Coaching Robotics Competitions with Tekkotsu

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

Outlines

A. Introduction
B. Robot Programming with Tekkotsu
C. Locate and Report Color Balls Inside a Maze
D. Locate and Report AprilTags Inside a Maze
E. Search and Push Canisters into a Pen
F. Search, Pickup, and Transport Canisters
G. Discussion and Conclusion

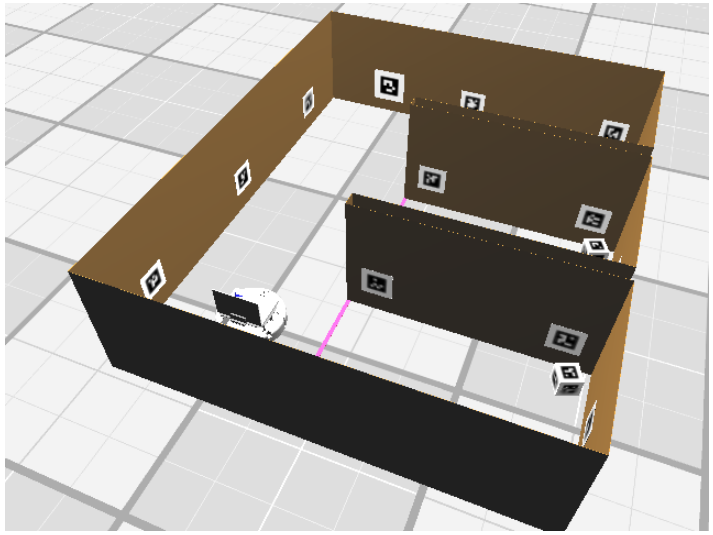
A: Introduction

- The robotics competitions were held along with the Annual ARTSI (Advancing Robotics Technology for Societal Impact) Students Research Conferences.
- The mission of the ARTSI Alliance was to provide education and research opportunities to engage undergraduate students from non-traditional backgrounds in the study of robotics in areas that are relevant to society.
- ARTSI has been newly combined with other NSF BPC Alliances (A4RC and EL Alliance) and Demonstration Projects (AARCS) to become a new NSF project: Institute for African-American Mentoring in Computing Sciences (iAAMCS)

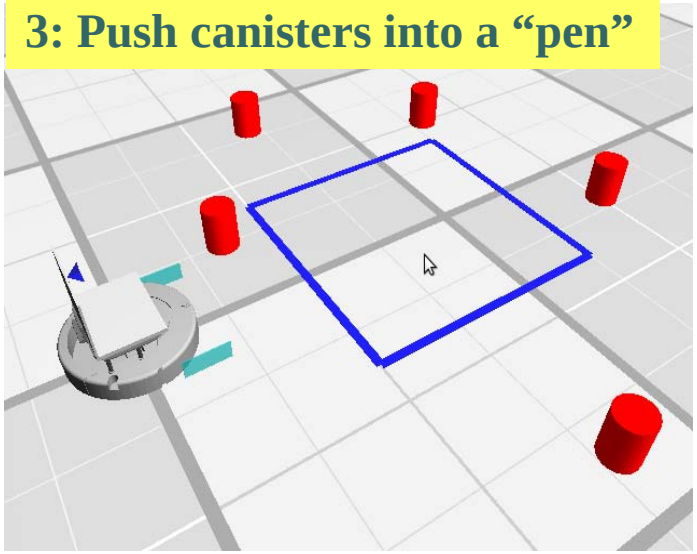
Robotics Competition Tasks

They are involved in robot sensing, navigation, manipulation, localization, and so on.

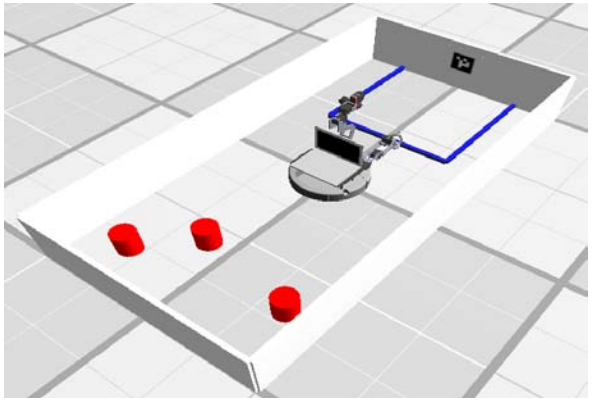
1,2: Search objects inside a maze and Report Locations of objects



3: Push canisters into a "pen"

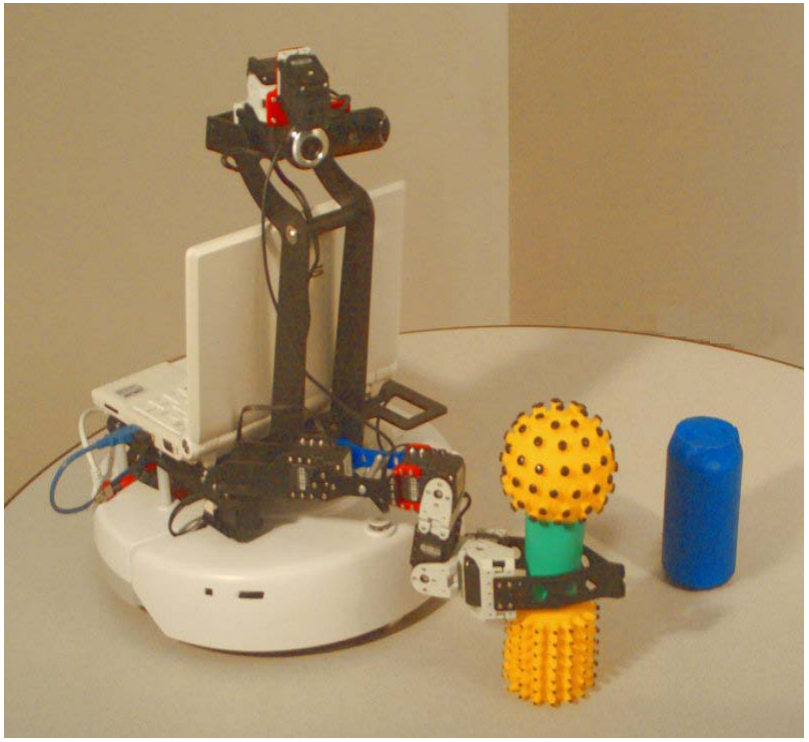


4: Pickup and transport canisters



Robots used in ARTSI Alliance

ASUS Net book PC
on top of
IRobot Create Robot



Calliope
Built from
ASUS/Create

B. Programming with Tekkotsu

- It provides lower level primitives for
  - Sensory processing,
  - Smooth control of effectors, and
  - Event-based communication
- It provides higher level facilities, including
  - A hierarchical state machine formalism for managing control flow in the application,
  - A vision system,
  - An automatically maintained world map, and
  - Newly added Tekkotsu crew, including
    Lookout, MapBuilder, Pilot, and Grasper

Behaviors, Events, and State Machine

- Application programmer simply define subclasses that inherit from the Tekkotsu base classes, and override any member functions requiring customization
- Behaviors and Events
  - Behaviors: Construct, Activate, Deactivate, Listen Events, Process Events,
  - Events: Generator, Source, Type
- State Machine
  - Robot moves from state to state.
  - Each state has an associated action: speak, move, etc.
  - Transitions triggered by sensory events or timers.

State and Transition

- State nodes are behaviors
  - Enter (Activate), Leave (Deactivate), Listen Events, Process Events
- Transitions are also behaviors
  - A transition starts to work whenever its source state node becomes active.
  - Transitions listen for sensor, 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).

Shorthand Notation

- A shorthand notation is used instead of C++ code to build state machines. The shorthand is turned into C++ by a state machine compiler.
- Node definition:

  label: ClassName`(args)[inits]
- Transition definition:

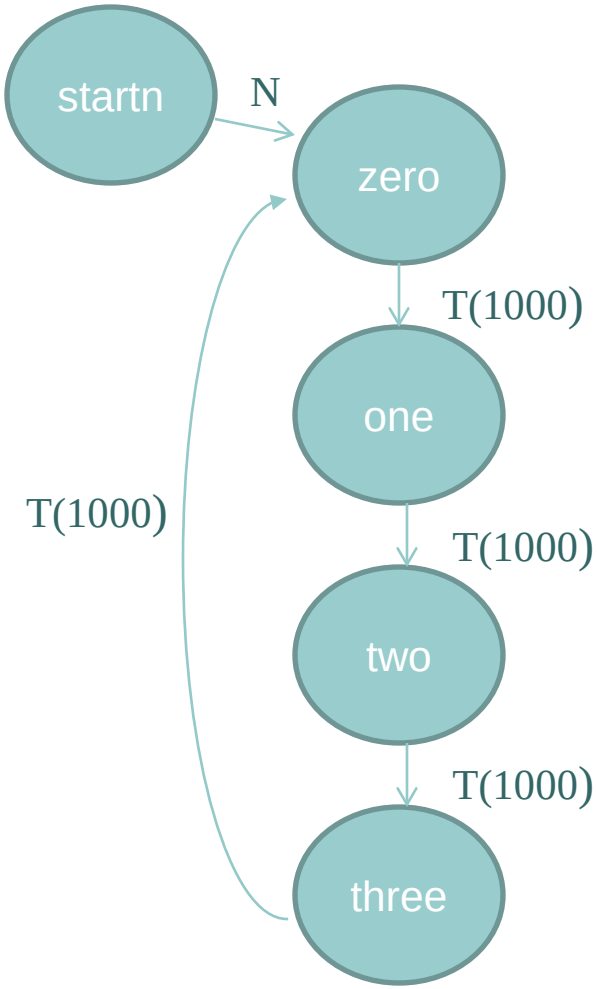
  =label:TransAbbrev`(args)[inits]=>
- Node Class definition

```
$nodeclass ClassName(parameters) : ParentName(args) :
 initializers : methodname {
    body
}
```

Advanced Knowledge of Tekkotsu

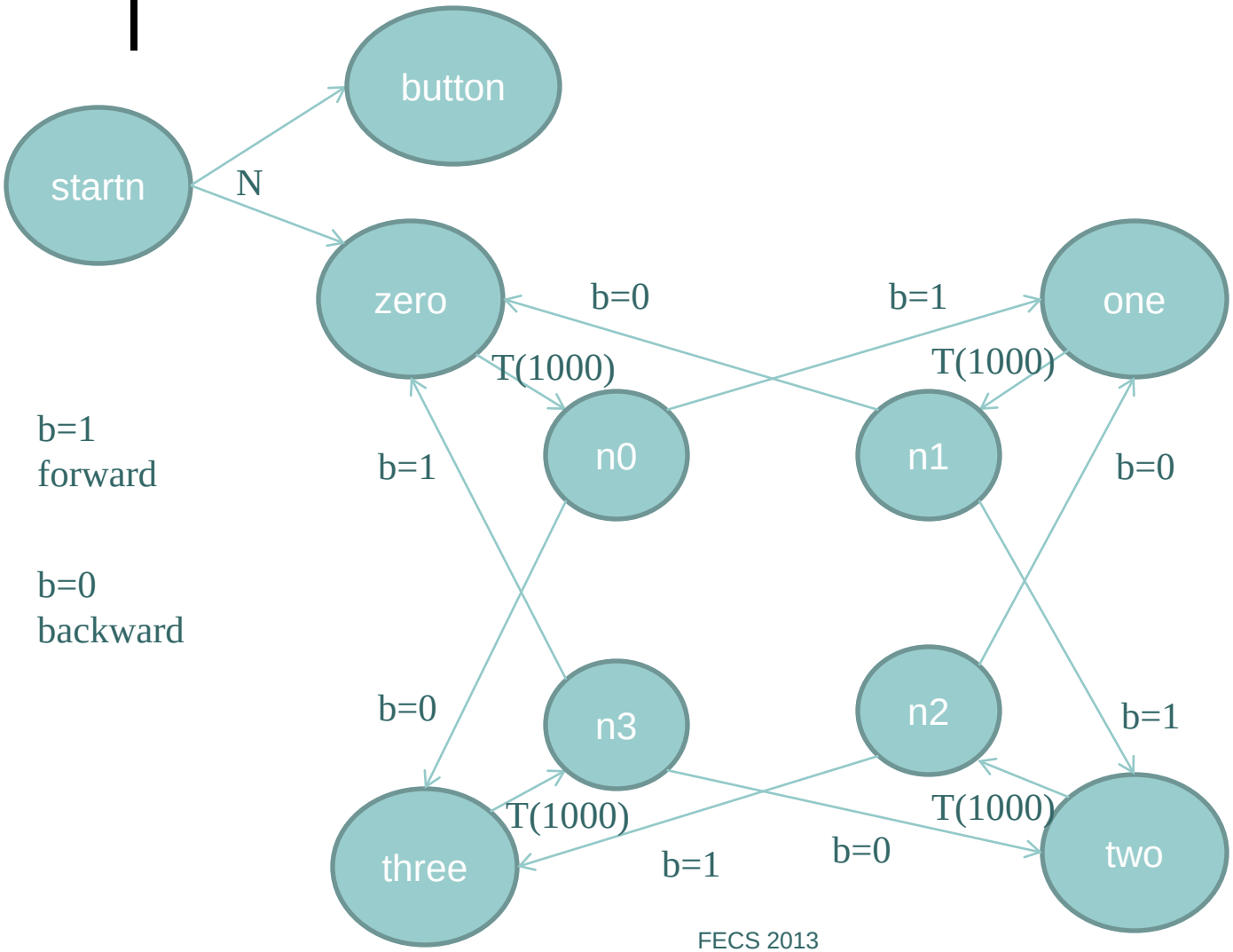
- Transit from one state to multiple states simultaneously so as to support parallel actions or behaviors,
- Transit from one state to one of multiple states based on different conditions so as to make a conditional transition, and
- Pass and/or share data among states so as to provide approaches of the data flow and the memory

Example: Counter4




```
#include "Behaviors/StateMachine.h"

$nodeclass Counter4 : StateNode {

  $setupmachine {
    startn: StateNode
    zero: SpeechNode("zero")
    one: SpeechNode("One")
    two: SpeechNode("two")
    three: SpeechNode("three")

    startn =N=> zero =T(1000)=> one
     =T(1000)=> two =T(1000)=> three
     =T(1000)=> zero
  }
}

REGISTER_BEHAVIOR(Counter4);
```

Example: Counter4 with Reverse



Counter4 with Reverse (Cont.)

Parallel transition

startn =N=> {zero, button}

Branch transition

n1 =S<Next>(forward)=> two
=S<Next>(backward)=> zero

```
$nodeclass Counter4R : StateNode {
  $provide int b(1);

  enum Next {
    forward,
    backward
  };

  $nodeclass ButtonPressNode : StateNode {
    virtual void doStart() {
      erouter->addListener(this, EventBase::botton … );
    }

    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);
  }
  ……
```

Shared Variable

Drive Robots with Pilot

- Request Types
  - walk – essentially a WalkMC request
  - waypointWalk – provides Waypoint walk functionality
  - setVelocity – set speed and go forever
  - localize – look for landmarks and invoke the particle filter
  - goToShape – path plan and travel to the location of a shape on the world map
  - *More functions are planned...*

Walk Requests in Pilot

- pilotreq.dx
  - Forward distance in mm (negative means go backward)
- pilotreq.dy
  - Sideways distance in mm (positive to the left)
- pilotreq.da
  - Rotation angle in radians (positive is counterclockwise)
- pilotreq.forwardSpeed
  - Translation speed in mm/sec for @a dx or @a dy
- pilotreq.strafeSpeed
  - Sideways translational speed used for setVelocity
- pilotreq.turnSpeed
  - Rotational speed in radians/sec for @a da
- pilotreq.collisionAction
  - collisionIgnore;

Tekkotsu State Machine Code: Travel Along a Unit Square

```
$nodeclass PilotLab : VisualRoutinesStateNode {

  $nodeclass Forward1000 : PilotNode(PilotTypes::walk) : doStart {
    pilotreq.dx = 1000;
  }
  $nodeclass Left90 : PilotNode(PilotTypes::walk) : doStart {
    pilotreq.da = M_PI/2;
  }

  $setupmachine {
    Forward1000 =C=> Left90 =C=>Forward1000 =C=> Left90 =C=>
    Forward1000 =C=> Left90 =C=>Forward1000 =C=> Left90
  }
}
```

Programming the MapBuilder

```
$setupmachine {
    BuildMap =C=> ExamineMap
}
```

- Use a MapBuilderNode to submit a MapBuilder request to indicate what you want the Map Builder to do, e.g., find blue lines and project them to local space.

```
$nodeclass BuildMap :
    MapBuilderNode(MapBuilderRequest::localMap) : doStart {
        mapreq.addObjectColor(lineDataType,"blue");
}
```

- Use any subsequent state node to examine the results.

```
$nodeclass ExamineMap : VisualRoutinesStateNode : doStart {
    cout << "MapBuilder found" << localShS.allShapes().size()
    << " blue lines." << endl;
}
```

MapBuilderRequest Parameters

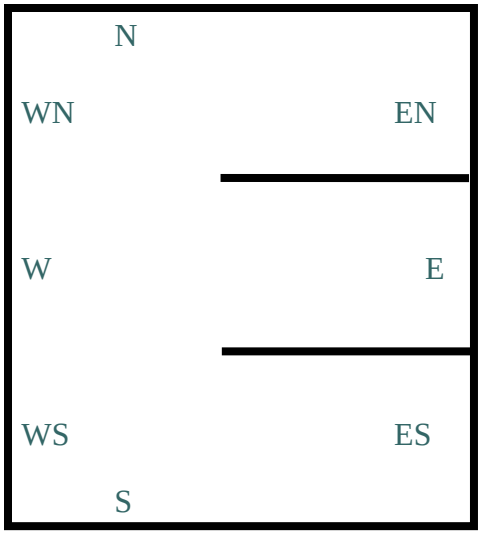
- RequestType
  - cameraMap
  - groundMap
  - localMap
  - worldMap
- Shape parameters:
  - objectColors
  - occluderColors
  - maxDist
  - minBlobArea
  - markerTypes
- Utility functions:
  - clearCam, clearLocal, clearWorld
  - rawY
  - immediateRequest
- Lookout control:
  - motionSettleTime
  - numSamples
  - sampleInterval
  - pursueShapes
  - searchArea
  - doScan, dTheta
  - manualHeadMotion

How to Detect AprilTags

```
$nodeclass AprilTest : VisualRoutinesStateNode {
  $nodeclass Look : MapBuilderNode (MapBuilderRequest::localMap) : constructor {
    mapreq.setAprilTagFamily(); // Use the default tag family
  }
  $nodeclass Report : SpeechNode : doStart {
    NEW_SHAPEVEC(tags, AprilTagData, select_type<AprilTagData> (localShS));
    textstream << "I saw " << tags.size() << " good April tags");
    SHAPEVEC_ITERATE(tags, AprilTagData, t)
      textstream << "  Tag " << int(t->getTagID()) << " is here.";
      textstream << "  Distance " << int(t->getCentroid().coordZ()) << " millimeters.";
    END_ITERATE;
  }
  $setupmachine {
    Look =C=> Report
  }
}
```

C. Locate and Report Color Balls Inside a Maze

- This is the task for the 2010 Robotics Competition held along with the 2nd Annual ARTSI Student Research Conference
- The task is to get a robot to localize itself within a maze, navigate efficiently between locations, observe objects in the maze, and report on what it has observed.



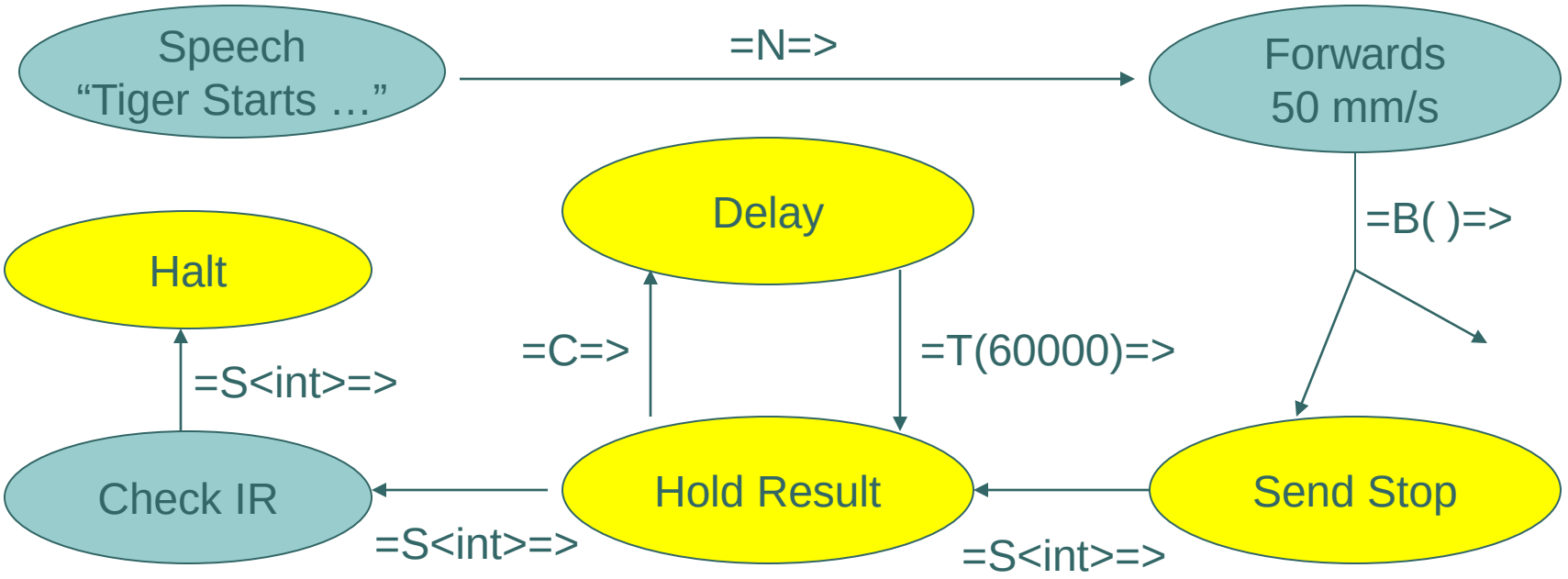
Milestones

- What students need to know first
  - Following wall
  - Recognizing color balls (objects in the maze)
  - Detecting and visiting bi-color navigation marks
  - Memorizing what objects have been seen already
- Milestones
  - Travel though the maze by using “following wall”.
    - Able to start “follow wall behavior” and able to stop
  - Travel though the maze and report detected color balls
    - Report only once per ball, so need memory
  - Travel though the maze and report detected color balls and their locations

Skills and Challenges

- Basic Programming Skills
  - Instantiate a class
  - Derive a subclass
- Tekkotsu Programming Challenges
  - New state machine programming language
  - New semantics of state machine
  - Setup computer camera setting for a specific light condition
- Uncertainty and Failure
  - Bi-color markers are difficult to detect
  - Dealing with false color balls

Follow Wall

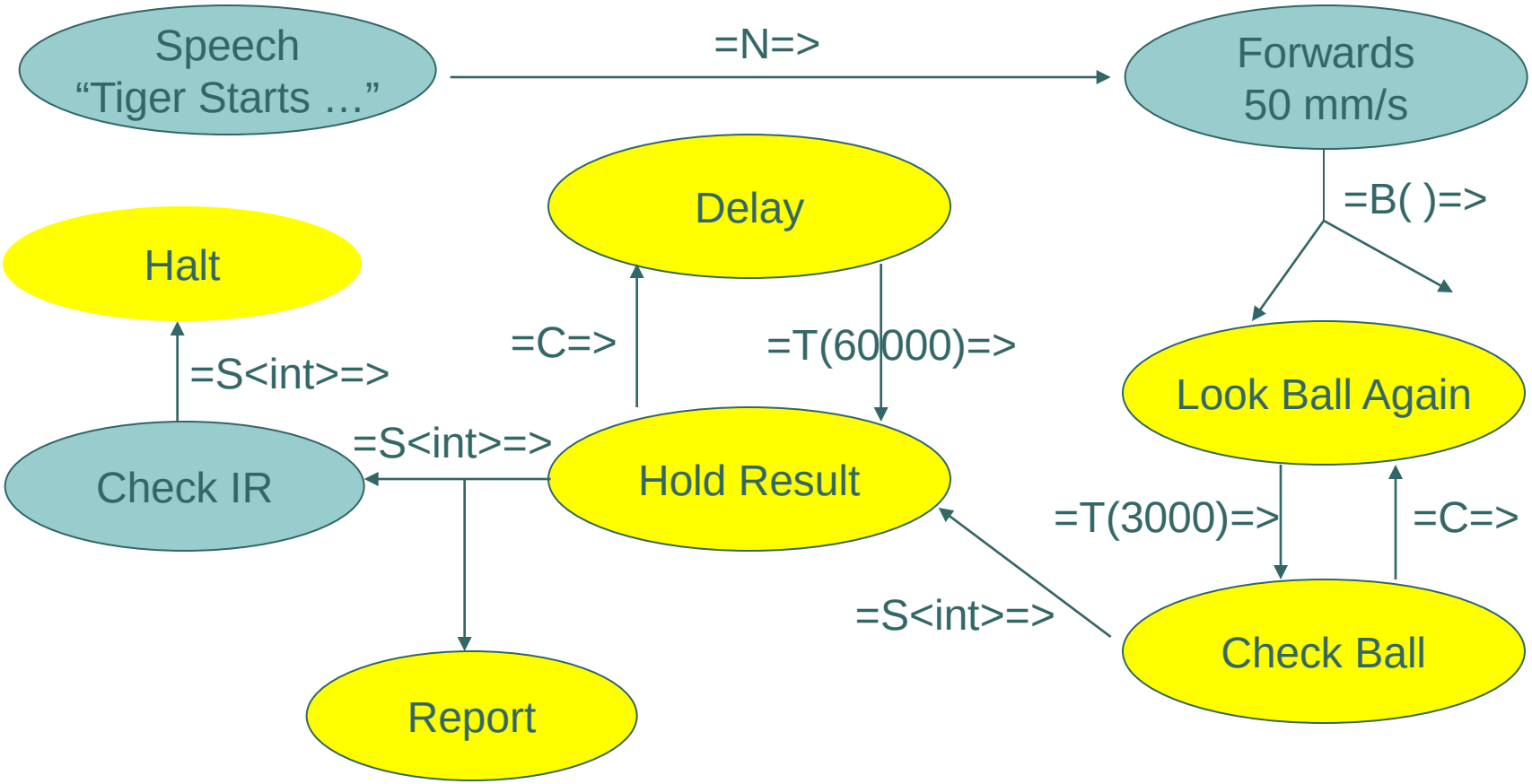


The robot can be placed in anywhere and can be stopped after a given period of time

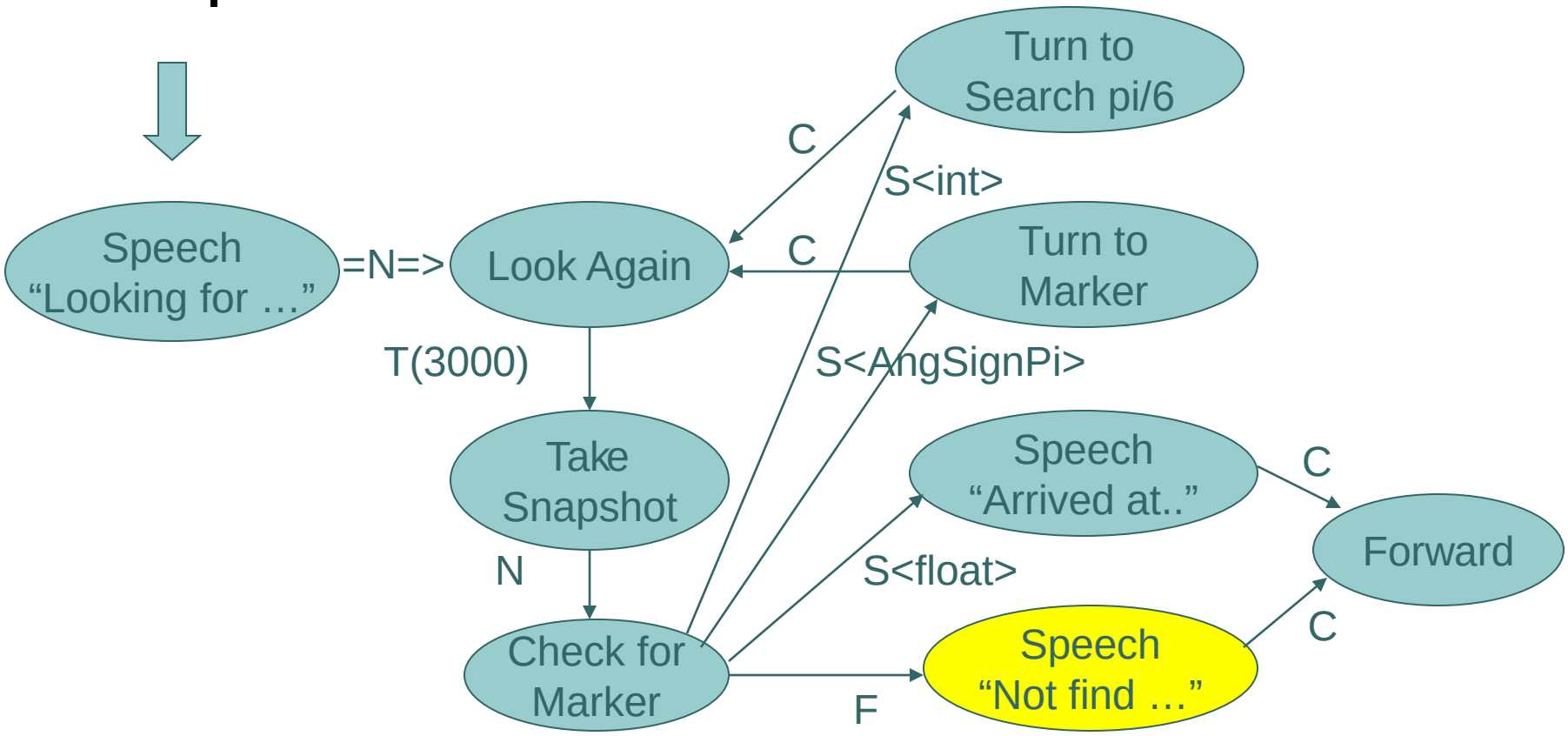
Memorize Color Balls

- Color ball encoding:
  - Orange(Red): 1
  - Blue: 2
  - Green: 4
- The color ball combinations can be encoded as follows
  - {Red, Blue} = 3,
  - {Red, Green} = 5,
  - {Blue, Green} = 6,
  - {Red, Blue, Green} = 7
- The color ball sequences can be encoded as follows
  - (Red, Blue, Green)= 124,
  - (Blue, Green, Red) = 241, and etc.

Find Balls and Report



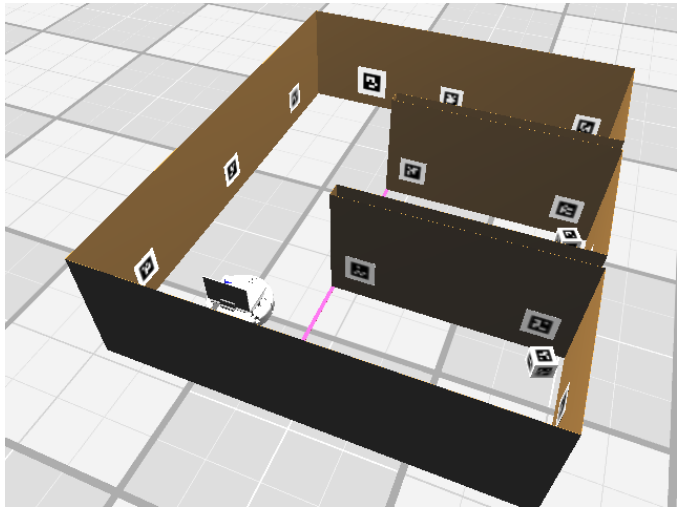
Detect and Visit Marker

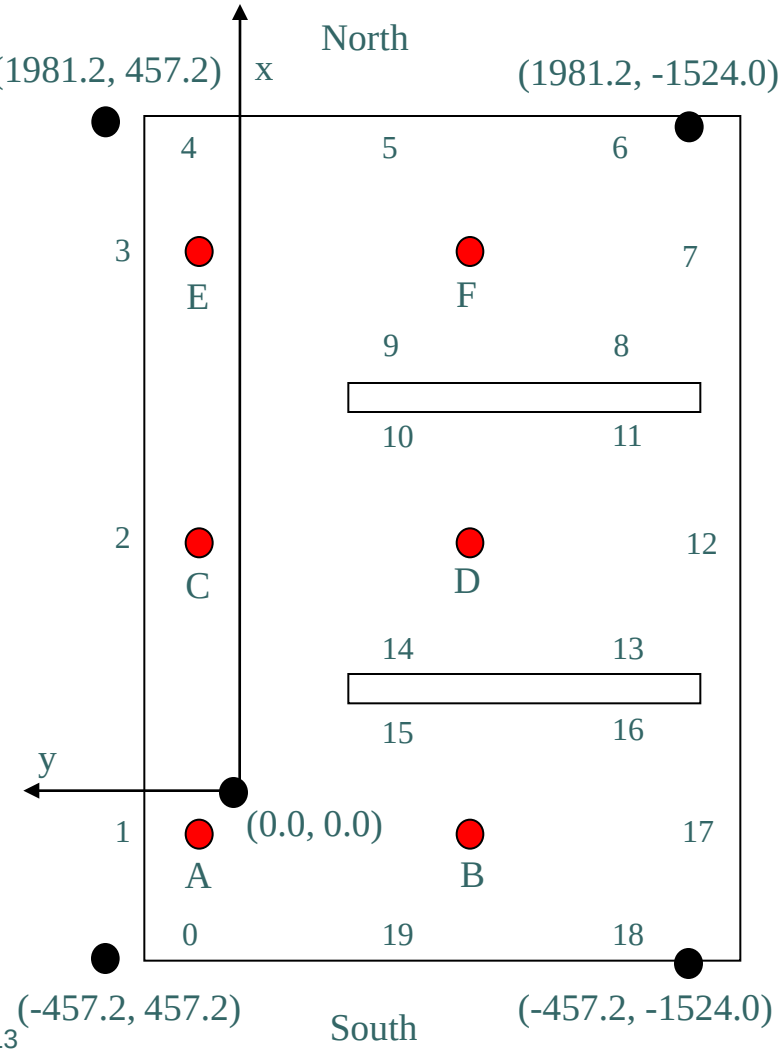


In case of failure in finding any marker, the robot just goes straight to hit the wall

D. Locate and Report AprilTags Inside a Maze

- This is the task for the 2011 Robotics Competition held along with the 3rd Annual ARTSI Student Research Conference





Task and Differences

- This task is the same as the one in P4 except
  - The bicolor makers are replaced by the AprilTags, and
  - The three color balls are replaced by three cubes with each covered by different AprilTags
- The major differences between C and D
  - In C, each bicolor marker is treated as a topological navigation marker and the metric measurements of the maze are not used at all.
  - In D, a metric map of the maze is used in the robot localization and path planning and execution.

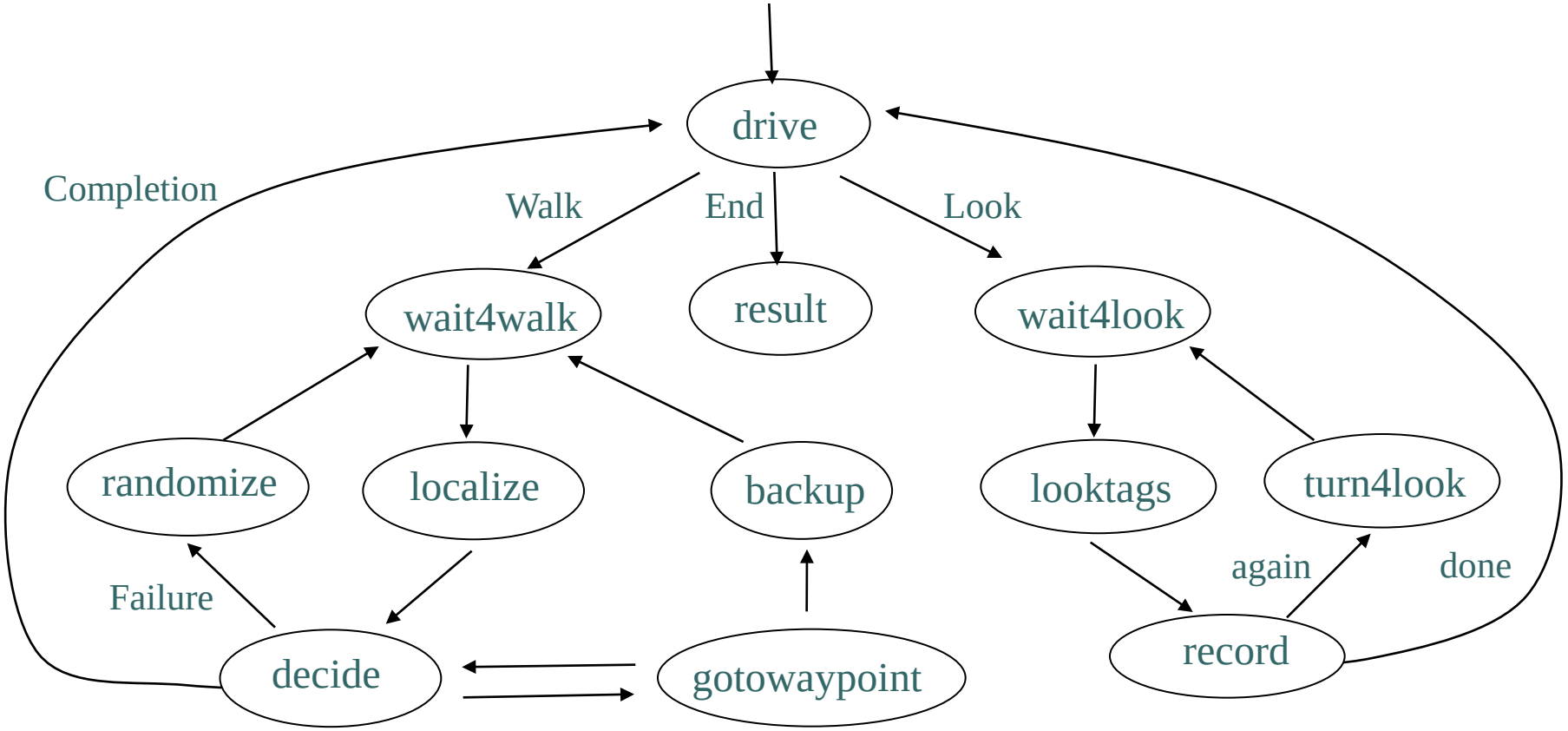
Milestones

- What students need to know first
  - Drive the robot forward or make a turn,
  - Check the robot location inside the maze,
  - Localize the robot within the maze, and
  - Look AprilTags that are facing to the robot.
- Milestones
  - Memorize object the robot has observed in each alcove.
  - Look for the object in each alcove
  - Drive robot to a waypoint in the maze. (goto function)
  - Put it together

Uncertainty and Failure

- 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

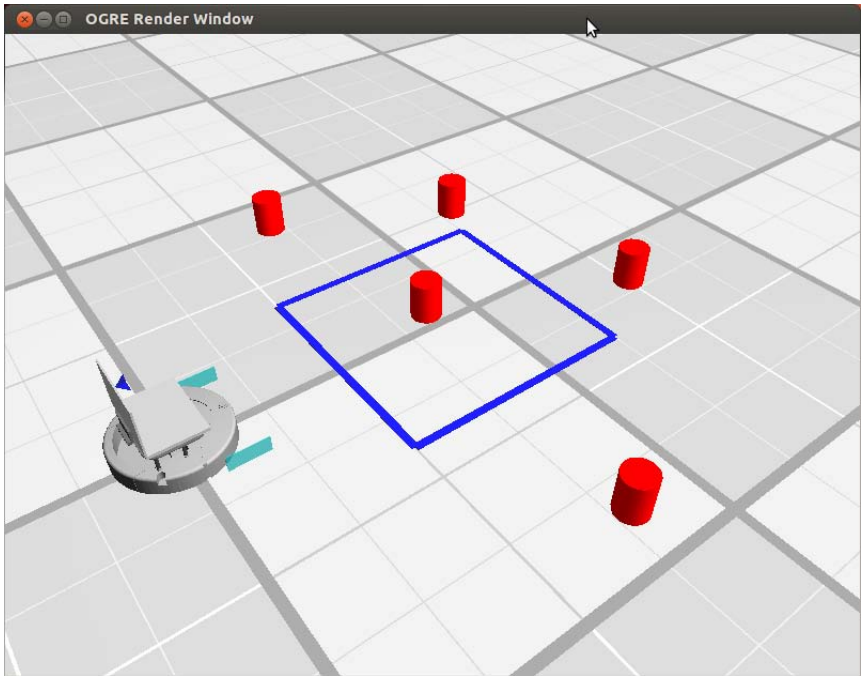
Put It Together



E. Pushing Canisters into Pen

This is the task for the 2012 Robotics Competition held along with the Fourth ARTSI Student Research Conference

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.



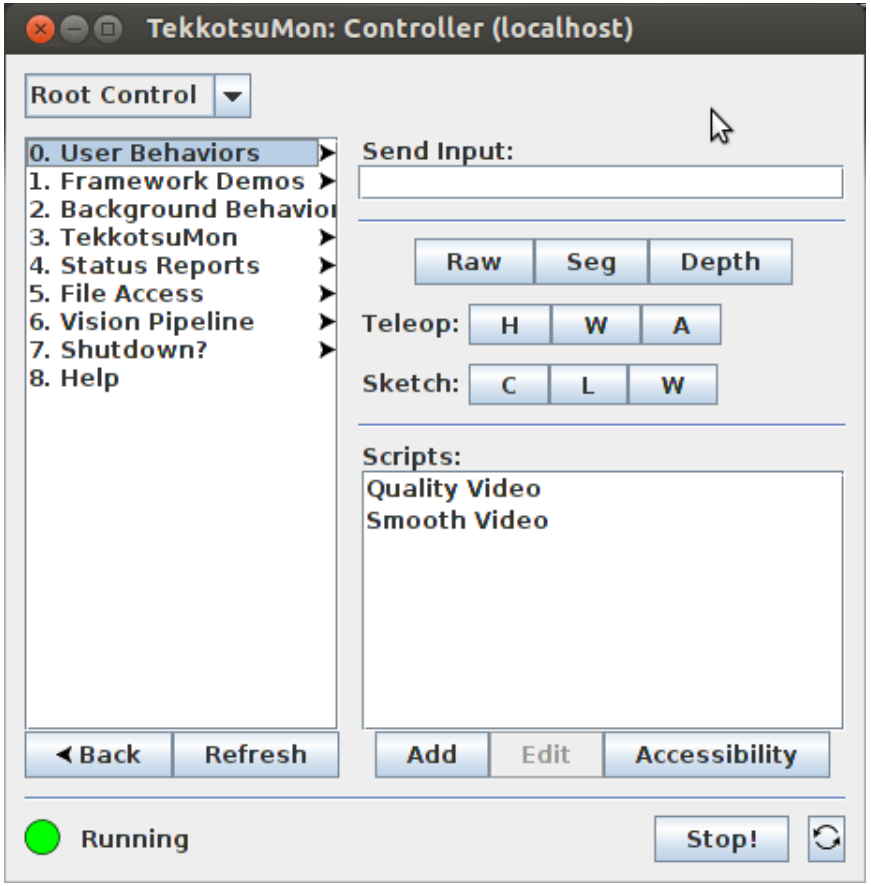
Five red canisters are scattered around a 2-by-2 meter space. Near the center of the space is a 0.75-by-0.75 meter "pen" drawn with blue masking tape.

Steps to Accomplish the Task

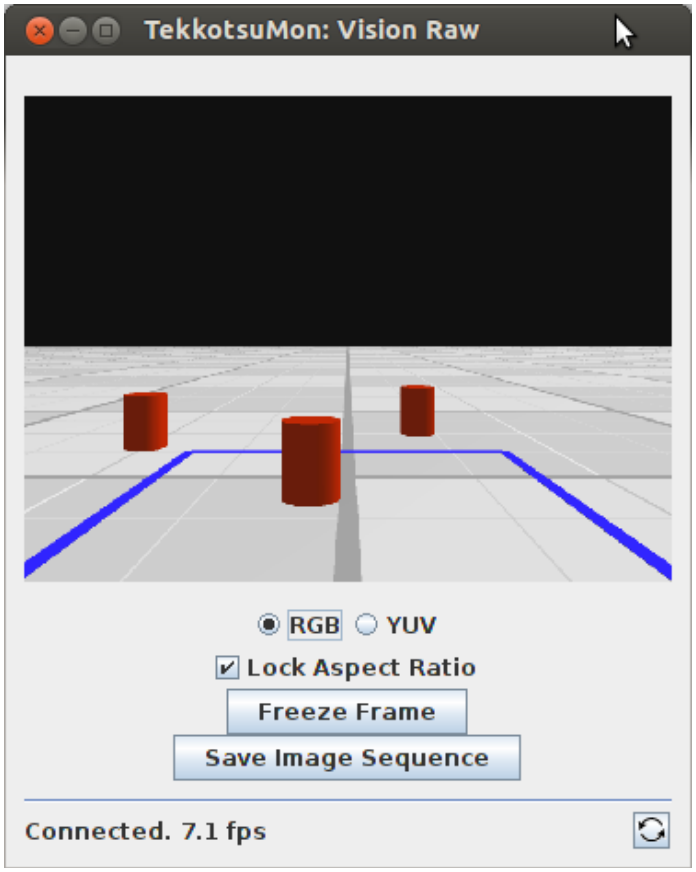
1. Familiar with Tekkotsu GUI tools and Mirage simulator
2. Understand overall strategy and related algorithms (Euclidean geometry)
3. Push one canister into the pen, when
   1) A pen corner and a canister outside the pen are facing the robot simultaneously
   2) A pen corner and a canister outside the pen can be detected by the robot simultaneously
4. Push all canisters into the pen, when
   1) A pen corner and a canister outside the pen can be detected by the robot simultaneously
   2) A pen corner and a canister outside the pen cannot be detected by the robot simultaneously

1. Tekkotsu GUI tools

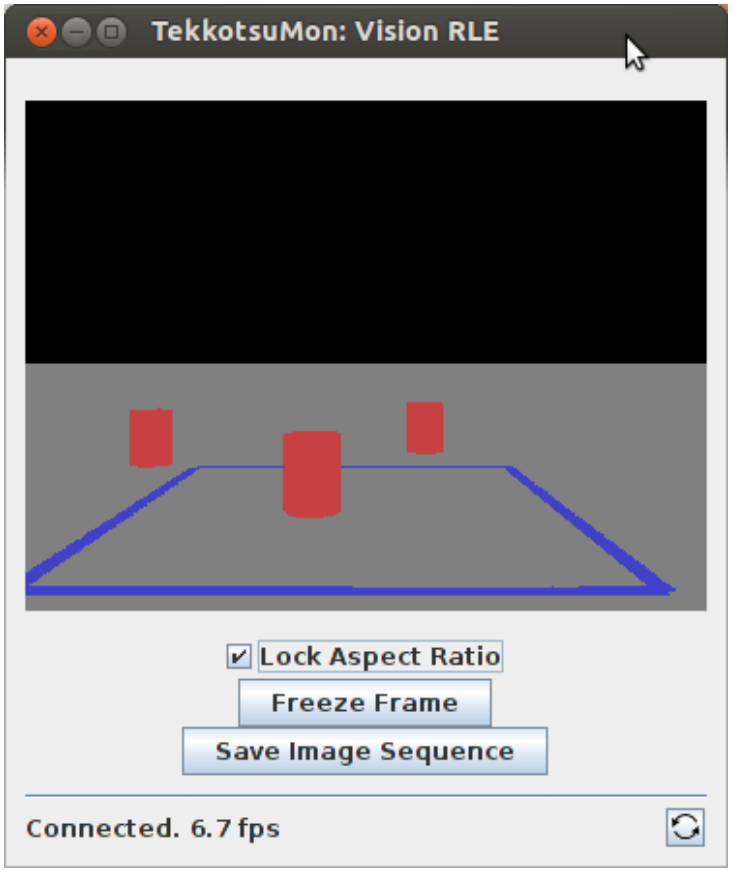
- Raw: Raw image
- Seg: Segmented image
- Depth: Depth image
- H: Head operation
- W: Walk operation
- A: Arm operation
- C: Camera Space
- L: Local Space
- W: World Space



Tekkotsu GUI tools (Cont.)

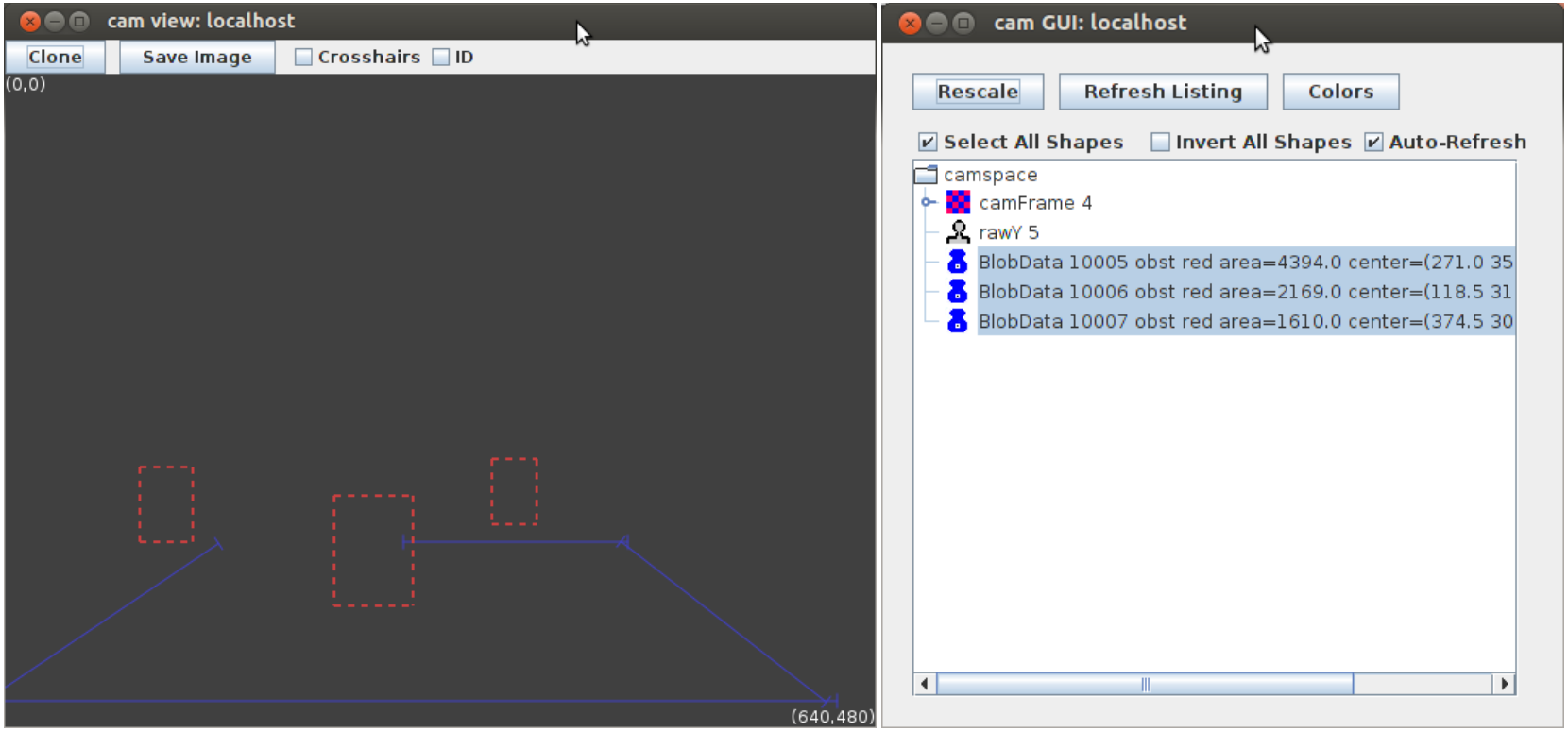


What the robot sees



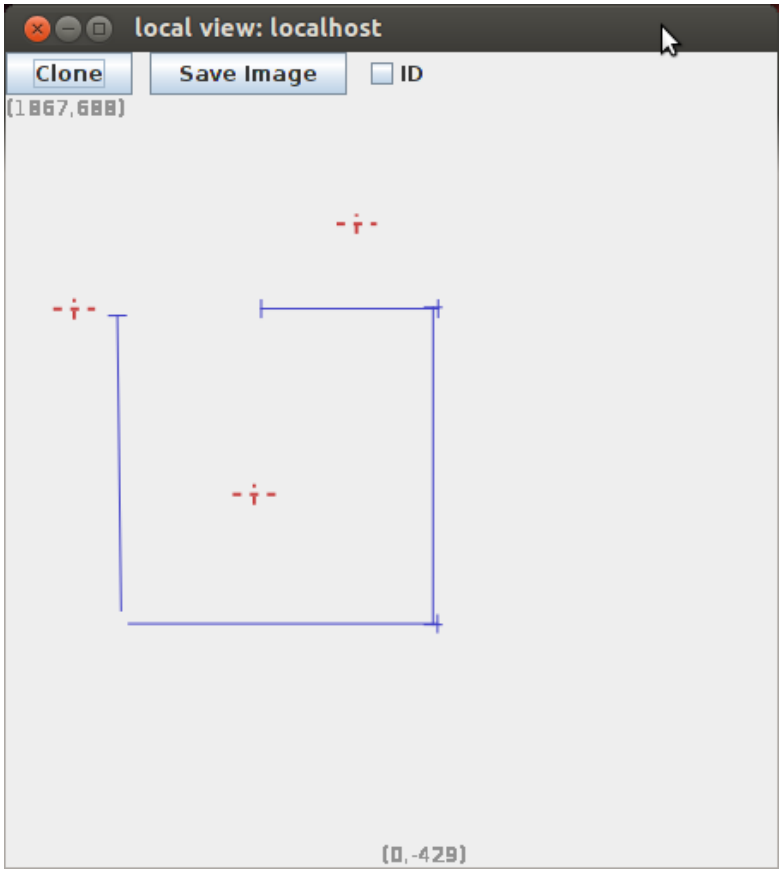
Segmented image

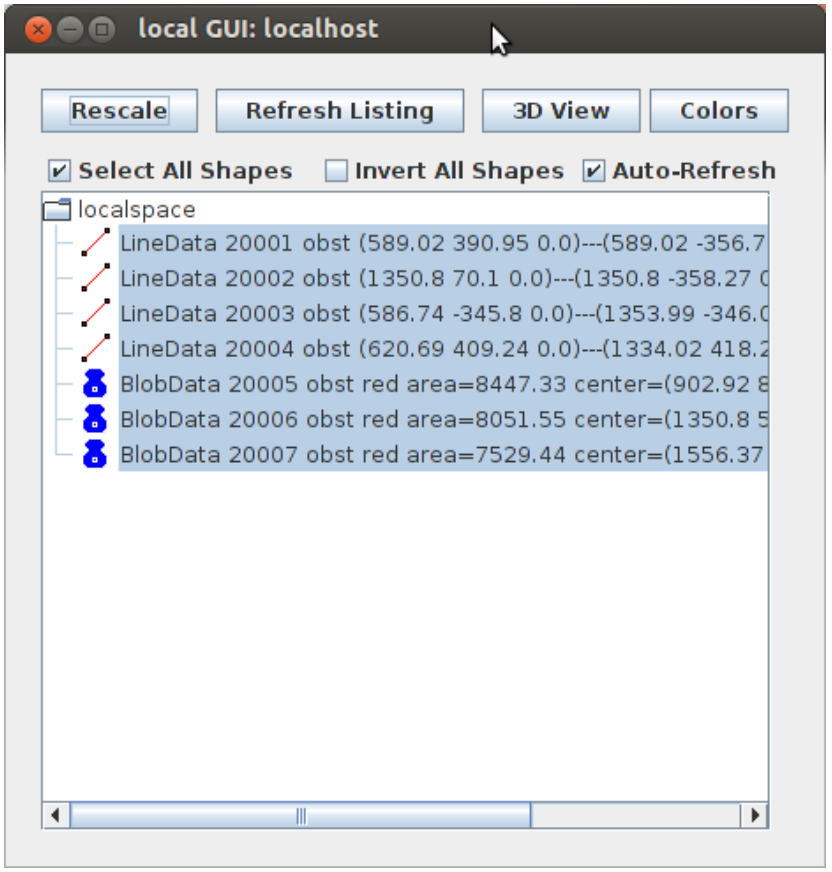
Tekkotsu GUI tools (Cont.)



Three canisters and four lines are detected and shown in camera space

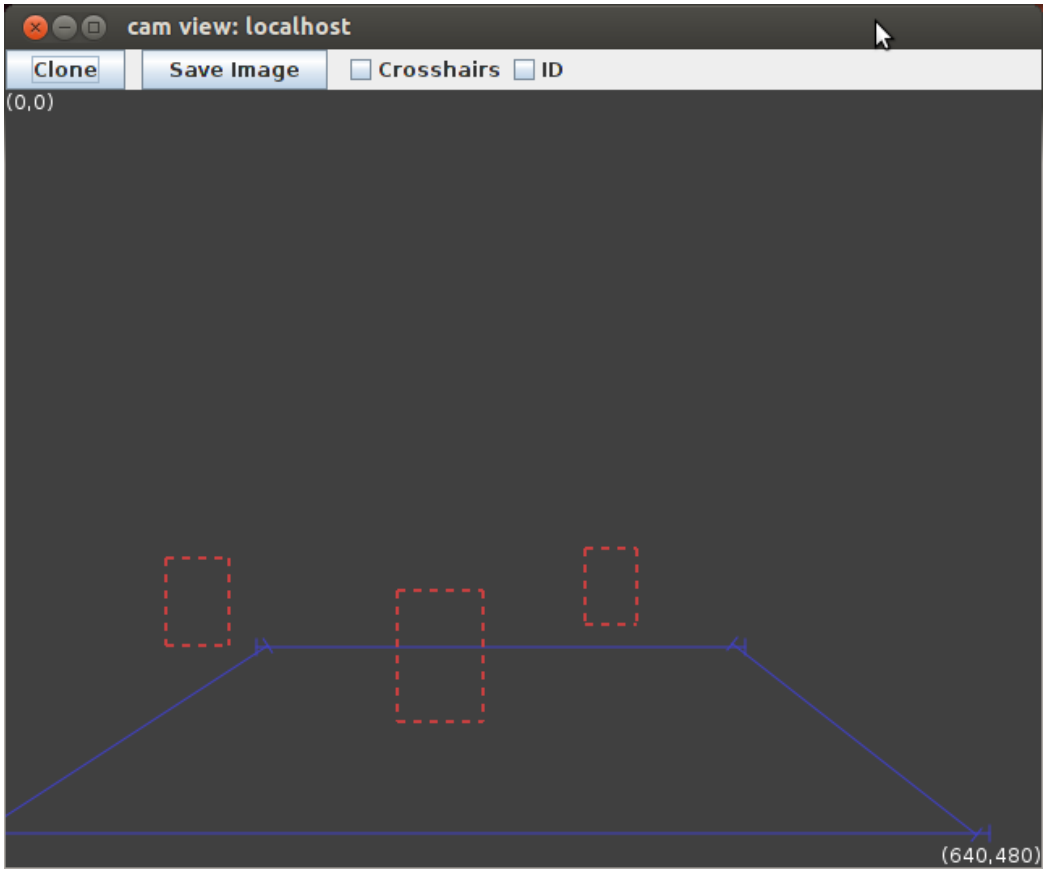
Tekkotsu GUI tools (Cont.)

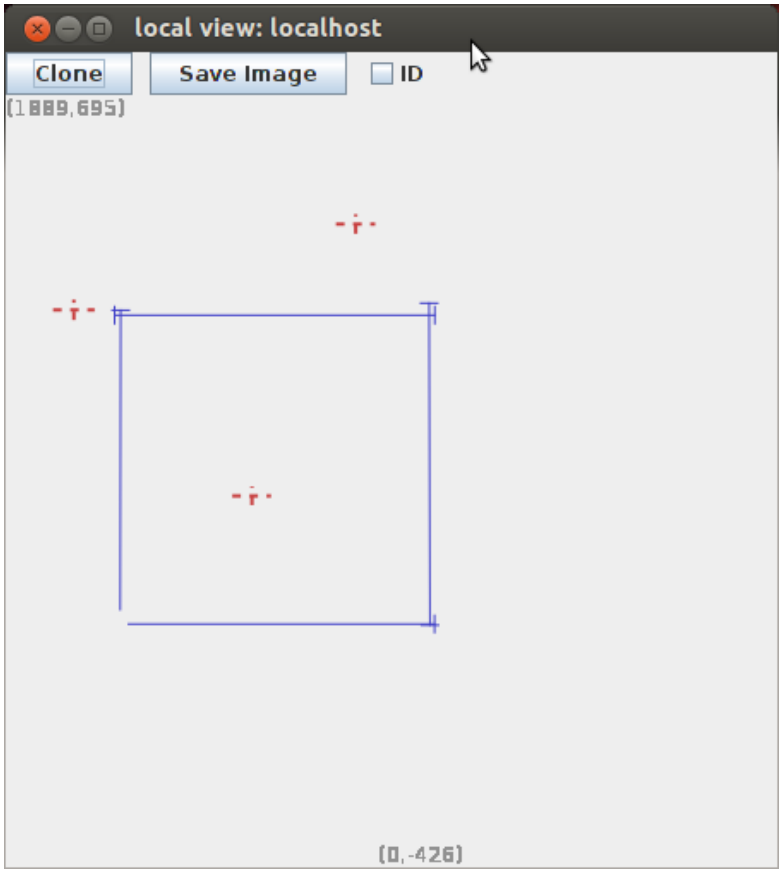




Three canisters and four lines are detected and shown in local space

Tekkotsu GUI tools (Cont.)





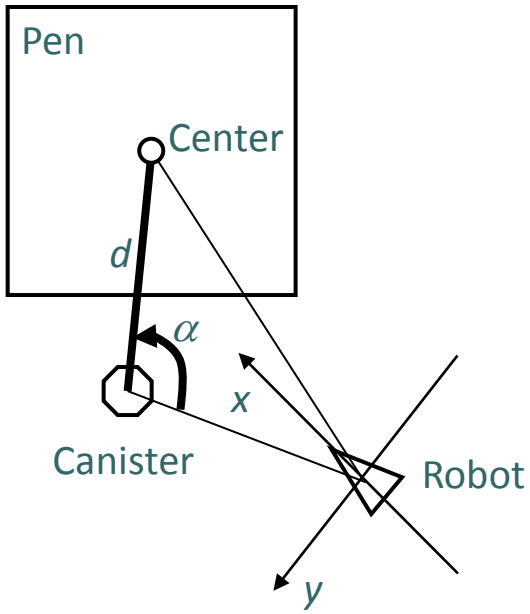
After occlusion is considered

Tekkotsu State Machine Code: Detect blue lines and red pillar blobs

```
$nodeclass Report : VisualRoutinesStateNode {

  $nodeclass BuildMap : MapBuilderNode(MapBuilderRequest::localMap) : doStart {
      mapreq.addObjectColor(lineDataType,"blue");
      mapreq.addOccluderColor(lineDataType,"red");
      mapreq.addObjectColor(blobDataType,"red");
      mapreq.addBlobOrientation("red", BlobData::pillar);
  }

  $nodeclass MapInfo : VisualRoutinesStateNode : doStart {
      cout << "# of lines: " << localShS.allShapes(lineDataType).size() << endl;
      cout << "# of cans = " << localShS.allShapes(blobDataType).size() << endl;
  }

  $setupmachine {
       BuildMap =C=> MapInfo
  }
}
```

2. Overall Strategy and Algorithms

Overall Strategy



1. Get the pen center and a canister outside the pen in the local space
2. Move to the canister and then push it into the pen center.
3. Backup and get out of the pen.
4. Go back to 1 until the job done.

Pen Center Calculation

- Detect a pen corner
- Compute the two pen corners that are adjacent to the detected corner
- Compute the midpoint.

Some Geometric Formulas

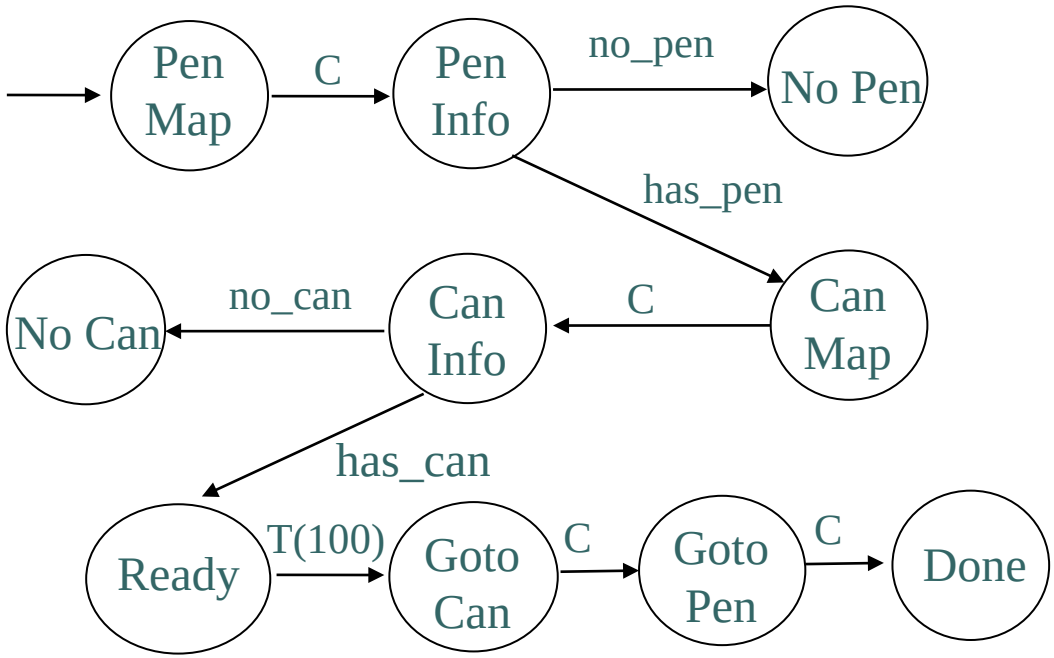
```
Point RotationTransform (Point& c, float angle) {
    float x = c.coordX()*cos(angle) + c.coordY()*sin(angle);
    float y = -c.coordX()*sin(angle) + c.coordY()*cos(angle);
    return Point(x, y);
}
```

```
turn2Canister = canister.atanYX();
fwd2Canister = canister.xyNorm();
```

```
int ceny = center.coordY();
int cenx = center.coordX();
int cany = canister.coordY();
int canx = canister.coordX();
turn2Center = atan2(ceny-cany, cenx-canx)-turn2Canister;
fwd2Center = canister.xyDistanceFrom(center);
```

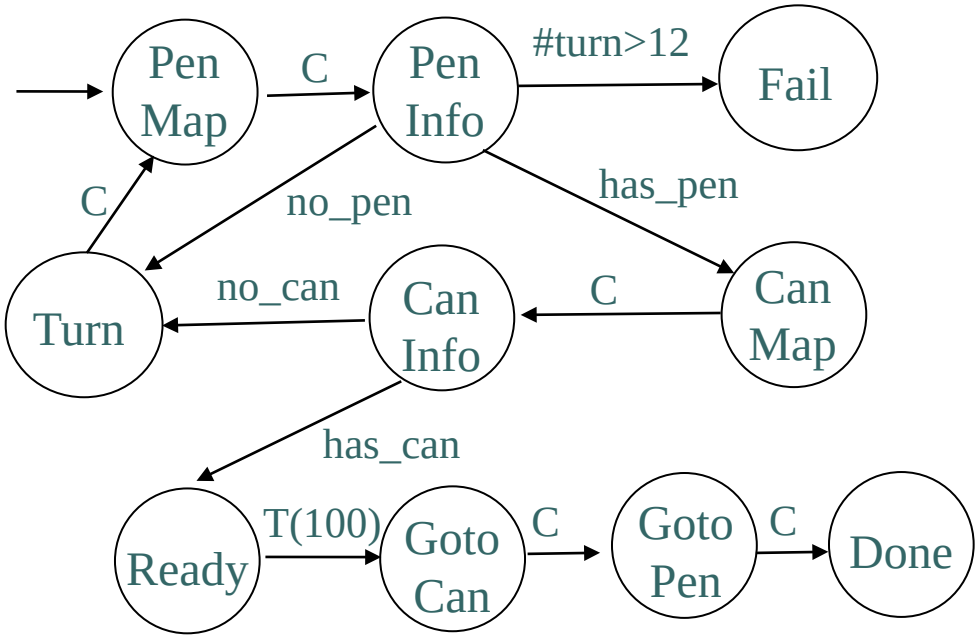
3. Push One Canister into the Pen

Push the canister into the pen, if a canister outside the pen and a pen corner are facing the robot simultaneously. Otherwise, say “no pen” or “no can”.



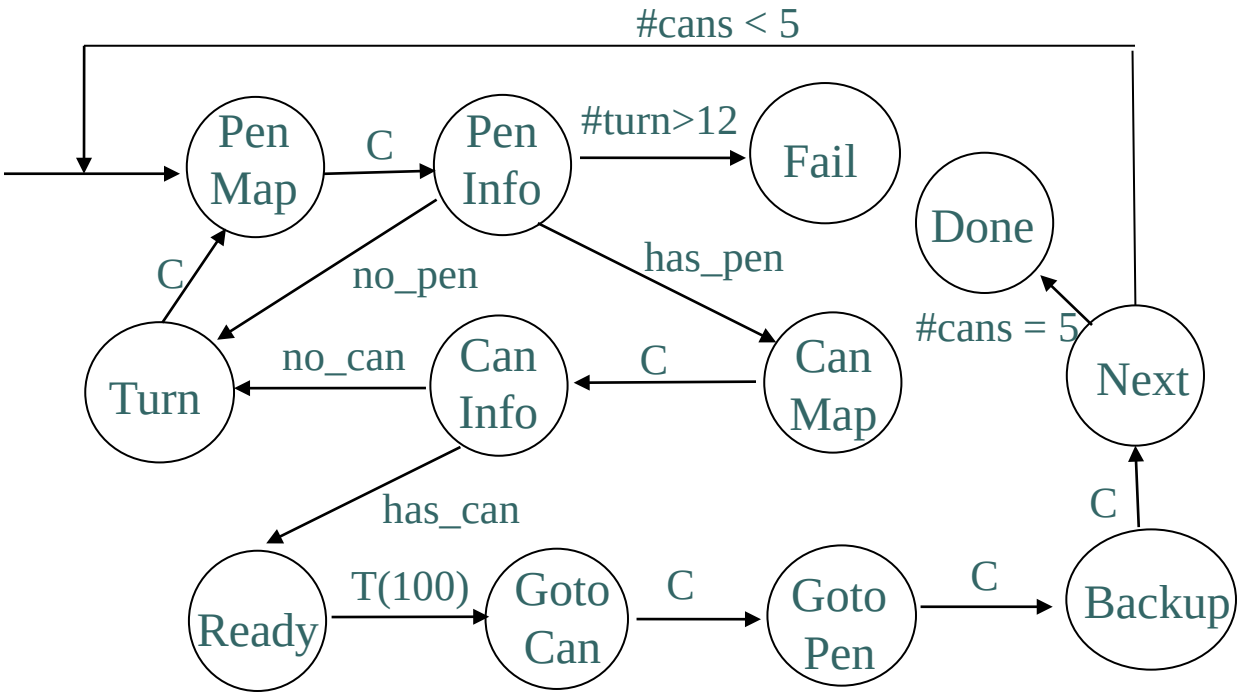
Push One Canister (Cont.)

Push the canister into the pen, if a canister outside the pen and a pen corner can be detected by the robot simultaneously. Otherwise, say “fail”.



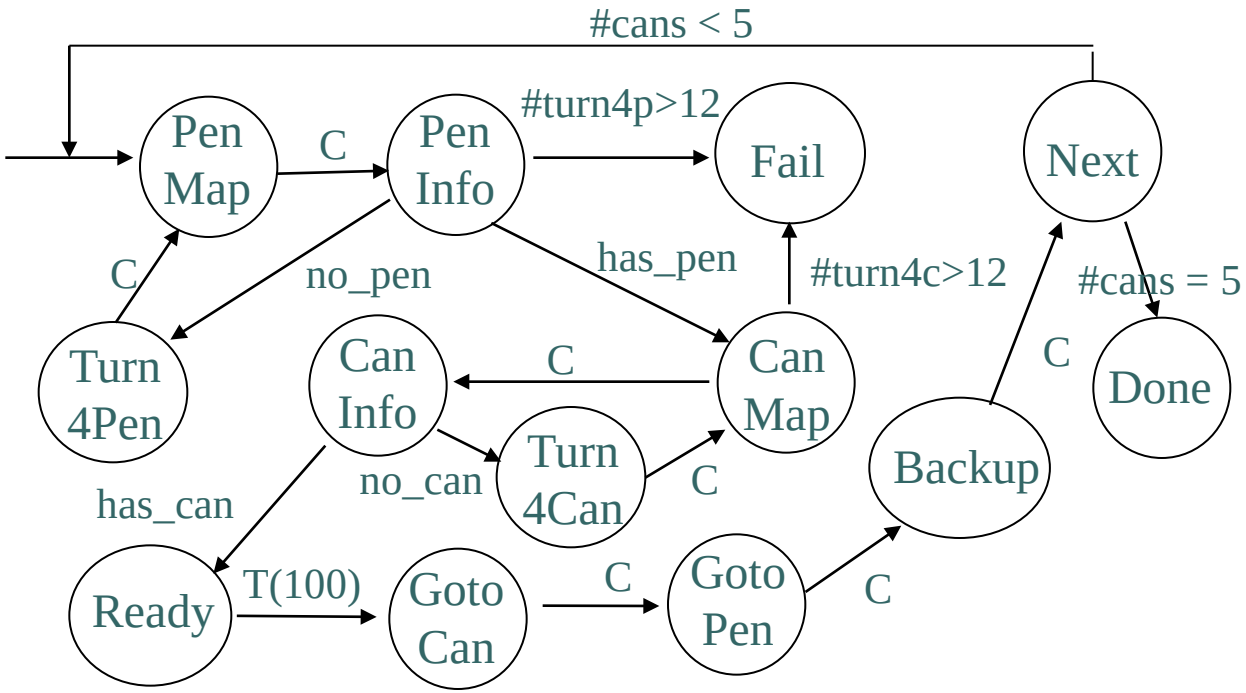
4. Push All Canisters into the Pen

Push all canister into the pen, if any canister outside the pen and a pen corner can be detected by the robot simultaneously. Otherwise, say "fail".



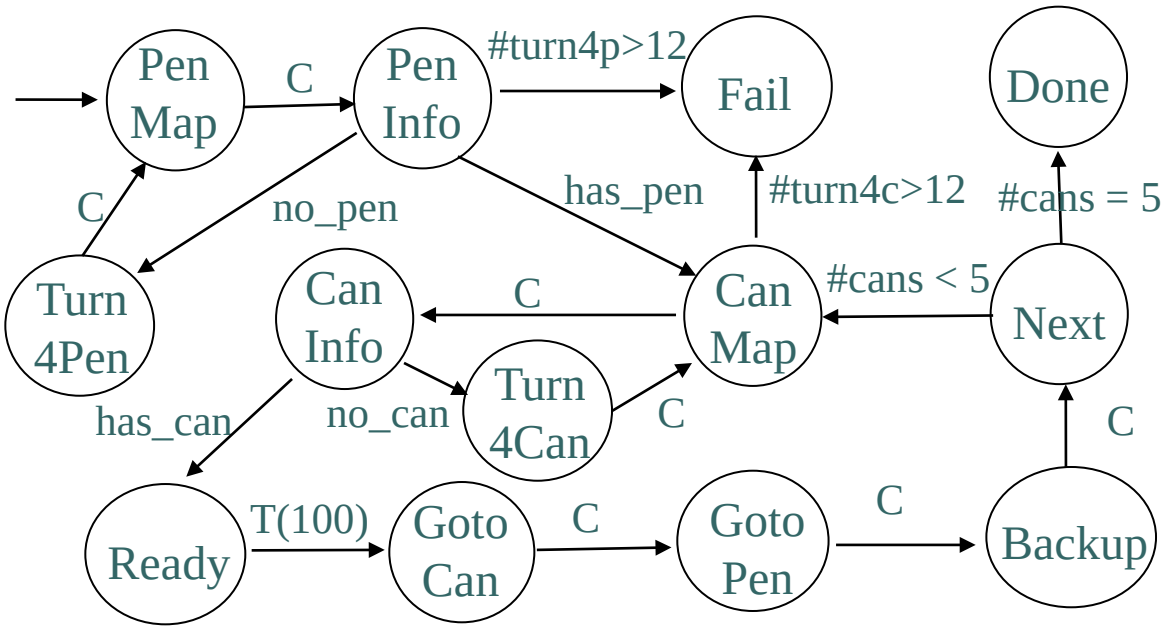
Push All Canisters (Cont.)

Push all canister into the pen, in case that the robot may not be able to detect a pen corner and a canister outside the pen simultaneously.



New Strategy

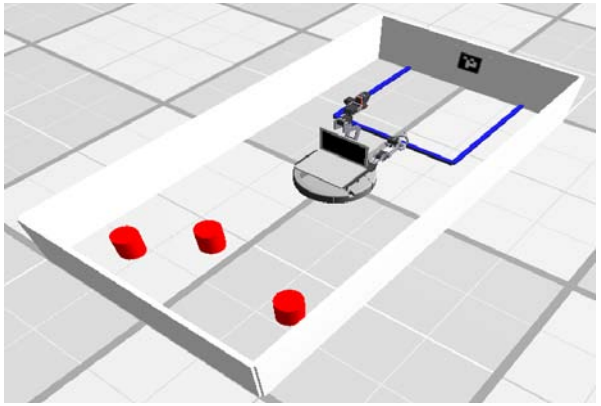
1. Get the pen center
2. Get a canister outside the pen
3. Push the canister into the pen.
4. Backup and get out of the pen.
5. Go back to 2 until the job done.



F. Pickup Canisters

This is the task for the 2013 Robotics Competition held along with the Fourth ARTSI Student Research Conference

The task is to use a Calliope2SP robot to locate each cylinder, pick it up using its gripper, and transport it to the goal location as quickly as possible .

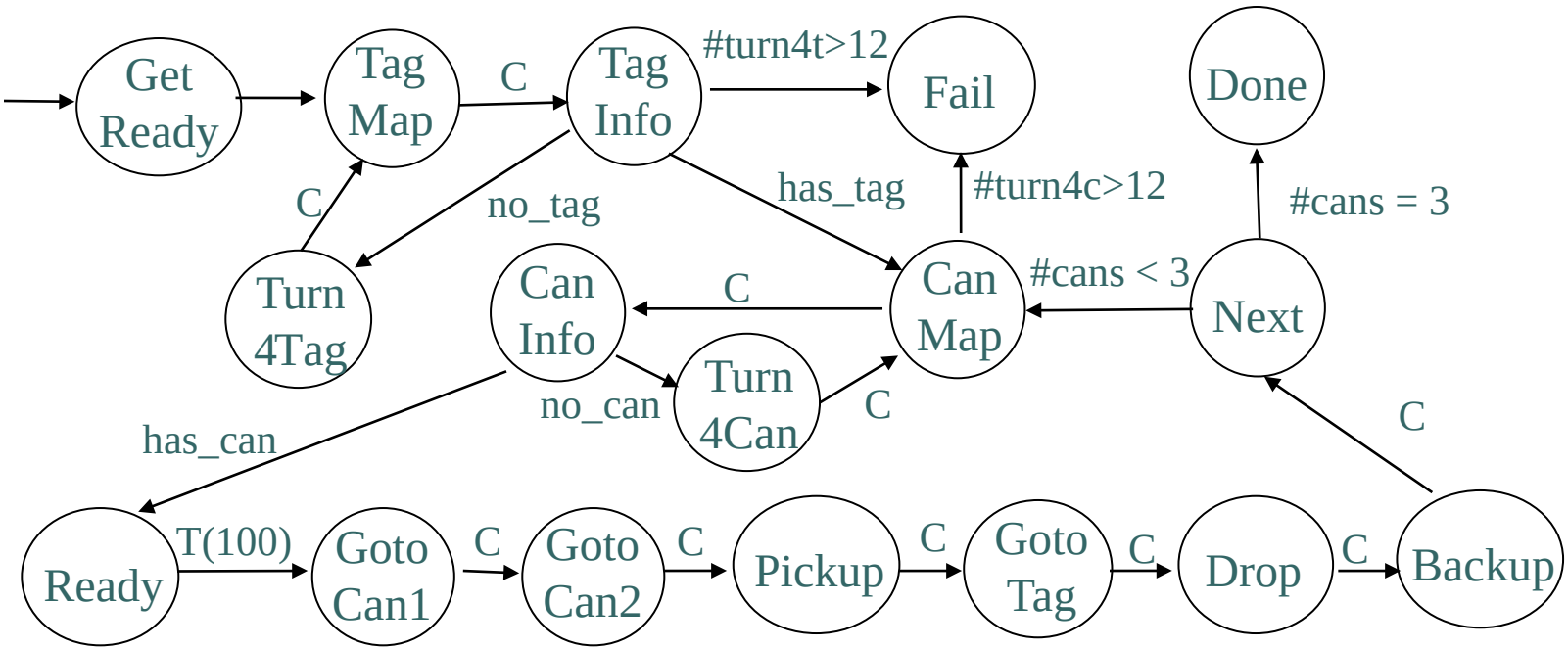


Three colored cylinders are scattered around the 1 by 2 meters arena, far from the goal location defined as by a 2x2 foot box below the AprilTag.

Steps to Accomplish the Task

1. Overall Strategy
2. Locate an AprilTag and move to it
3. Pickup a canister when it is placed in between the two fingers of the gripper
4. Locate a canister, move to it, and pick it up
5. Putting all together: Locate a canister, move to it, pickup it, transport it to the goal location, and drop it

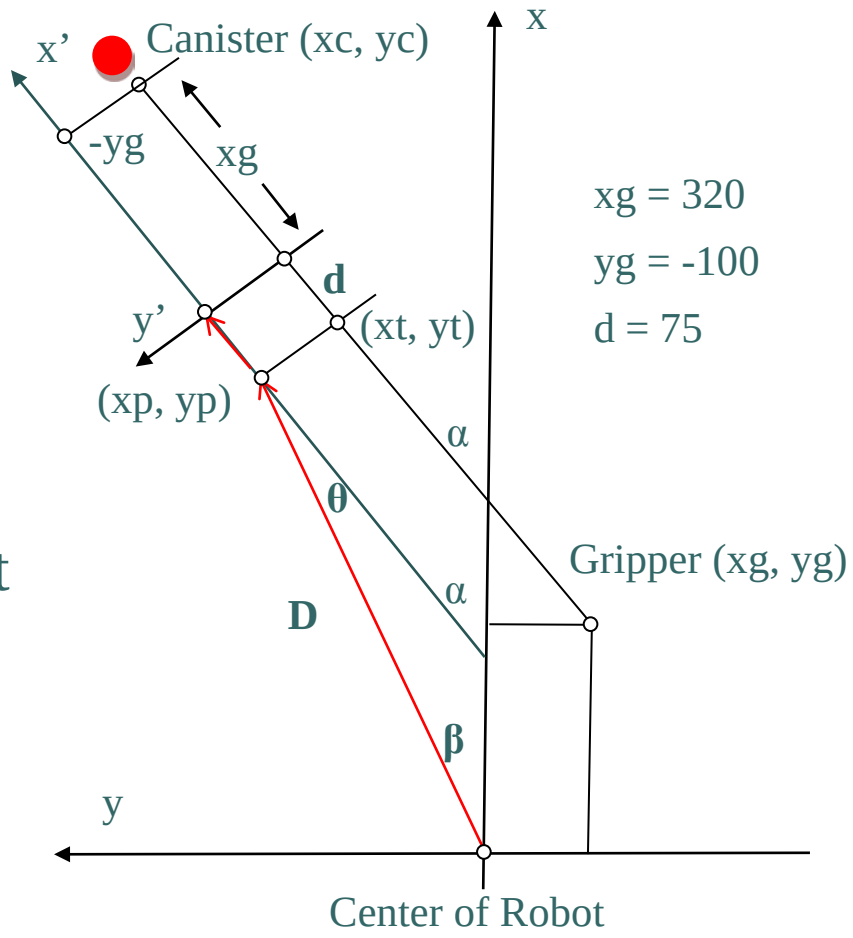
1. Overall Strategy



Path from Robot to a Canister

❖The robot should follow the route with red color to reach to a canister. To this end, students need to calculate D, β, and θ.

❖Note that (xg, yg) can be obtained by measuring the Calliope2SP robot, d is a constant which should be a little longer than the gripper's finger, and (xc, yc) can be obtained via Tekkotsu.

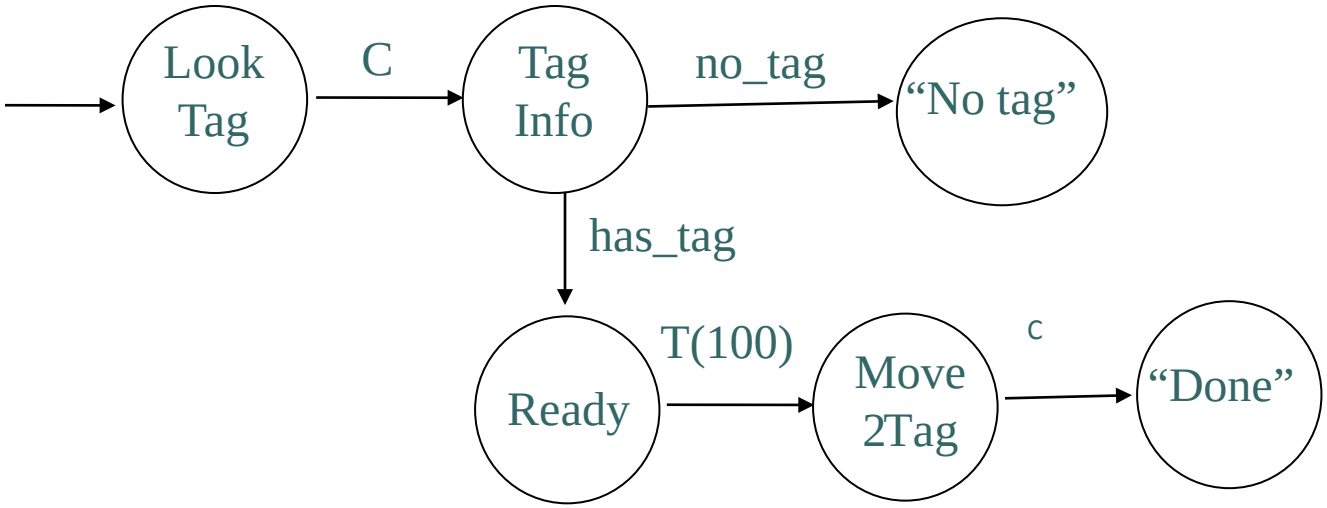


Some Geometric Formulas

```
slope = canister - gripper;
α = slope.atanYX();

xc = canister.coordX();
yc = canister.coordY();

xg = gripper.coordX();
yg = gripper.coordY();
```

```
xt = xc - (xg+d)*cos(α);
yt = yc - (xg+d)*sin α);

xp = xt + yg*sin(α);
yp = yt - yg*cos(α);

Point pp(xp, yp);
β = pp.atanYX();
D = pp.xyNorm();
θ = α - β;
```

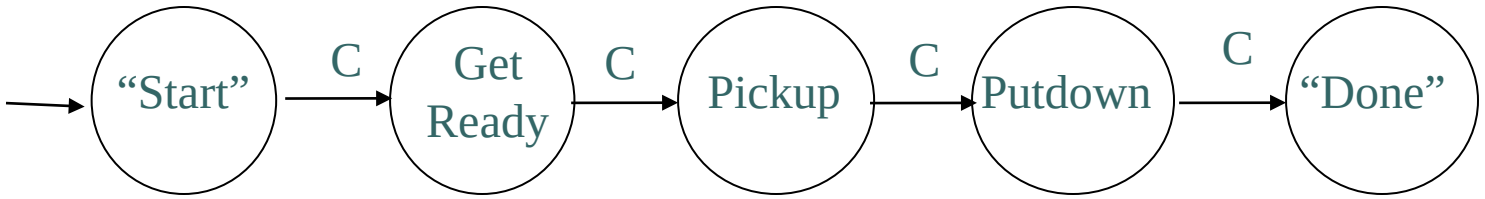
2. Locate an AprilTag and Move Close to It



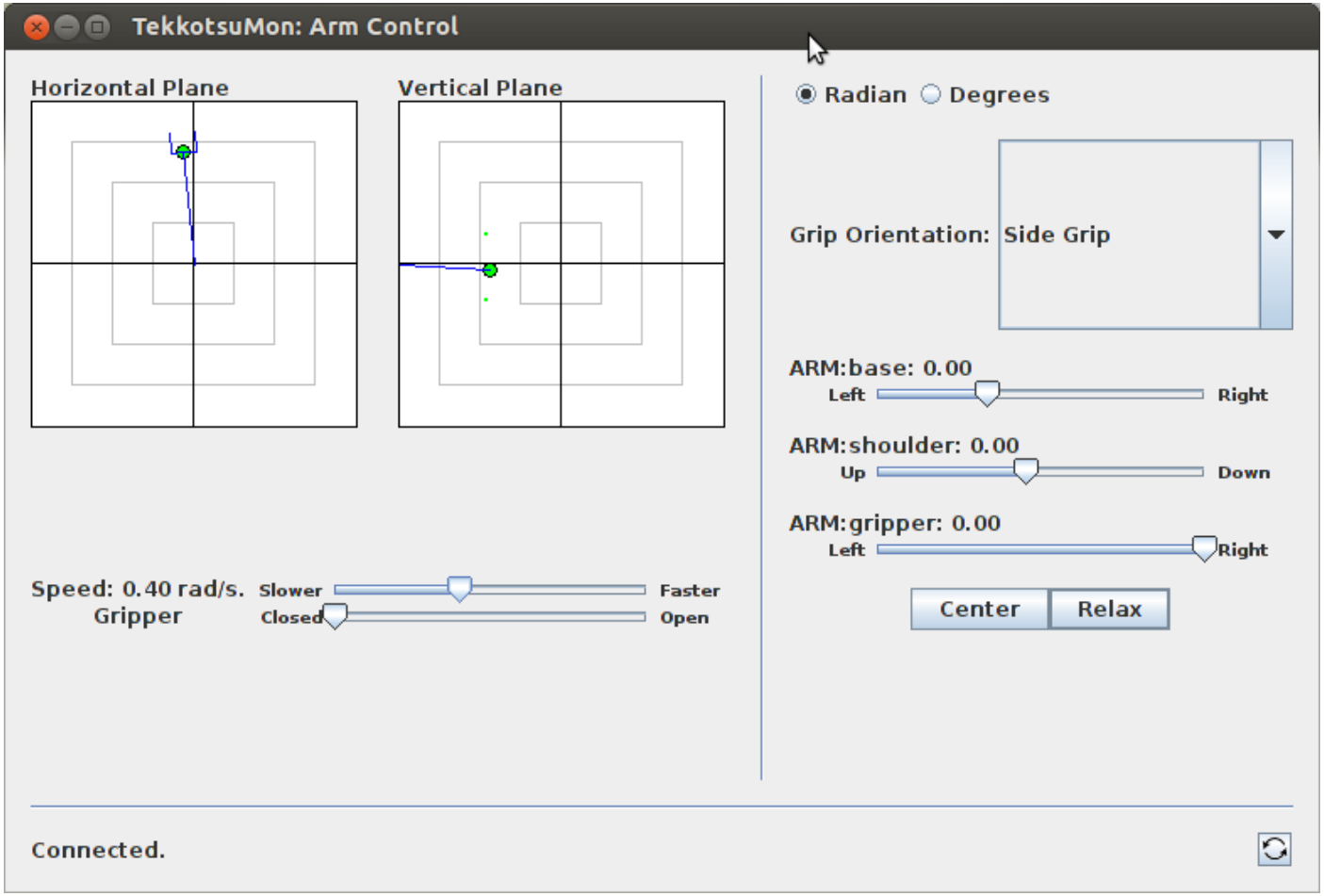
A state machine code that implements the above state transition diagram is given. Students are asked to test this code and then modify the code so that the robot can search for an AprilTag, if no tag is visible to the robot.

3. Pickup a Canister

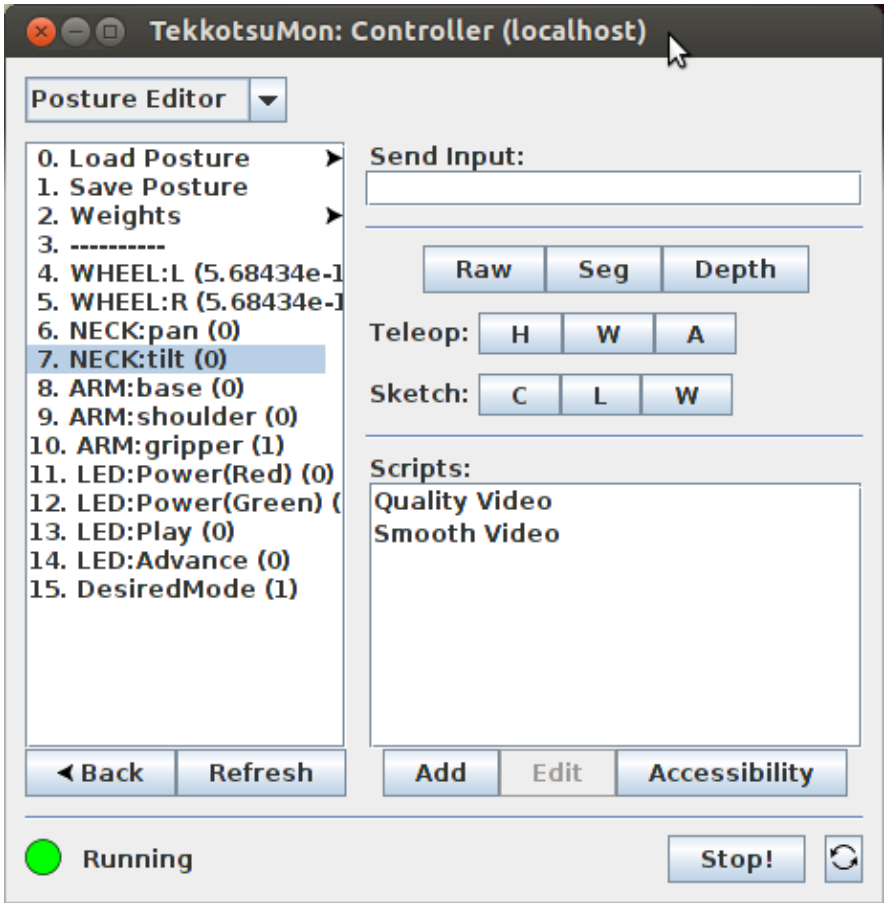
- A state machine code is given to students. This code will make the robot in a getting ready posture by using the Tekkotsu's Dynamic Motion Sequence Node.
- Students are asked to study the code and then add two more motion sequences: Pickup and Putdown so that if a canister is placed in between the gripper's two fingers, the robot should be able to pick the canister up.



Tekkotsu GUI tools (Arm & Gripper)



Tekkotsu GUI tools (Posture Editor)



Posture File

```
#POS
NECK:pan          0.0000
NECK:tilt         0.0000
ARM:base          0
ARM:shoulder      -0.8
ARM:gripper       1.1
#END
```

Motion Sequence File

```
#MSQ
advanceTime 2000
load downopen.pos
……
#END
```

Head, Arm, Gripper Control

Head Control			
HeadOffset+0: pan	Right (-1.0)	Center (0)	Left (1.0)
HeadOffset+1: tilt	Down (-1.0)	Center (0)	Up (1.0)

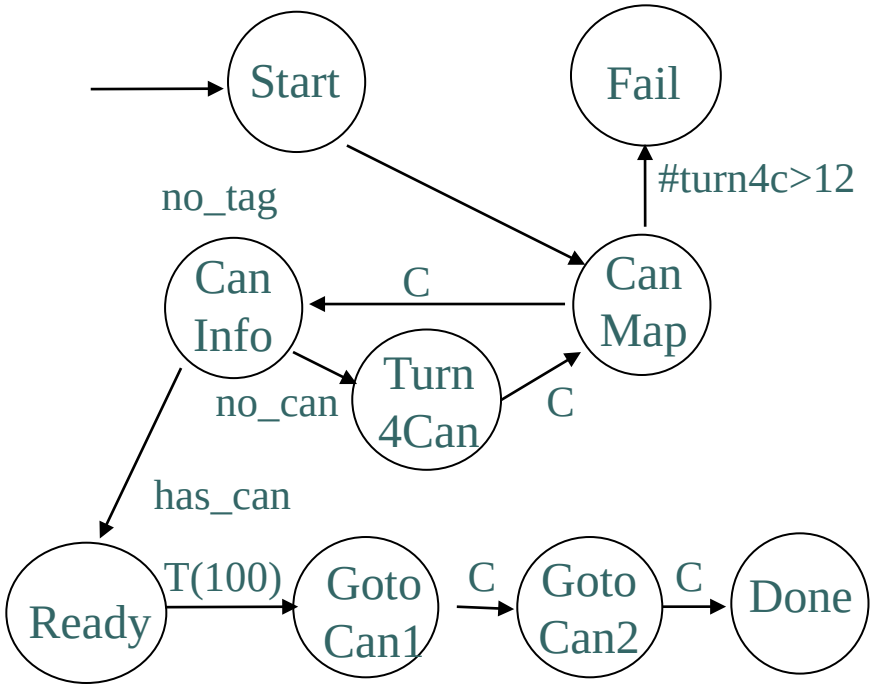
Arm and Gripper Control			
ArmOffset+0: Base	Right (-2,5)	Center (0)	Left (1.0)
ArmOffset+1: Shoulder	Down (-1.0)	Center (0)	Up (1.0)
ArmOffset+2: Gripper	Close (0)		Open (2.5)

```
//Example: Arm Base: Center, Shoulder: Down, and Gripper: Open
getMC()->setOutputCmd(ArmOffset+0, 0);
getMC()->setOutputCmd(ArmOffset+1, -1.0);
getMC()->setOutputCmd(ArmOffset+2, 1.2);
getMC()->advanceTime(2000);
```

4. Locate a Canister, Move to It and Pick It up

A sample state machine code that will detect canisters and drive the robot to the closest canister is given.

Students need to figure out the formulas to calculate D, β, and θ to move to canister



Students need to combine this code with pickup canister so that the robot can locate a canister, move to it, and pick it up.

5. Putting all Together

A: Write a state machine code so that the robot will locate a canister, move to it, pick it up, transport it to the goal location, and drop it.

B: Extend the above code so that the robot will continue to its work until it finishes to pickup all three canisters. Note, please make sure the robot will not pick up a canister again that has been already transported to the goal location.

C: Please modify your code in B so that the robot will be able to memorize the AprileTag location.

G. Discussion and Conclusion

- Our robotics team at JSU has participated in the robotics competitions 4 times held along with the ARTSI Annual Student Research Conferences four times since 2010
- The robotics competitions have provided an opportunity to encourage undergraduate students from non-traditional backgrounds in the study of robotics.
- They have also provided rich materials for robotics education. In fact, three of the four competition tasks have already been transformed into our major robot programming lab projects.

Pros and Cons of Tekkotsu

- Pros
  - Builds a set of high-level interacting software components.
  - Thus, relieves a programmer of the burden of specifying low-level robot behaviors, and hence makes it possible to accomplish challenging robotic applications.
- Cons
  - Some robot vision results are heavily affected by the lighting conditions in the environment.
  - There are a lot false negatives for bicolor makers and a lot false positives for color balls and cylinders.

Thank You

Questions?