Using the Gyroscope in Botball

Reza Torbati, Andrew Zhang
Norman Advanced - Reza Torbati: kofcrotmgiv@gmail.com, Andrew Zhang: infinitepolygons@gmail.com

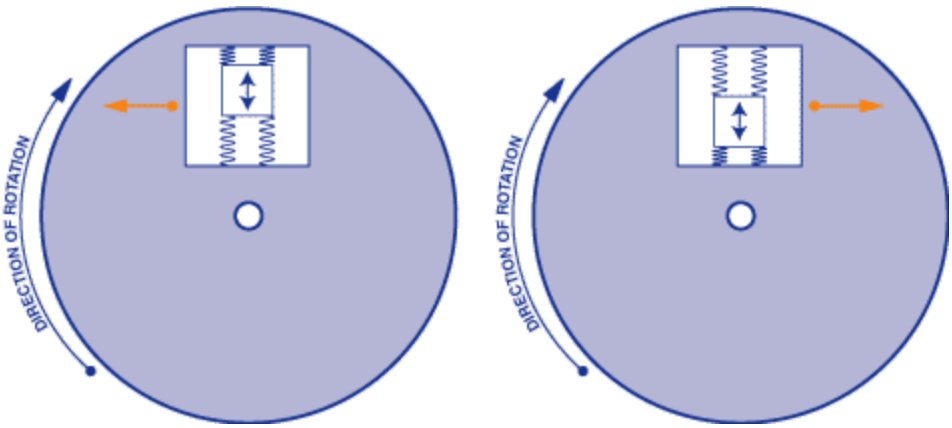
Using the Gyroscope in Botball for Simple and Consistent Navigation

1. Introduction

Consistent movement has been a goal chased within Botball since its inception. Both turning and driving straight, with consistency, are invaluable. In 2016, KIPR released the Wallaby, which contained a nine-axis inertial measurement unit (IMU) that included an accelerometer, magnetometer, and a gyroscope. Earlier Botball controllers lacked gyroscopes. This built-in gyroscope is capable of accurately tracking the angular velocity of the Wallaby while driving, which gives the robot the ability to turn accurately and consistently, as well as the capability to drive in a straight path.

2. How an MEMS Gyroscope Works

There is a wide array of methods by which MEMS (microelectromechanical systems) gyroscopes operate. Fundamentally, they measure how much an object with known characteristics (such as spring constant) deflects/shifts due to centrifugal acceleration; this measured movement can then be translated into angular velocity.[1] This allows us to measure the angular velocity of a robot using something that is extremely small and can be built into the board.



3. Using the Gyroscope in Botball

A. Reading from the Gyroscope using KIPR's libraries

KIPR has implemented several functions to read from the gyroscope. The three most important ones are:

- gyro_z() (to find the angular velocity in the z axis for turning when the Wallaby is parallel to the ground)
- gyro_y() (to find the angular velocity in the y axis for turning when the Wallaby is perpendicular to the ground and screen is sideways)
- gyro_x() (to find the angular velocity in the x axis turning when the Wallaby is perpendicular to the ground and screen is normal)

KIPR also has a function called gyro_calibrate(), but it is not implemented and does not do anything as of the writing of this paper.

B. Finding the Bias

Most MEMS gyroscopes are consistently off by a specific value known as the bias. The bias is caused by various environmental factors such as gravity, slight vibrations in the gyroscope, changes in temperature and more.[2] While some MEMS gyroscopes have built in ways to account for the bias, the Wallaby's lacks such compensation. To get an accurate reading from the gyroscope in a Wallaby, the bias must be found and subtracted from all future readings. To do this, find what the gyroscope is reading when it is completely still (when the gyroscope should read a value of 0) and then set that value to a variable which can then be subtracted from all future readings.

```
void calibrate_gyro()
{
    //takes the average of 50 readings
    int i = 0;
    double avg = 0;
    while(i < 50)
    {
        avg += gyro_z();
        msleep(1);
        i++;
    }
    bias = avg / 50.0;
    printf("New Bias: %f\n", bias);//prints out your bias.
}
```

This is an example of one function that finds the bias of the gyroscope. The program finds the average of fifty gyro_z readings and sets that to the global variable `bias`. Note that calibrate_gyro only needs to be called once at the beginning of the program. The robot must be completely still when it is called or the bias value will not be accurate. The robot must also be warmed up since it was last turned on (this can be done by briefly running the motors or by

simply waiting for a few minutes after turning it on). If the robot is not warmed up before finding the bias, the bias will likely change once the robot starts moving.

C. Finding the Relative Angle with the Gyroscope

Once the bias is accounted for, the relative angle of the Wallaby can be found by converting angular velocity to angular position. The formula to find angular position from angular velocity, when angular acceleration is constant, is angular velocity times time. If it is assumed that angular acceleration is constant over 10ms then a variable `theta` can be updated with the code shown below.

```
double theta = 0;
while(1 == 1)
{
    msleep(10);
    theta += (gyro_z() - bias) * 10;
}
```

This code constantly updates the value of `theta` by integrating the graph of the angular velocity with respect to time by taking a right hand Riemann sum of the velocity graph over 10ms intervals, which will convert the angular velocity to angular position. The units for this method of finding theta typically translate to about 560000 KIPR Degrees to every 90 standard degrees, but this value will be different depending on the specific Wallaby and the Wallaby's angle to the ground.

D. Turning with the Gyroscope

After the challenge of tracking the relative angular position of the robot has been met, it is a small step to be able to turn using the gyroscope. One way of doing this is shown below.

```
void turn_with_gyro(int left_wheel_speed, int right_wheel_speed, double targetTheta)
{
    double theta = 0;//declares the variable that stores the current degrees
    mav(right_motor, right_wheel_speed);//starts the motors
    mav(left_motor, left_wheel_speed);
    //keeps the motors running until the robot reaches the desired angle
    while(theta < targetTheta)
    {
        msleep(10);//turns for .01 seconds
        theta += abs(gyro_z() - bias) * 10;//finds the positive change of theta over the 10ms
    }
    //stops the motors after reaching the turn
    mav(right_motor, 0);
    mav(left_motor, 0);
}
```

This code sets the motors to the desired speeds and lets them run until the Wallaby's angular position, represented by `theta`, is greater than or equal to the target theta. While there are many different ways to use the gyroscope to effectively turn, this method is not only simple but also works very well at slow speeds and allows for arcing because it runs the motors until the Wallaby itself has turned to the target theta, regardless of what the rest of the robot does.

E. Driving Straight Using the Gyroscope

Driving straight with the gyroscope can be done in several different ways, but they all involve creating a variable to track how much the angular position has changed and then keeping that variable as close to 0 as possible. A basic example of this is shown below.

```
void simple_drive_with_gyro(int speed, double time)
{
    double startTime = seconds(); //used to keep track of time
    double theta = 0; //keeps track of how much the robot has turned
    while(seconds() - startTime < time)
    {
        //if the robot is essentially straight then just drive straight
        if(theta < 1000 && theta > -1000)
        {
            mav(right_motor, speed);
            mav(left_motor, speed);
        }
        //if the robot is off to the right then drift to the left
        else if(theta < 1000)
        {
            mav(right_motor, speed + 100);
            mav(left_motor, speed - 100);
        }
        //if the robot is off to the left then drift to the right
        else
        {
            mav(right_motor, speed - 100);
            mav(left_motor, speed + 100);
        }
        msleep(10);
        theta += (gyro_z() - bias) * 10;
    }
}
```

This function checks if the Wallaby's relative angle has moved to the left or to the right and makes a hardcoded decision based off of its findings. However, as a result of the simplicity of this function, it attempts to be a "one size fits all" solution and the robot will often get off track once the function ends. This is only one of many methods of driving straight. At Norman Advanced, we replaced the simple_drive_with_gyro function with something very similar to this:

```
void drive_with_gyro(int speed, double time)
{
    double startTime = seconds();
    double theta = 0;
    while(seconds() - startTime < (time / 1000.0))
    {
        if(speed > 0)
        {
            mav(right_motor, (double)(speed - speed * (1.920137e-16 + 0.000004470956*theta -
7.399285e-28*pow(theta, 2) - 2.054177e-18*pow(theta, 3) + 1.3145e-40*pow(theta, 4))));

            mav(left_motor, (double)(speed + speed * (1.920137e-16 + 0.000004470956*theta -
7.399285e-28*pow(theta, 2) - 2.054177e-18*pow(theta, 3) + 1.3145e-40*pow(theta, 4))));
        }
        else//reverses corrections if it is going backwards
        {
            mav(right_motor, (double)(speed + speed * (1.920137e-16 + 0.000004470956*theta -
7.399285e-28*pow(theta, 2) - 2.054177e-18*pow(theta, 3) + 1.3145e-40*pow(theta, 4))));

            mav(left_motor, (double)(speed - speed * (1.920137e-16 + 0.000004470956*theta -
7.399285e-28*pow(theta, 2) - 2.054177e-18*pow(theta, 3) + 1.3145e-40*pow(theta, 4))));
        }
        msleep(10);
        theta += (gyro_z() - bias) * 10;
    }
}
```

Here, we created a quartic function to continuously update the speeds of each motor based on the angular offset of the Wallaby. However, this function is still attempting to do the same thing as simple_drive_with_gyro, just in a more complicated way.

4. Conclusion

The inclusion of the gyroscope in the Wallaby has provided teams with a fundamentally simpler and more consistent solution for reliable navigation. As shown in this paper, with just a few lines of code and basic programming knowledge, the gyroscope can allow any team to increase the consistency of their robots to the level of some of the most complex libraries that only the most advanced teams in Botball have been able to create in the past.

5. References

1. "How a Gyro Works." *Gyroscope*, learn.sparkfun.com/tutorials/gyroscope/how-a-gyro-works.
2. Weinberg Download PDF, Harvey. "Gyro Mechanical Performance: The Most Important Parameter." *LTC3786: High Efficiency Li-Ion Battery-to-USB Boost Converter | Analog Devices*, www.analog.com/en/technical-articles/gyro-mechanical-performance.html.