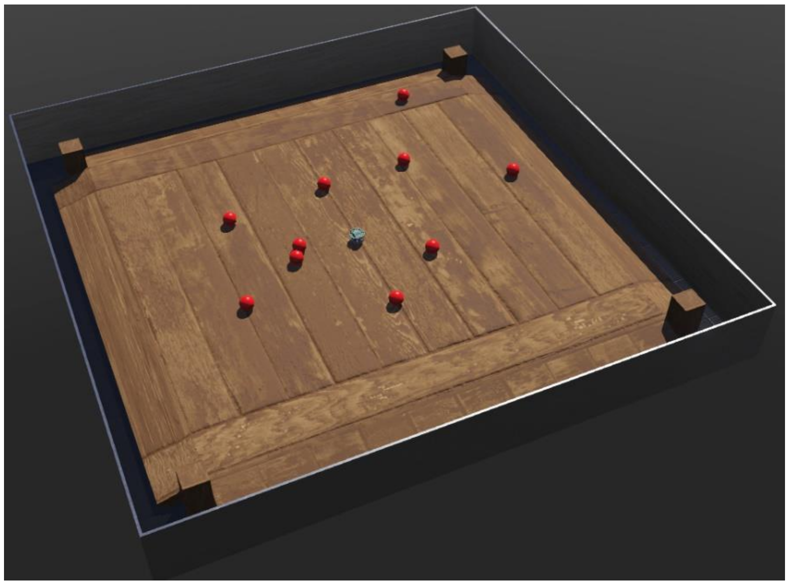
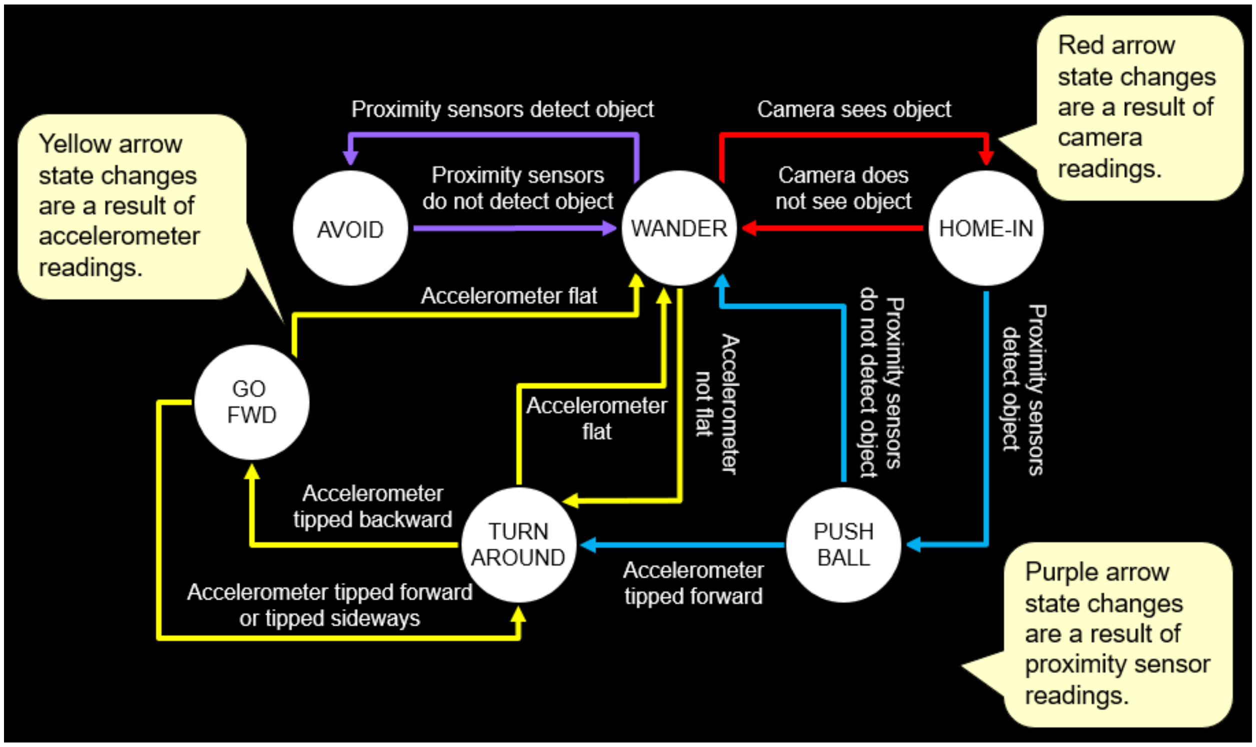
LAB 3 – Homing and Tracking

(1) Download the **Lab3_HomingAndTracking.zip** file and unzip it. Load up the **BallsOnCrate** world. The goal of this lab is to make the robot search out the red balls and push them off the side of the crate, without the robot falling off the edge. In general, your robot should be able to push them all off after only about 5 minutes of running.



(2) The **Lab3Controller** code has been started for you. You should set up a state machine with **6** states as follows:



Yellow arrow state changes are a result of accelerometer readings.
Proximity sensors detect object
Camera sees object
Red arrow state changes are a result of camera readings.
Proximity sensors do not detect object
Camera does not see object
AVOID
WANDER
HOME-IN
Accelerometer flat
GO FWD
Accelerometer flat
Accelerometer not flat
Proximity sensors do not detect object
Proximity sensors detect object
Accelerometer tipped backward
TURN AROUND
PUSH BALL
Accelerometer tipped forward
Accelerometer tipped forward or tipped sideways
Purple arrow state changes are a result of proximity sensor readings.

The robot being used is the e-puck robot. You will make use of the two front proximity sensors, the camera, the accelerometer and the motors.

From experience, I have noticed that many students struggle with knowing where to start with this state machine. Therefore, I will provide steps that will help.

(a) First, let's ensure that our robot does not fall off the edge of the environment. Currently, the code for the **WANDER** behavior has been implemented. It causes the robot to move forward and occasionally veer off path by curving left or right. It is set up so that 1 out of 5 timesteps on average, the robot decides to curve. Once it decides to curve, it chooses randomly left or right … and then for the next 25 to 75 timesteps it continues curving in that same direction. The counter is set to reset whenever a new state begins.

To ensure that the robot does not fall off the edge, you need to detect when the robot is tipping forward as it gets onto the ramps on the boundaries of the environment. We will need to store a circular array of accelerometer values. Add these before the **WHILE** loop:

```
byte   accelIndex = 0;
double accelValues[][] = new double[10][3];
double accelTemp[] = new double[3];
```

The `accelValues` array will hold ten (x,y,z) accelerometer readings.

Add this code at the top of the **WHILE** loop (after the `//Sense:` comment) so that it reads the latest accelerometer value and places it in the array after the last reading, ensuring that it wraps around to the start of the array again.

```
accelValues[accelIndex] = accelerometer.getValues();
accelIndex = (byte)((accelIndex + 1) % 10);
```

At this point, we have an array that will hold the latest 10 accelerometer readings. Add the following code that computes the average (x,y,z) accelerometer readings from these 10 values and stores this average in the **accelTemp** array.

```
// Total up the past accel values
for (int j=0; j<3; j++)
  accelTemp[j] = 0;
for (int i=0; i<10; i++)
  for (int j=0; j<3; j++)
    accelTemp[j] += accelValues[i][j];
for (int j=0; j<3; j++)
  accelTemp[j] = accelTemp[j]/10.0;
```

Look at slide 15 in the notes and create some **boolean** variables called **tippedForward**, **tippedBackward**, **tippedSideways** and **flat** which look at this **accelTemp** average and determines if the robot has tipped forward, backward, sideways (i.e.., tipped left or right) or is flat, respectively. You will use these booleans in your code. Keep in mind that the values will never be exactly 0 and that the values always fluctuate. Therefore, code such as **`boolean`** `tippedForward = accelTemp[0] < 0;` will not work because a flat robot will have values sometimes a little below 0 and sometimes a little above!

To test your code, print out the values of these 4 booleans right after you set them and then add the following code right before the last two lines of the WHILE loop so that it stops the robot from moving:

```
leftSpeed = rightSpeed = 0;
```

When you run the code, the robot should not move and you should see your Boolean values printing repeatedly. It should be showing that the robot is flat. You can zoom in, select the robot and drag it forward (red arrow) onto the ramp in front of it. Once it is on the ramp, your booleans should show that it is tipping forward. Then use the blue arrow on the robot to spin it around **180** degrees and you should see it showing that the robot is tipping backward.

(b) In your WHILE loop, add appropriate IF statements to 1st SWITCH statement to the **WANDER**, **GO_FORWARD** and **TURN_AROUND** cases such that you implement the yellow arrows in the state machine above.

Also, add code to the 2nd SWITCH statement (for states **GO_FORWARD** and **TURN_AROUND**) so that the **leftSpeed** and **rightSpeed** variables are set appropriately. For the **TURN_AROUND** case, the robot should turn around (left or right randomly) about **180** degrees … until the robot is facing up the ramp. It should not turn for any specific amount of timestep, instead it should just examine the latest accelerometer reading and decide if it still needs to keep turning or if the accelerometer indicates that the robot has turned around. Your code should smoothly and efficiently turn the robot so that it is ready to head back up the ramp again. The **GO_FORWARD** mode is used to actually move the robot up the ramp (after it has turned around) until it is off the ramp and on level ground again. It should move the robot forward smoothly and efficiently, as it examines the latest accelerometer reading.

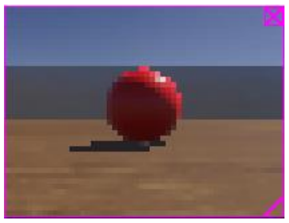
Remove the line that you added near the bottom of the WHILE loop (i.e., `leftSpeed = rightSpeed = 0;`) so that the robot can move again. If your code is written properly, the robot should wander around and never go off the edges because it should turn around when it encounters a ramp. Make sure that this works before continuing.

(c) In the Sense part of your WHILE loop (i.e., after you read the accelerometers and before the first SWITCH statement) add code to read the camera sensor to look for a ball (based on slides 10 and 11 in the notes). Keep in mind that the code in the slides is not complete! It only shows how to read the color values for one pixel. You will need to put the code for reading a pixel in a loop so that you read an entire row of pixels.

You should make more booleans (based on slide 9) to detect whether a ball is on the **detectedLeft**, **detectedRight**, **detectedStraight** ahead or if it is **notDetected**. You should disable your **println()** statements for your accelerometer and add **println()** statements to display these new booelans.

Again, add the following code right before the last two lines of the WHILE loop so that it stops the robot from moving: `leftSpeed = rightSpeed = 0;`

Run your code. You should see the robot's camera image in the left corner of the environment so that it looks as shown here on the right → (If you cannot see the image box appear, then it was probably closed. To view it again, select the robot, then select **Overlays** from the menu. Then select **Camera Devices** and **Show 'camera' overlay** from the menu choices).



Run your code to make sure that it works properly by displaying your **detectedLeft**, **detectedRight**, **detectedStraight** and **notDetected** booleans. When restarting the simulation, the robot should have a red ball centered, so you should see that **detectedStraight** is true, provided that your code has been written properly. While it is running, you can select the ball in front of the robot and slide it sideways left and sideways right and then out of the camera's view. Make sure that all 4 booleans are all producing the correct results before you continue. Once this works, remove the line that you added near the bottom of the WHILE loop (i.e., `leftSpeed = rightSpeed = 0;`) so that the robot can move again.

Now you need to add code to implement the red arrows in the state machine by adding an appropriate IF statement to each of the the **WANDER** and **HOME-IN** states of the first SWITCH statement in the loop. Then add code to the **HOME-IN** state of the 2nd SWITCH statement to steer the robot towards the ball. When in **HOME-IN** mode, the robot should head straight towards the ball that it sees. It should aim reasonably straight towards the ball, correcting its position if it starts veering off (by examining your 4 booleans). If multiple balls are seen, it may be somewhat unpredictable as to how it aims towards the red that it sees but it should be reasonable.

Although the code should work fairly well at the moment, you should implement the remaining purple arrows in the state machine so that the robot does not get stuck on one of the 4 corner pillars. The **AVOID** state should cause the robot to turn left or right (randomly chosen) to avoid the object if any one of its two frontmost proximity sensors detects an object. This behaviour should only kick-in if the robot gets too close to the 4 corner posts in the environment. It should NOT kick-in when a ball is encountered.

Implement the remaining purple arrows in the state machine so that the **PUSH_BALL** mode starts only after a ball has been found (i.e., a ball has been detected and is being homed-in on). It should check the proximity sensors to see if the robot is up against the ball. That is … if the robot was in the **HOME-IN** state and the proximity sensor detects something, you will assume that it is a ball. The robot should then go into the **PUSH_BALL** state which simply causes the robot to go straight so that it pushes the ball outwards towards the edge of the crate. If it ever loses the ball while in this mode, the robot should start wandering again. If it encounters the ramp (i.e., the robot starts tipping forward), then it should go right into **TURN_AROUND** mode. By then, the ball will have likely rolled off the edge of the ramp.

Test your code. It should perform quite well. On rare occasions, things could happen and the robot may still end up off the edge of the ramp. Don't worry about this. If you want, you can try to debug why this happens and you can alter the state machine accordingly to fix it. However, you do not need to worry about this. Your robot should be able to push all the balls off of the edges.

Submit your **Lab3Controller.java** code. Make sure that your name and student number is in the first comment line.

Tips:

- For quick tests, you can move the robot to any location and save the world so that the robot starts there each time. But you'll need to make a backup of the original world file so that you can use it for your final test.