Omnidirectional Robot Control with the Botball XBC Controller: Software library Development.
Jeffrey G. Cumber and Preston R. Johnson
Robert E. Lee Engineering Academy
cumberj@duvalschools.org, prjohn16@yahoo.com

Omnidirectional Robot Control with the Botball XBC Controller: Software Library Development

1. Abstract

The goal of our research was to develop a software library that will make it easy to control an omnidirectional robot using an XBC or CBC Botball controller. We developed functions that enable the robot to slide along a heading, rotate, and move in an arc centered on any wheel or face.

2. Introduction

Omnidirectional robots are robots that can move instantaneously in any direction from any heading without having to turn first. These robots are also called holonomic. A holonomic drive platform is advantageous in robotics because it allows faster response to changing environmental circumstances. These robots can "slide" out of the way without having to wait for the base to turn to a particular heading. In contrast, robots such as the iRobot Create must take time to turn and then move out of the way of a potentially undesirable condition.

3. Robot Description

Omnidirectional robots typically are of two different types: four-wheeled and three-wheeled. They can also be of two different configurations: leading-face or leading wheel. Combining the types and configurations gives a total of four different robot setups. A four wheel wheel-leading robot is a '+' robot. A four wheel face-centered robot is an 'X' robot. For the three-wheeled robots, face-leading robots are 'Y' robots and wheel-leading robots are 'Δ' (Delta). See figure 1.

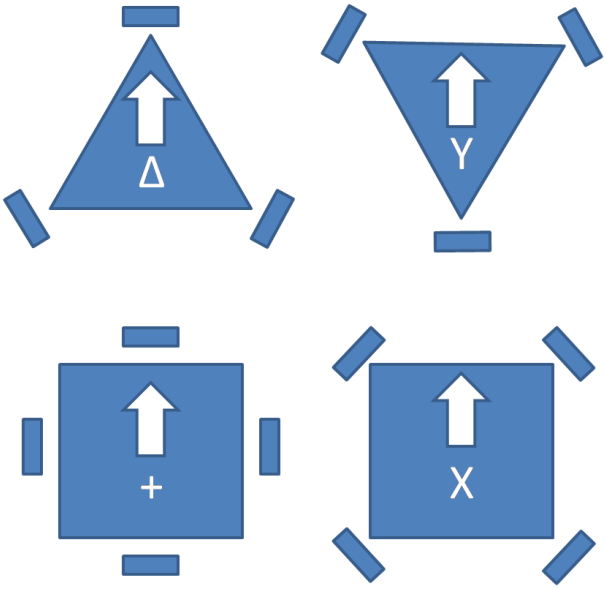

Figure 1- Holonomic robot configurations

The advantage of a four-wheeled robot is efficiency. In a four-wheel setup, for some angles (0, 90, 180, and 270 for a wheel-centered robot) two of the wheels are in the direction of motion and can run at full power while the other two can be idle. For other angles the calculations are trivial. In a three-wheeled setup at most one wheel is in the direction of motion and the other two must be powered at less than full value to achieve the desired motion. This complicates the calculations of the individual motor speeds.

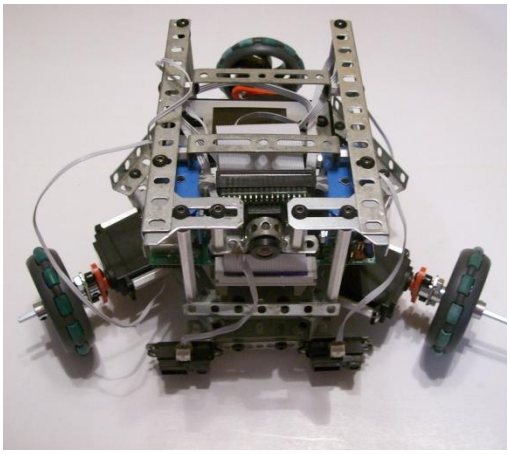

Figure 2- XBC-based holonomic 'Y' style robot

The advantages of the three-wheeled platform make it more suitable, in our opinion, for a robot drivetrain even with the increased difficulty of calculations. The top advantage is that for a three-wheeled platform, all three wheels are always in the same plane and touching the floor surface. This is necessary for omnidirectional wheels to work. For a four-wheeled system, one wheel may be above the surface, giving no traction and rendering holonomic movement impossible. Another advantage of the three-wheeled system is size reduced size, cost, and weight due to only three wheels and motors being used.

Our robot is therefore of the three-wheeled variety. We chose to make it face-leading to facilitate the placement of sensors at the front and to enable the robot to get closer to objects. Using our configurations defined above makes it a 'Y' type robot (Fig. 2). Our numbering convention for wheels and faces is to call the leading face F1, and proceed clockwise around the robot. The wheel numbering convention will correspond to the face opposite the wheel. Thus, W1 is at the rear of the robot, opposite F1 (Fig 3.) All library functions will be written for this type of robot due to time constraints, but translating them to other configurations is not difficult.

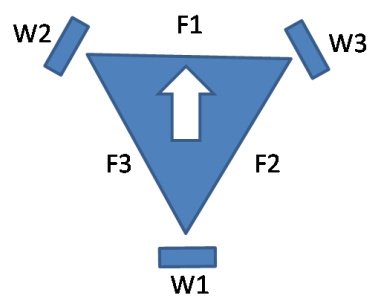

Figure 3- Wheel and Face numbering

4. Function Descriptions

a. spin(int degrees, int speed); Rotates robot in place clockwise for a specified number of degrees at a specified speed. This is the easiest function to write, as rotation is achieved by turning all three motors on at the same speed in the same direction. To calculate degrees of turn, motor counter positions are stored and compared to a calculation of clicks/degree. Negative speed numbers are used to turn counterclockwise.

b. vector(int angle, int speed); Slides robot on a heading of clockwise degrees from forward at a specified speed. To develop this function we thought about what

happens when the robot is moving forward. When the robot moves in a straight line forward, W1 is not moving, and W2 and W3 are moving at full speed. However, due to the angles of the wheels (120 degrees apart, or 60 degrees from the center line) the maximum speed is speed times the cosine of 60 degrees. Thus, in the three-wheeled configuration we lose maximum speed in return for maneuverability. When the robot is sliding on a heading of 90 degrees, W1 is now parallel to the direction of motion and can contribute full speed to movement. We now have a relationship between the speed of W1 and the heading angle. At a zero degree angle, the speed of W1 is zero, and at 90 degrees the speed is full, so we can define the speed of W1 as a function of heading angle by W1speed = speed * sin(angle). We do need to reduce this speed by a factor of cos(30) at the 90 degree heading to make up for the angles of W2 and W3. For W2 and W3, they are also moving at full speed but W3 is reversed from the zero degree heading value. Now we have the speeds of W2 and W3 related to the heading angle: W2speed = speed * cos of heading angle + 60 degrees, and W3speed = speed * cos of heading angle – 60 degrees.

c. f_arc(int face; int speed, int radius); Rotates robot in an arc centered on a particular face with a specified radius and speed. A radius of 0 (zero) will rotate the robot around a stationary face. Really all this function does is turn on the wheel opposite the desired rotation face at the desired speed. The two wheels adjacent to the face do not move, and the robot revolves around them. Increasing the radius will turn on the stationary wheels at increasing speeds, so a radius of 100 will turn the wheels on full and the robot will slide in a straight line sideways. This method of using a "percent" radius was chosen because as the radius gets closer to a straight line the value approaches infinity. The numbers get very large and it becomes difficult to handle them in the context of our control method.

d. w_arc(int wheel; int speed, int radius); Rotates robot in an arc centered on a particular wheel with a specified radius and speed. A radius of 0 (zero) will rotate the robot around a stationary wheel.

5. Future Development

Currently we are working on an additional function that will combine the vectoring function with the spin function, allowing the robot to turn *while* sliding. This functionality is necessary for true holonomic motion, but it is difficult to implement. To get this type of motion from our platform, we need a constantly varying motor signal, which can be done in a number of ways. The first is to generate an array of motor speeds, and then write those speeds to the motors one at a time using a `for` loop. We can also do calculations on the fly, but we've found that the number of trig calculations slows down the robot considerably for long movements (and we do not have full functionality as of yet.) With this approach we still have to write our motor speeds one at a time using a `for` loop.

We have also developed a unique motion tracking protocol that incorporates both rotation and vectoring to follow objects, as well as object avoidance using IR Rangefinders. These two methods are currently being revised and rewritten and will be included as space and time allows.

Our ultimate goal is to create a flexible platform that can be used to compete in the KIPR Open and other robotics competitions. We have seen in past years' competitions situations in which the ability to move in any direction instantaneously would be a great benefit.

6. Functions

```
void spin(int speed){
    mav(1, speed);
    mav(2, speed);
    mav(3, speed);
}

void vector(float angle, float speed){
    float r_angle = (angle * pi / 180.0);
    int spd1 = (int)(cos(r_angle) * speed);
    int spd2 = (int)(cos(r_angle+(pi/3.0)) * -speed);
    int spd3 = (int)(cos(r_angle-(pi/3.0)) * speed);
    mav(1, spd1);
    mav(2, spd2);
    mav(3, spd3);
}

void f_arc(int face, int speed, int radius){
   if (face=1){
        mav(1, speed);
```

```
        mav(2, -(radius*speed/100));
        mav(3, (radius*speed/100));
        }
  if (face=2){
        mav(2, speed);
        mav(3, -(radius*speed/100));
        mav(1, (radius*speed/100));
        }
  if (face=3){
        mav(3, speed);
        mav(1, -(radius*speed/100));
        mav(2, (radius*speed/100));
        }
}

void w_arc(int face, int speed, int radius){
  if (face=1){
        mav(1, radius * speed / 100); //percent radius
        mav(2, speed);
        mav(3, speed);
        }
  if (face=2){
        mav(2, radius * speed / 100);
        mav(3, speed);
        mav(1, speed);
```

```
            }
        if (face=3){
            mav(3, radius * speed / 100);
            mav(1, speed);
            mav(2, speed);
            }
    }
```

7. References

Rojas R and Förster A G: "Holonomic Control of a robot with an omnidirectional drive." *Künstliche Intelligenz*, BöttcherIT Verlag, 2006.

Yong L, et al: "Omni-directional mobile robot controller based on trajectory linearization." *Robotics and Autonomous Systems 56 (2008) 461–479*.

"How to build a Robot Tutorial" *Society of Robots*, http://www.societyofrobots.com/robot_omni_wheel.shtml. Accessed 5/28/2010.

Kruhak, I: "Holonomic Robot." http://cs.nmu.edu/~SeniorProjects/ikruhak/Holonomic%20Robot.ppt. Accessed 5/28/2010.