# CPRE 1850 – CONDITIONALS

## LAB 4
## SECTION CN

**SUBMITTED BY:**

**MAXWELL MILLER**

**10/3/2025**

**Problem**

The problem presented in this lab has primary to do with using vector accelerations. The goal is to use both the corrected and raw gravitational accelerations to determine the position of the Dual Shock controller in a variety of positions.

**Analysis**

The lab requires the use of functional abstraction design to make a program that is useable. The lab instructions also indicate that the use of a tolerance will be necessary, and the reuse of the magnitude function from the third lab is necessary to calculate motion.

**Design**

When designing my implementation of this program, I decided to first get my motion detection function - *in_motion* - working first. But first I needed a working vector magnitude function to process *ax*, *ay*, and *az*, so I copied over my magnitude function from lab 3. Then, I modified the *lab4.c* file, so it printed the magnitude from *ds4rd.exe*, then I moved around the controller and then held it still in a variety of orientations. Given this data, I settled on a still tolerance parameter of ±5%. Using that in combination with my *close_to* method I was able to make the *in_motion* function work. I then tested it for a while and fine-tuned the parameters.

Next, I turned my focus to the direction detection functionality. I chose to implement a function for each that detected if the controller was facing that direction - *is_top*, *is_bot*, *is_right*, *is_left*, *is_front*, and is_back. Each of these functions contains thresholds for all three axis, and ideal positions for each one. For these functions, however, the gravitational adjusted acceleration is used instead. Additionally, each function contains a "DERTMINANING TOLERENCE" comment, which indicates the primary direction that indicates the axis that the controller is facing. The other tolerances are larger to allow for more margin of error when determining the direction. I decided to go with this design after struggling to align the controller, so I made a universal structure for the functions with these constants in it.

In order to determine the constants for each of the direction functions, I ran the *lab4.c* file with a *printf* statement to print out each of the raw accelerations. Once I found which acceleration what the one approaching one, and which were close to zero, I determined the constants for each direction. After testing I found a determining tolerance of ±25%, and a non-determining tolerance of ±40%.

Once all of my directional functions were working, I then put it all together with an else-if chain, starting with confirming that there isn't any motion, then checking for each direction. I

then implemented a *Direction* enumeration that allows me to track what the last detected orientation was, allowing me to prevent repeat outputs.

**Testing**

The most difficult things were determining if the code is functional and determining the correct tolerance constants. The problem was that those two things aren't mutually independent, and the wrong idea in the code with the correct tolerance may appear as if the tolerance is actually the problem. So, in order to stem that issue, I built a version of the file with all of the state handling commented out, so I could determine the proper tolerances.

**Comments**

This is likely not the most efficient way to design this, and the tolerances for direction seem quite large, it seems that they are necessary to get the program to work as expected. Additionally, the program uses two different acceleration vectors, gravitational and raw. Gravitational accelerations (*gx*, *gy*, & *gz*) are used in determining direction, while the raw accelerations (*ax*, *ay*, & *az*) are used in determining motion.

```
/*----------------------------------------------------------------------------
                            SE/CprE 185 Lab 04
                Developed for 185-Rursch by T.Tran and K.Wang
----------------------------------------------------------------------------*/

/*----------------------------------------------------------------------------
                                 Includes
----------------------------------------------------------------------------*/
#include <stdio.h>
#include <math.h>


/*----------------------------------------------------------------------------
                                 Defines
----------------------------------------------------------------------------*/

enum Direction {
      TOP,
      BOTTOM,
      RIGHT,
      LEFT,
      FRONT,
      BACK,
      NONE
};


/*----------------------------------------------------------------------------
                               Prototypes
----------------------------------------------------------------------------*/
double mag(double ax, double ay, double az);
int close_to(double tolerance, double point, double value);
int in_motion(double ax, double ay, double az);

int is_top(double gx, double gy, double gz);
int is_bot(double gx, double gy, double gz);
int is_right(double gx, double gy, double gz);
int is_left(double gx, double gy, double gz);
int is_front(double gx, double gy, double gz);
int is_back(double gx, double gy, double gz);
```

Continued on the next page.

```
/*------------------------------------------------------------------------
                              Implementation
------------------------------------------------------------------------*/
int main(void) {
      int t, b1, b2, b3, b4;
      double ax, ay, az, gx, gy, gz;

      enum Direction lastDirection = NONE;

      while (1) {
            scanf("%d, %lf, %lf, %lf, %lf, %lf, %lf, %d, %d, %d, %d",

               &t, &ax, &ay, &az, &gx, &gy, &gz, &b1, &b2, &b3, &b4);

            int motion = in_motion(ax, ay, az);

            if (in_motion(ax, ay, az)) { // DO NOTHING  }
            else if (is_top(gx, gy, gz) && lastDirection != TOP) {
                  lastDirection = TOP;
                  printf("TOP\n");
            }
            else if (is_bot(gx, gy, gz) && lastDirection != BOTTOM) {
                  lastDirection = BOTTOM;
                  printf("BOTTOM\n");
            }
            else if (is_right(gx, gy, gz) && lastDirection != RIGHT) {
                  lastDirection = RIGHT;
                  printf("RIGHT\n");
            }
            else if (is_left(gx, gy, gz) && lastDirection != LEFT) {
                  lastDirection = LEFT;
                  printf("LEFT\n");
            }
            else if (is_front(gx, gy, gz) && lastDirection != FRONT) {
                  lastDirection = FRONT;
                  printf("FRONT\n");
            }
            else if (is_back(gx, gy, gz) && lastDirection != BACK) {
                  lastDirection = BACK;
                  printf("BACK\n");
            }
            // If Triangle button is pressed, end the program
            if (b1) { break; }

      }
      return 0;
}
```

Continued on the next page.

```
double mag(double ax, double ay, double az) {
      // returns the vector acceleration magnitude
      // ie. sqrt(a^2 + b^2 + c^2)
      return sqrt( ( ax * ax ) + ( ay * ay ) + ( az * az ) );
}

int close_to(double tolerance, double point, double value) {
      return fabs(point - value) <= tolerance;
}

int in_motion(double ax, double ay, double az) {
      const double MOTION_TOL = 0.05;

      double aMag = mag(ax, ay, az);

      return !close_to(MOTION_TOL, 0.0, aMag);
}


int is_top(double gx, double gy, double gz) {
      const double X_TOL = 0.4;
      const double Y_TOL = 0.25;  // DERTMINANING TOLERENCE
      const double Z_TOL = 0.4;

      const double TARGET_GX = 0.035;
      const double TARGET_GY = 0.98;
      const double TARGET_GZ = 0.199;

      int a = close_to(X_TOL, TARGET_GX, gx);
      int b = close_to(Y_TOL, TARGET_GY, gy);
      int c = close_to(Z_TOL, TARGET_GZ, gz);

      return a && b && c;
}
```

Continued on the next page.

```
int is_bot(double gx, double gy, double gz) {
      const double X_TOL = 0.4;
      const double Y_TOL = 0.25;  // DERTMINANING TOLERENCE
      const double Z_TOL = 0.4;

      const double TARGET_GX = 0.0;
      const double TARGET_GY = -1.0;
      const double TARGET_GZ = 0.0;

      int a = close_to(X_TOL, TARGET_GX, gx);
      int b = close_to(Y_TOL, TARGET_GY, gy);
      int c = close_to(Z_TOL, TARGET_GZ, gz);

      return a && b && c;

}

int is_right(double gx, double gy, double gz) {
      const double X_TOL = 0.25;  // DERTMINANING TOLERENCE
      const double Y_TOL = 0.4;
      const double Z_TOL = 0.4;

      const double TARGET_GX = -1.0;
      const double TARGET_GY = 0.0;
      const double TARGET_GZ = 0.0;

      int a = close_to(X_TOL, TARGET_GX, gx);
      int b = close_to(Y_TOL, TARGET_GY, gy);
      int c = close_to(Z_TOL, TARGET_GZ, gz);

      return a && b && c;
}
```

Continued on the next page.

```
int is_left(double gx, double gy, double gz) {
      const double X_TOL = 0.25;  // DERTMINANING TOLERENCE
      const double Y_TOL = 0.4;
      const double Z_TOL = 0.4;

      const double TARGET_GX = 1.0;
      const double TARGET_GY = 0.0;
      const double TARGET_GZ = 0.0;

      int a = close_to(X_TOL, TARGET_GX, gx);
      int b = close_to(Y_TOL, TARGET_GY, gy);
      int c = close_to(Z_TOL, TARGET_GZ, gz);

      return a && b && c;
}

int is_front(double gx, double gy, double gz) {
      const double X_TOL = 0.4;
      const double Y_TOL = 0.4;
      const double Z_TOL = 0.25;  // DERTMINANING TOLERENCE

      const double TARGET_GX = 0.0;
      const double TARGET_GY = 0.0;
      const double TARGET_GZ = -1.0;

      int a = close_to(X_TOL, TARGET_GX, gx);
      int b = close_to(Y_TOL, TARGET_GY, gy);
      int c = close_to(Z_TOL, TARGET_GZ, gz);

      return a && b && c;
}

int is_back(double gx, double gy, double gz) {
      const double X_TOL = 0.4;
      const double Y_TOL = 0.4;
      const double Z_TOL = 0.25; // DERTMINANING TOLERENCE

      const double TARGET_GX = 0.0;
      const double TARGET_GY = 0.0;
      const double TARGET_GZ = 1.0;

      int a = close_to(X_TOL, TARGET_GX, gx);
      int b = close_to(Y_TOL, TARGET_GY, gy);
      int c = close_to(Z_TOL, TARGET_GZ, gz);

      return a && b && c;
}
```