Drawing Robot to Record Paths Formed from Arc and Line Segments

Charles N. Winton
University of North Florida
cwinton@unf.edu

Abstract

By making small modifications to the graphics simulator used with the Botball Educational Robotics program and to an iRobot Create base, both a simulated and actual drawing robot can be produced for tracing out line and arc segments using identical programs in each environment. This leads directly to an examination of drawing geometric figures using dead reckoning and demonstration of congruent behavior between simulator and actual robot. In particular, the 5 centered arch approximation for drawing an elliptical arch using ruler and compass is examined and used for deriving means to trace out an elliptical path using a small series of arc segments.

1 Introduction

With the introduction of the iRobot Create module [1], a low cost robot base unit employing differential steering is now available for educational use. The Create Open Interface [2] is the specification for the built-in command structure and state variables for the Create module. These are accessed via (TTL level) serial connection, providing a means for controlling the base and for accessing its built-in sensors. A 25-pin connector in the "cargo bay" of the Create module provides additional digital and analog signal I/O capabilities.

The XBC component of Interactive C (IC) as supplied by the KISS Institute for Practical Robotics (KIPR) in support of the Botball Educational Robotics Program [3] includes a library of functions and global variables for utilizing the Create open interface from IC. In this manner, the XBC can control a Create module, obtaining its sensor values and issuing Create movement commands. Moreover, the IC environment also includes a graphical simulator (written by David P. Miller) that can be configured for use with either Create or XBC motor commands. The simulator provides a basic Create-like robot to use for testing movement strategies in user defined "worlds". The robot can be configured to paint (or not) a trail to record its movements.

One question is whether the simulated robot's behavior and that of a Create are essentially the same when controlled by the same program. Addressing this question requires providing the Create with an ability to paint a trail. Once this capability has been added, then a next step is to examine what kinds of paths can be generated. Straight line and arc segments form the basic building blocks and provide a natural tie between robotic implementation and underlying geometric principles. Paths corresponding to conic sections are a natural next step.

In this paper we examine implementation of the basic capabilities for painting a trail while following straight line or arc segments, doing so in such a manner that the exact same program works for both simulation and actual application. This is then employed to demonstrate construction of an elliptical path using arc segment approximation.

2 Differential Steering

Differential (two-wheel) steering is commonly used for robots because it is mechanically simple to construct and provides a simple means for navigating from point A to point B (turn in place and go straight). If the velocities of the two wheels are of different (fixed) magnitudes, then the path traced by the midpoint between the wheels will be an arc. Determining the path that will be followed is a forward kinematics problem, where wheel speeds are given. For purposes of analysis, physical issues of wheel slippage, friction, and time to accelerate to constant wheel speed are ignored.

Given a wheelbase of length w, to turn through an arc of radius r and angle α the outer wheel needs to travel a distance $(r + w/2)\alpha$ and the inner wheel a distance $(r - w/2)\alpha$. If t is the amount of time taken for tracing the arc, then if the outer wheel moves at velocity v_1 and the inner at velocity v_2, you have the two equations

- $(r + w/2)\alpha = v_1 t$
- $(r - w/2)\alpha = v_2 t$

which combine to yield $(r + w/2)/(r - w/2) = v_1/v_2$ or $v_2 = v_1(2r - w)/(2r + w)$. Hence, to turn through angle α, fix the outer wheel velocity wheel velocity v_1 and calculate v_2, traveling until the outer wheel has gone $(r + w/2)\alpha$. Note that the equations remain valid for $0 \le r \le w/2$.

The Create module open interface commands at present do not provide direct access to wheel encoder data to calculate wheel distances (an upgrade that includes this capability should soon become generally available); however, there are open interface commands for estimating the distance the midpoint of the unit has traveled, including when the unit is progressing along an arc. Since path behavior is described using the center point between the wheels, both the simulated robot and its Create implementation need to paint trails from the center.

As delivered, the Create has no means for installing a painting device in its center. The simulated robot (as delivered) paints its trail from its rear end rather than the center. Since the graphical simulator that comes with IC is written in IC, only minor modifications are needed so that the trail is painted from the center of the robot as it maneuvers. Analogously, by drilling a ½" hole in the Create cargo bay angled toward the center, a standard white board marker can be installed to be raised or lowered to paint a (real) trail on a white board surface as the Create runs the same program.

Since the Create open interface limits maneuvers to line and arc segments, it is natural to explore the limitations for maneuvering and dead reckoning this imposes. Moreover, maneuvers can be tested using the simulator and then verified using the Create. A first exercise is maneuvers analogous to the skating challenge of drawing "school figures" such as figure eights and S curves. Drawing an ellipse is a more significant challenge, especially given the limitation of using only arc and straight line segments. This can be achieved by use of an obsolete drafting technique, the 5-centered arch, which produces a close approximation for an elliptical arch using ruler and compass methods. Its implementation will conclude this paper.

3 Drawing Functions

The first decision is what drawing functions to implement. It is evident that a function "arc" to draw arc segments and a function "move" to draw line segments are desirable. Additionally a function "pen" to turn on or off the paint trail is needed. For a scratch built robot controlled by the XBC, XBC motor commands can be used for this purpose. Since our focus here is on the Create, the IC library functions for using Create open interface commands are employed.

The IC Create library functions are identified by the prefix "create_" and global state variables by the prefix "gc_". The primary Create movement command for our purposes is "create_drive_direct" since it provides means for controlling each wheel's velocity. Unless command scripts are employed (supported in IC's Create library but not by the simulator), the Create module has to be polled repeatedly via its serial interface to see if the designated distance has been reached. It is also necessary to limit polling frequency to avoid corrupting the serial data stream. A viable move function that works for either the simulator or Create can be programmed as follows:

```
void move(float x, int vel) { // move x centimeters,
                              // vel in mm/sec (-500 to 500)
    int dist; dist=(int)(x*10.0); // change cm to mm
    create_distance();  // update Create internal distance
    gc_distance=0;      // and initialize distance global
    msleep(50L);        // pause before next signal to Create
    if (dist < 0) {     // conform for 2 possible cases
        dist=-dist; vel=-vel;
    }
    if (dist != 0) {
        create_drive_direct(vel, vel);
        if (vel > 0)    // pause between distance checks
          while(gc_distance < dist) {
              create_distance(); msleep(50L);
          }
        else
          while(gc_distance > -dist) {
              create_distance(); msleep(50L);
          }
    }
    create_stop();  // stop
}
```

If in testing the Create doesn't go straight, the bias can be compensated for by adding an appropriate adjustment to one of the wheel velocities in the create_drive_direct command, although that could adversely affect simulation results.

Drawing an arc is similar, except motor velocity information needs to be calculated:

```
void arc(float r, float deg, int vel) {
    // arc of radius r cm; deg,vel ++,-- is CCW, +-,-+ is CW
    int dist, velR, velL, curve; // vel is to be the outer wheel speed
    float w=24.5; // Create wheel separation in cm
    create_distance(); gc_distance=0; msleep(50L);
    if (deg > 0.0) { // average of the two wheel distances is measured by Create
        velR=vel; velL=(int)((2.0*r-w)/((2.0*r+w))*(float)vel);
        dist=10*(int)(((pi*(2.0*r-w)*deg/360.0)+(pi*(2.0*r+w)*deg/360.0))/2.0);
    }
    else {
        velL=vel; velR=(int)((2.0*r-w)/((2.0*r+w))*(float)vel);
        dist=10*(int)(((pi*(2.0*r-w)*-deg/360.0)+(pi*(2.0*r+w)*-deg/360.0))/2.0);
    }
    // use the differential velocities to move in arc
    if (dist != 0) {
        create_drive_direct(velR, velL);
        if (vel>0)    // pause between distance checks
          while(gc_distance < dist) {
              create_distance(); msleep(50L);
          }
        else
          while(gc_distance > -dist) {
              create_distance(); msleep(50L);
          }
    }
    create_stop();  // stop
}
```

Activation/deactivation of the pen for the simulator is just a matter of flipping a global variable named "iROBOtrail".
For the Create an assembly where pen motion is controlled by a servo motor is used. Since servo motors are ignored by the simulator, both requirements can be included within a single function.

```
void pen (int up) {
    if (up==1) {
        iROBOtrail=0;
        set_servo_position(pen,20);     // calibrated servo position for up
        enable_servos(); sleep(1.0);    // servos on to lift pen
        disable_servos();               // servos off
    }
    else {
        iROBOtrail=1;
        set_servo_position(pen,170);    // calibrated servo position for down
        enable_servos(); sleep(1.0);    // servos on to lower pen
    }
}
```

The pen construction used with the Create is pictured in Figure 1 below.

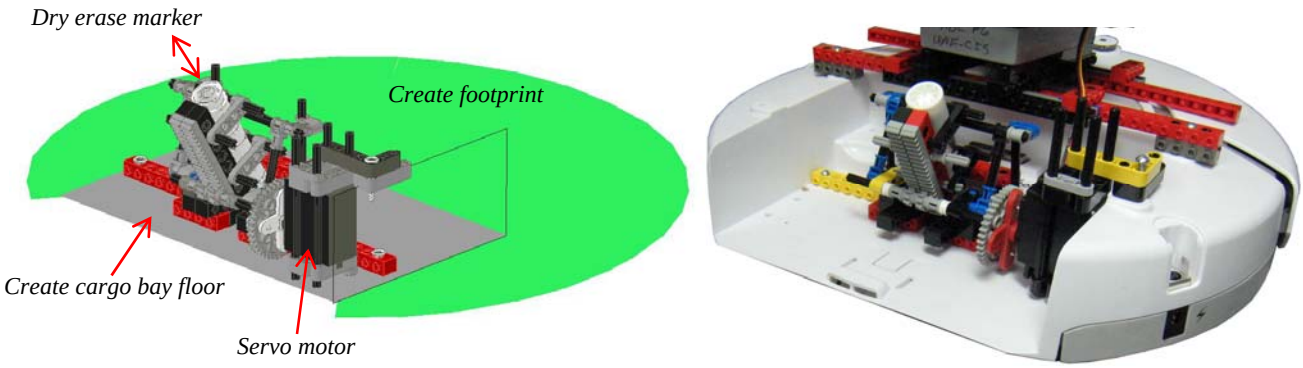

Figure 1: Pen Construction for Create Module

4 School Figures

A simple figure 8 can now be drawn by:

```
pen(0);
arc(40.0,90.0,200); arc(40.0,-360.0,200); arc(40.0,270.0,200);
pen(1);
```

Time lapses between distance updates from the Create (or the simulator for that matter) cause variations in accuracy depending on the parameters employed and provide an object lesson on the limits of dead reckoning. When this code is run on both the simulator and the Create, some drift is evident as can be seen in Figure 2.

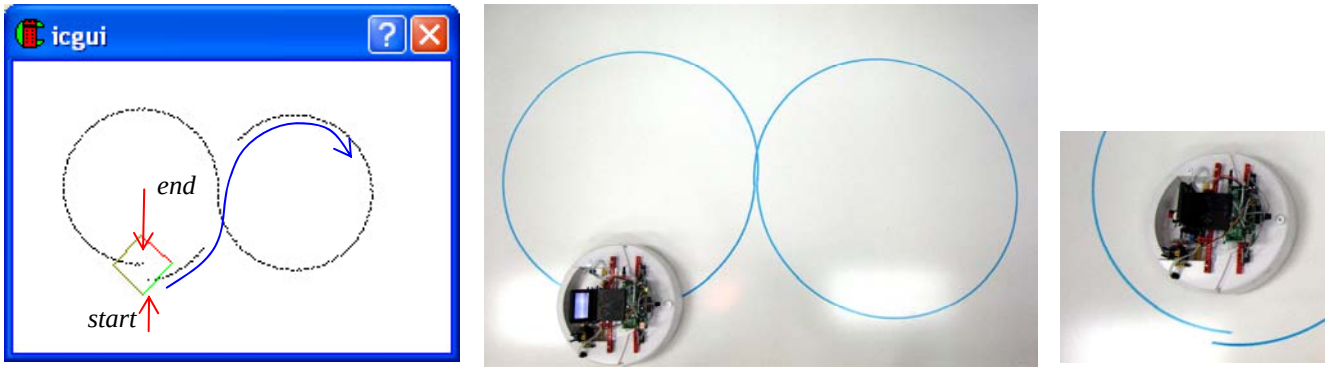

Figure 2: Simple Figure 8 as Drawn by Simulator and Create

At this point, one can begin to consider what might be required for a robot to approximate movement along a path that is not given by arcs or straight lines. Continuous adjustment of motor speeds is beyond the scope of this paper, and in fact the Create open interface limits movement to arcs and straight lines. To illustrate the difficulty this imposes for approximating general curvilinear motion, the problem of following an elliptical route will be considered.

5 Drawing an Ellipse

An ellipse (aligned on x and y axes) is simply the locus of points satisfying an equation of the form $x/a^2 + y/b^2 = 1$. If a>b, then the line segment from (0,0) to (a,0) is called the semi-major axis and the one from (0,0) to (0,b) the semi-minor axis. An ellipse is also a conic section and can be described geometrically in a variety of ways. The interested reader is referred to [4] for a list of these. Our drawing robot is limited to drawing arcs and straight lines, so it is evident that a series of arcs need to be used to approximate an elliptical path. It is not obvious what these should be. Interestingly, pre-CAD drafting techniques provide a ruler and compass construction that with just 3 arcs can form a close approximation for one quadrant of an ellipse. Although considered obsolete for drafting, this technique provides an easily implemented means for using our drawing robot to follow an elliptical path.

5.1 Drawing an ellipse using arcs – 5-centered arch

The 5-centered arch technique is so named because it provides a good drawing approximation for an elliptical arch using just 5 compass centers. The approximation is reasonably accurate so long as the ellipse is not greatly elongated or too circular. It only requires specifying the semi-major and semi-minor axes of the ellipse (a and b in Figure 3 below).

The technique is simple: using ruler and compass

1. Construct an a x b guide box
2. From corner (a,b) run a line orthogonal to the line segment (0,b):(a,0) to locate center C3 on the y-axis and C5 on the x-axis
3. Using the distance r from C5 to (a,0), measure off 2r from (0,b) to locate point (0,b-2r) on the y-axis
4. Using the distance between C3 and (0,b-2r) as a radius, locate center C4 by drawing an arc from C3 to where it intersects the circle of radius 2r having center C5
5. Using guidelines through C3:C4, C4:C5, and C3:C5 to determine arc end points, draw an arc radius of radius r from (a,0) with center C5 to the C4:C5 guideline, then one of radius 2r from the C4:C5 guideline with center C4 to the C3:C5 guideline, and finally one of radius the distance from C3 to (0,b) with center C3 across to (0,b).
6. Mirror points C1 and C2 provide the other 2 centers to complete the arch.

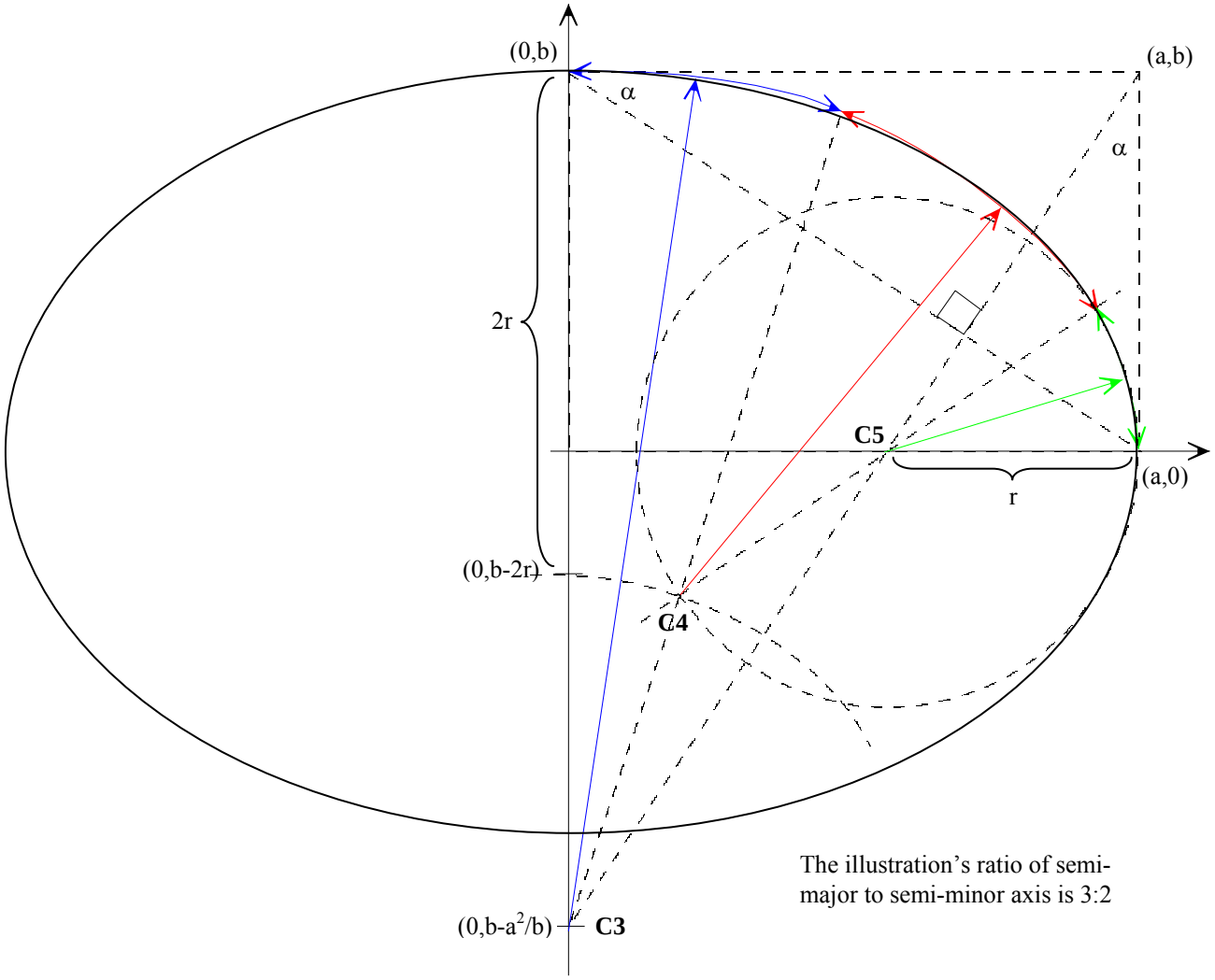

Figure 3: Ruler and Compass Construction for 5-Centered Arch

Note: If semi-minor axis b is too close to semi-major axis a, the method fails. This happens if C3 falls inside the ellipse (when $2b^2 > a^2$).

It is not hard to show that $r = b^2/a$ ($\tan \alpha = b/a = r/b$) which provides two of the three needed radii (r and 2r). The third radius (from C3) is obtained similarly and is a^2/b. The remaining challenge is to determine arc lengths as required by the arc function (recall that this is just the radius times the angle of the arc). Finding the angle requires determining the coordinates (i, j) for C4, which is the point of intersection of two circles centered on C3 and C5 as specified earlier. Knowing C4 means the slope of the line through C3 and C4 and the slope of the line through C4 and C5 can be determined. The angle β subtended by the intersection of two lines of slope s_1 and s_2 is given by (see for example [5])

$\tan(\beta) = |s_1 - s_2| / |1 - s_1 s_2|$

For the special case that one of the lines if vertical, the angle is given by

$\tan(\beta) = 1/|s_1|$ where s_1 is the slope of the non-vertical line.

A bit of computation is required to obtain the coordinates (i, j) of C4 as given by Figure 4 below.

1. The distance d between C3 = $(0, b - a^2/b)$ and C5 = $(a - b^2/a, 0)$ can be calculated and is $(a/b + b/a)(a/b - b/a)(a + b)(a - b)$
2. Since $r^2 - s^2 = h^2 = (a^2/b - 2r)^2 - (d - s)^2$ (via Pythagoras)
   $s = (r^2 + d^2 - (a^2/b - 2r)^2) / 2d$
   $h = \text{SQRT}(r^2 - s^2)$
3. The point (h_1, h_2) bisecting the line between the two points where the circles intersect can be determined from the equation of the line C3C5 of slope t = a/b, which is given by
   $y = t(x - a + b^2/a)$
   $h_2 = t(h_1 - a + b^2/a)$; $(h_1 - a + b^2/a)^2 = s^2 - h_2^2 = h_2^2/t^2$
   $h_2 = -st/\ \text{SQRT}(1 + t^2)$
   $h_1 = a - b^2/a - s/\text{SQRT}(1 + t^2)$
4. The equation of the line (of slope – b/a) through (h_1, h_2) and C4 is given by $y = h_2 - (b/a)(x - h_1)$
5. This provides 2 simultaneous equations
   $h_2 - j = (b/a)(i - h_1)$ and $h^2 = (h_1 - i)^2 + (h_2 - j)^2$ from which we obtain
   $i = h_1 - ah/\text{SQRT}(a^2 + b^2)$ and
   $j = h_2 + bh/\text{SQRT}(a^2 + b^2)$
6. Using the coordinates of C3, C4, and C5, the slopes of the bounding lines for each arc can now be calculated
   slopeC4C5 = $-j/(a - b^2/a - i)$
   slopeC3C5 = a/b (this is t)
   slopeC3C4 = $(j - b + a^2/b)/i$
7. With this information, the tangent formulae noted above can now be applied to determine the needed angles

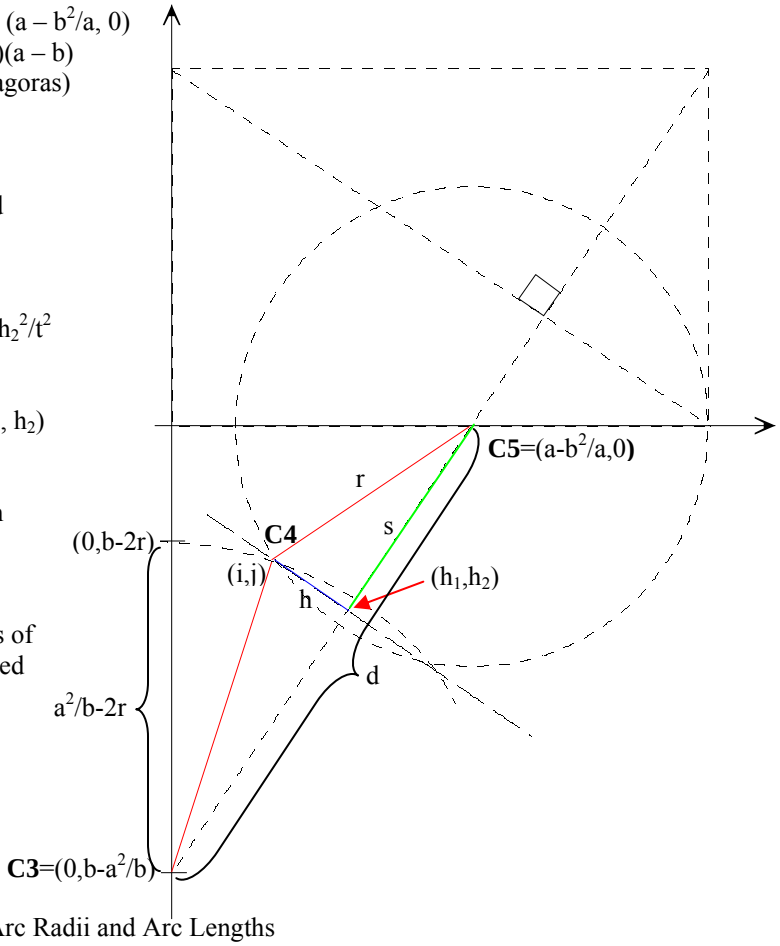

Figure 4: Deriving Arc Radii and Arc Lengths

It is a (relatively) small matter to imbed these equations within an IC program. Note that for a full ellipse, where the arcs cross an axis to the next quadrant the sequence of arcs repeats in reverse order, so a single arc can accomplish the transition across each axis. Hence if we start with the arc of radius 2r and double the angle for each of the other two we can obtain a reasonable approximation for an ellipse with just 8 arcs.

Figure 5 shows the results of implementing this approach using the simulator and the Create.

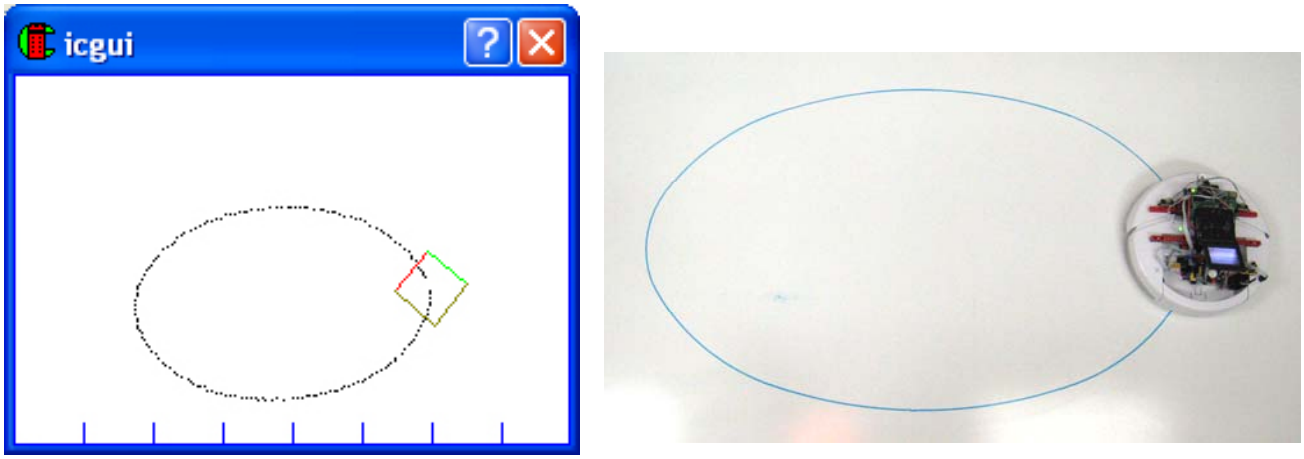

Figure 5: Drawing a Full Ellipse

6 Conclusion

Whenever robot simulation is used, there is always a question of congruency between simulated and implemented robot behavior. More often than not the simulation produces analogous behavior, but not congruent behavior. Using IC's robot simulator in conjunction with an XBC controller and an iRobot Create base good congruency can be observed for paths constructed using straight line and arc segments. Using this approach to approximate an elliptical path demonstrates that the approach can be used to explore ways of approximating curvilinear motion and contrast simulated vs. implemented robot behaviors.

References

[1] iRobot Create Open Interface (v2) `http://www.irobot.com`, 2006.

[2] iRobot Create Programmable Robot `http://www.irobot.com/create`.

[3] KIPR. Botball robotics education. `http://www.botball.org`, 2008.

[4] Steven Dutch. Constructing Ellipses. `http://www.uwgb.edu/DutchS/MATHALGO/Ellipses.HTM`.

[5] Glen James and Robert C. James. *Mathematics Dictionary*. D. Van Nostrand Company, 1966. page 13.