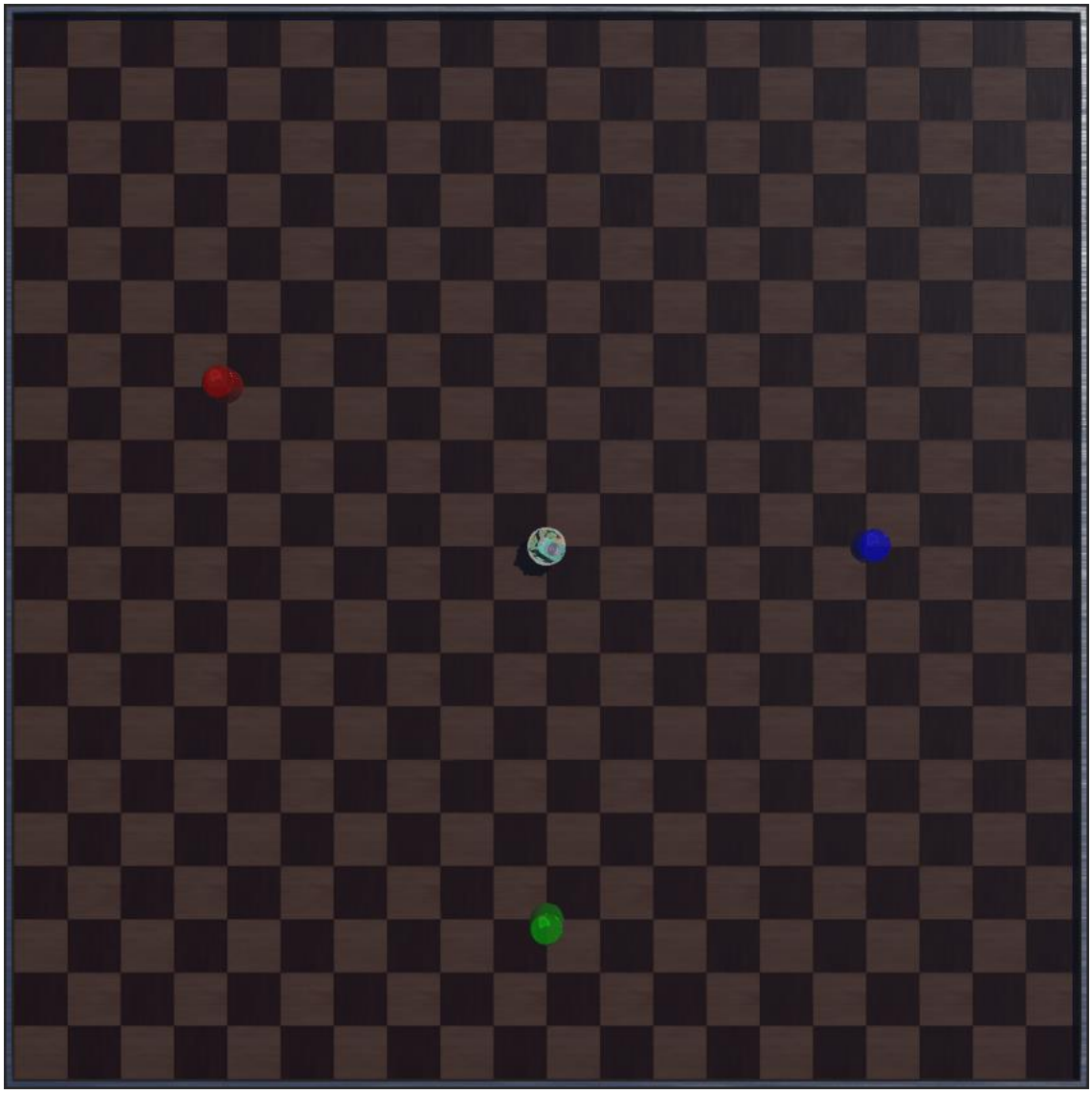
LAB 6 – Beacon Triangulation

(1) Download the **Lab6_BeaconTriangulation.zip** file and unzip it. Load up the **BeaconWorld** world. The goal of this lab is to compute the coordinates of the robot as it moves along a pre-defined piece-wise linear path in the environment and stops at various locations. You will compute the coordinates based on the triangulation technique discussed in the notes. There are three colored beacons that the robot must identify (by using its camera) to compute its location.



(2) The **Lab6Controller** code has been started for you. The robot starts at position ($\mathbf{x}_0$,$\mathbf{y}_0$, $\mathbf{a}_0$) = (0, 0, 90°). The code already moves the robot to a series of **16** locations. At each location, it stops and spins 360°, to look for all the beacons. The code is based on the inverse kinematics that you did in the previous lab. You will NOT alter the code to move the robot. Instead, you will insert code to make use of the camera while the robot is in the spin mode looking for the beacons. You will record the angles to the **red**, **green** and **blue** beacons and then plug in the mathematical formula to determine the location.

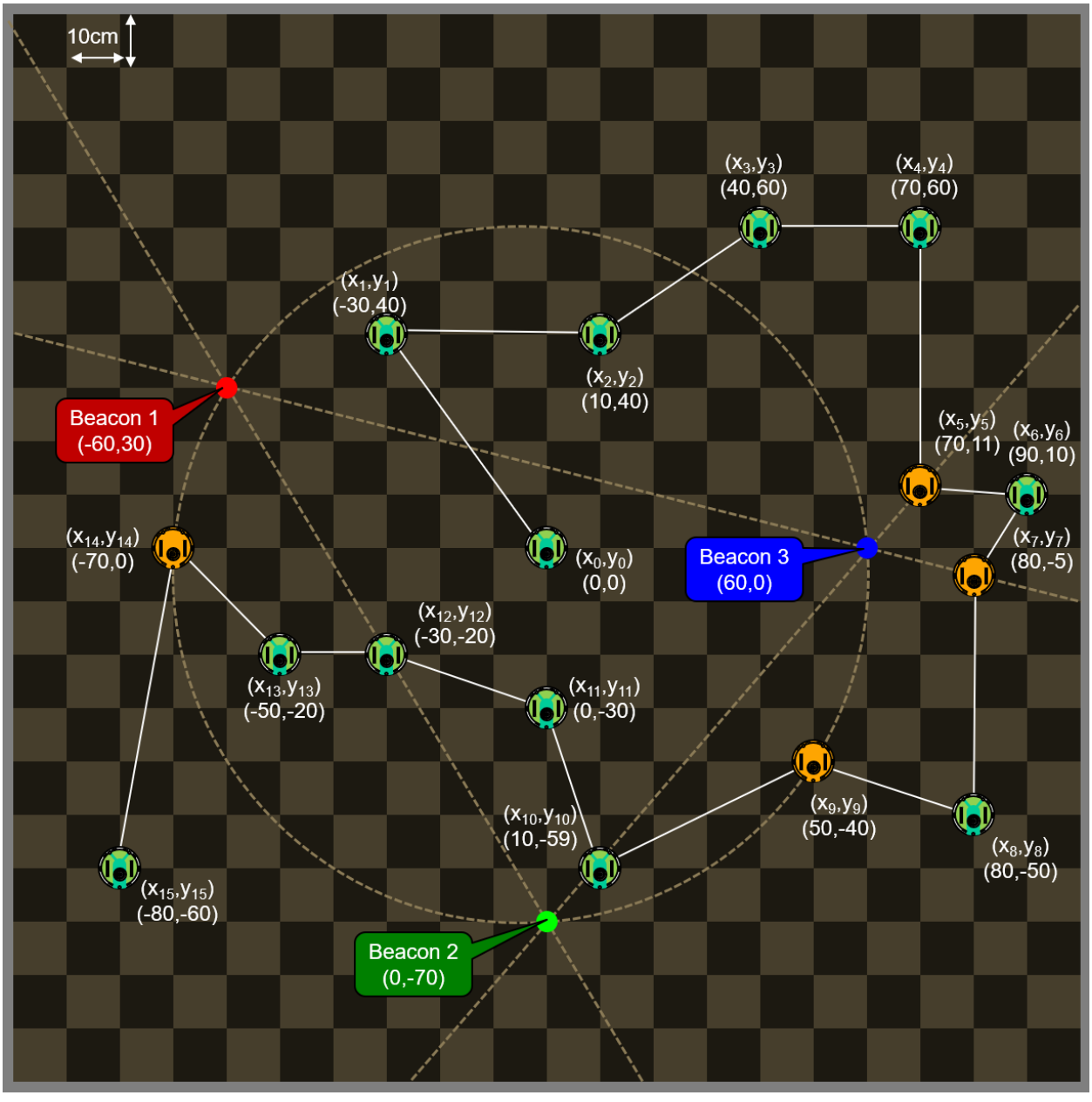
Here is code to check if pixel at column **i** (of the horizontal center row) is **red**:

```
// check if it is a red pixel
if ((Camera.imageGetRed(image, 52, i, CAMERA_HEIGHT/2) > 60) &&
    (Camera.imageGetGreen(image, 52, i, CAMERA_HEIGHT/2) < 50) &&
    (Camera.imageGetBlue(image, 52, i, CAMERA_HEIGHT/2) < 50)) {
  ...
}
```

Assume the following constants when determining a camera pixel color:

Red Beacon:	if (red>60 and green<50 and blue<50)
Green Beacon:	if (red<50 and green>60 and blue<50)
Blue Beacon:	if (red<50 and green<50 and blue>60)

Below is the path that the robot will take in the environment. The locations of the robot are shown. However, you will not be able to compute these locations too accurately. In fact, in the code given to you, I had to tweak these numbers a little, since the physics simulation of the robot does not move too accurately. Do NOT change the tweaked numbers. For all intent and purposes, you can assume that the robot moves to the locations shown below. Note that there are four orange locations. These are locations that the robot cannot compute its location accurately. The two rightmost … because one beacon is out of the line-of-sight … and the other two because the robot lies on the circle formed by the three beacons.



Your code must show your **16** estimated locations in a format like this (although your estimates will not be this accurate):

```
(x0, y0) = (0, 0)
(x1, y1) = (-30, 40)
(x2, y2) = (10, 40)
(x3, y3) = (40, 60)
(x4, y4) = (70, 60)
(x5, y5) = CANNOT COMPUTE LOCATION
(x6, y6) = (90, 10)
(x7, y7) = CANNOT COMPUTE LOCATION
(x8, y8) = (80, -50)
(x9, y9) = (50, -40)  INACCURATE
(x10, y10) = (10, -59)
(x11, y11) = (0, -30)
(x12, y12) = (-30, -20)
(x13, y13) = (-50, -20)
(x14, y14) = (-70, 0)  INACCURATE
(x15, y15) = (-80, -60)
```

For locations that cannot be computed or cannot be determined accurately, indicate this bedside the coordinate as shown above. But DO NOT assume that it is positions **5**, **7**, **9** and **14**. That is … your code should work for any path given and automatically determine the locations that are not computable. Note that for points **9** and **14**, you can determine that the points are inaccurate because the absolute value of **d** will be less than 100.

DO NOT change the **MAX_SPEED** constant. It appears that the Webots simulator does not calculate the proper robot movements if you change this value. The robot will run slow.

Submit your **Lab6Controller.java** code. Make sure that your name and student number is in the first comment line.

Tips:

- Comment out the **FOR** loop in the **main()** function so that the robot just computes its location by spinning and not moving anywhere. Once the original location gives you a nice estimate, you can then get your code working for all the locations.