning, sensing, and uncertainty.

The ARTSI robotics competitions have provided a unique opportunity to encourage undergraduate students from non-traditional backgrounds in the study of robotics in areas that are relevant to society. They have also provided rich materials for robotics education. Coaching students for preparing for such competitions are challenging and time-consuming, but it also gives coaches an opportunity to learn how to help their students to solving the challenging problems.

The robot programming development tool Tekkotsu builds a set of high-level interacting software components. Thus, it relieves a programmer of the burden of specifying low-level robot behaviors [12]. This makes it possible to accomplish challenging robotic applications and to teach and practice more on robotics rather than programming details. In addition, the 3-D simulation software Mirage can be used along with the Tekkotsu for the simulation. But, there is a large learning curve to master the Tekkotsu fundamentals and its high-level software components. The Tekkotsu tool is under the constant construction and changes a lot over time.

One of the biggest problems we were facing in the robotics competitions is uncertainty. The environments have a huge impact on the robot performance. All our robotics competition tasks are related to the robot vision. The robots are required to recognize bicolor markers, color balls, and color cylinders. However, these jobs are sensitive to the lighting conditions in the competition site. There are a lot false negatives for bicolor makers and a lot false positives for color balls and cylinders. But, the AprilTags detection is very accurate and stable. This makes the robot localization more accurate. So, we applied the Tekkotsu built-in robot localization function in the second robotics competition.

Finally, there could be other alternative ways to solve the robotics competition problems. For examples, the world map could be used in the third and fourth competitions. Therefore, it is easy for robot to memorize the locations of the pen center and the AprilTag. But, maintaining a world map in Tekkotsu makes it slower and adds a little more uncertainty. Thus, we selected to use the local map. This also leave a chance to let users to keep track the objects they need.

Acknowledgments

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

8 References

[1] ARTSI Alliance, Available at http://artsialliance.org/

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

[3] D. S. Touretzky, *Calliope Mobile Manipulation Platform for Tekkotsu*, Available at http://chiara-robot.org/Calliope/

[4] D. S. Touretzky, *Calliope Robot*, Available at http://wiki.tekkotsu.org/index.php/Calliope_Robot

[5] 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.

[6] 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.

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

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

[9] ARTSI Alliance, *2012 Robotics Competition*, Available at http://artsialliance.org/2012-Robotics-Competition

[10] ARTSI Alliance, *2013 Robotics Competition*, Available at http://artsialliance.org/2013-Robotics-Competition

[11] 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.

[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.

[13] Xxuejun Liang, *Introduction to Robotics*, Available at http://www.jsums.edu/robotics/