# Robotics With the XBC Controller

Session 5

Instructor: David Culp

Email: culpd@cfbisd.edu

# Learning Goals

- The student will learn to recognize and implement proportional and bang-bang control methods. The student will learn the difference between a servo motor and a standard DC motor and be able to use a servo motor in Interactive C. In addition the student will learn about gears and drive trains.

# Simulating Sensor Input With the IC Simulator

- See visual demonstration.

# Servos

- Servos are simply DC motors that have control circuitry so they can be precisely positioned **to hold an angle (not rotate)**.
- Have three wires with a female plug. **(opposite to the motor and sensor plugs).**
- Plug into the XBC's servo ports(0-3).
- Make sure the black wire is in line with the terminal labeled "-"

# Using Servos…

- Extremely easy to use **to rotate an arm or open/close a grabber**.
- Activate all servos with the function enable_servos()
  - Only do this once
  - DO NOT USE IN A LOOP
- Deactivate servos with the disable_servos()

# Positioning servos

- Use the set_servo_position function to move a servo to a desired position.
  - int set_servo_position(int srv, int pos)
    - srv = Servo port number 0-3
    - pos= position(10-245) **is ~0-180 degrees**
- You can retrieve a servos current position with the get_servo_position function.
  - int get_servo_position(int srv)
    - Returns the current position of the servo on port# srv

# Using xbctest.ic

- Follow on screen demonstration to load and run “xbctest.ic” to test servos and other XBC hardware.

# Using Servos

- Plug a servo into port #0
- Open the interaction window
- Make certain the XBC is turned on.
- Issue the enable_servos() function call.
  - **Servos move to an *arbitrary* position**
- **Caution: Because servos are not accurately calibrated, position values < 10 or > 245 can drive a servo into its mechanical stop, draining the battery and over-heating the servo. Test motion before trying these extremes.**
- Now issue the command set_servo_position(0,10)
- Now issue the command set_servo_position(0,245)
- Set the servo to various positions and try the get_servo_position function.

# A Short Assignment

- Write a short program to slowly move a servo from position 10 to position 245.
- You will need a for-next loop.

# To Help You Get Started

- Remember a for-next loop looks like the following:
  - for (<*expr-1*>;<*expr-2*>;<*expr-3*>) <*statement*>
  - for (count = 100; count >=10; count-=10)
- Use a #define to define the servo port #
  - #define SERVO 0

# Starting the Program

```
#define SERVO 0
void main()
{
    int position;

    enable_servos();

   Your for-next-loop and servo control goes
   here

}
```

# Solution

```
#define SERVO 0
void main()
{
    int position;  // holds our current servo position

    enable_servos();  // We must turn the servos on first

    for (position = 10; position< 245; position +=5)
      {  // loop from 10 to 245 in increments of 5
        set_servo_position(SERVO,position);  // set servo 0 to position
        display_clear();
        printf("Position: %d", position);
        sleep(0.1);  // A slight pause or we go to fast
      }
}
```

# Bang-Bang Vs. Proportional Control

- “Bang-bang” control uses on/off extremes.
  - This is what we have been using.
  - If the sensor is “hit” do something.
  - If the sensor is not “hit” do something else.
  - No in between.

# Proportional Control

- Proportional control adjusts the output or response of the robot as the sensor readings change.
- In other words the sensor provides a **smoother *proportional*** feedback mechanism into the control loop.

# Getting Proportional Control From an IR Range Finder

- We need to pass the output of the IR sensor to the motor function.
- IR sensor returns 0-255. We must scale this to 0-100.
  - Sensor_Value/2.55
- The values are reversed. Large values indicate a very close object so we want to go slowly.
  - -(Sensor_Value/2.55)+100

# Mixed Math and Casting

- We cannot arbitrarily mix types in C.
  - Some languages allow this.
  - This slows those languages down because of overhead.
- We must “cast” the values being used to the same type.
  - Place the type you want to cast to in parentheses just before the value or variable you want to cast.
    - (int) 3.14159
    - (float) speed
    - (char) 3.14159

# Our Formula using Casting

- speed= -(int)( (float)analog(0)/2.55) + 100;
  - (float)analog(0) casts the return value of the analog function to a float.
    - This is so we can divide it by 2.55
  - The external (int) casts everything back to an int type so 100 can be added to it and assigned to the int type variable speed.

# An Example

```
void main()
{
    float MAX_IR = 1.5 // sensor max is actually ~150
    int speed;  // holds our speed
    while(!b_button())  // loop until b button pressed
      {
        speed= -(int)( (float)analog(0)/MAX_IR) + 100;
        motor(0,speed);
        motor(2,speed);
        sleep(0.15);
    }
    printf("DONE!!\n");
}
```

# Proportional Control

- In general proportional control allows greater control and a better range of response from your robot.
- Sometimes harder to implement.
  - Can lead to more elegant solutions.
- We will implement proportional control later in tonight's challenge.

# Gears and Drive Trains…

- Gears serve to provide a gear reduction in a mechanical object.
  - Gears are used to change the speed and force of the motor.
  - Any rotating object is spinning at a certain velocity and with a specific strength, in this case called torque.
  - We can change either quantity by introducing a new "gear ratio" into the drive system.
    - If we decrease the rate of spin we increase the torque of the output gear.
    - If we increase the velocity of the output gear we subsequently decrease the torque, or strength in which the output gear spins.

# Gears and Drive Trains

- Gears serve to change the direction of rotation in a mechanical object.
  - Each gear in the system “reverses” the direction of spin.
- We can easily calculate the **mechanical advantage** of the gear train.
  - MA = output teeth/input teeth.

# Example.....

- Picture shows a 24 tooth input gear and a 40 tooth output gear.
- Output teeth/Input teeth
  - 40:24 = 5:3
  - Each time the input spins 5 times the output has spun 3 times.

# Multiple Gears in a train…

- Multiple gears in a drive train are called a **compound gear train.**
- We take the product of the gear ratios**.**
- Example: We have a 3 tooth gear driving a 9 tooth gear connected to a shaft that drives a 3 tooth gear that drives another 9 tooth gear. What is our gear ratio?
  - We have two separate gear ratios, a 3:1 gear ratio, driving another 3: 1 gear ratio. This makes a total gear ratio of 9:1.

# Calculating RPM

- You have a DC motor which spins at 1000 RPM that drives a 4-tooth gear; this gear in turn drives a 64 tooth gear.  At what speed does the output gear spin?
  - Gear ratio = 64:4 = 16:1.
  - *Each time our 4 tooth driven wheel is rotated one complete turn the output wheel has only made it around 1/16 of a turn.*
  - 1000 RPM /16 = 62.5 RPM.

# A Problem for You to Solve

- *You have a DC motor that spins at 25 RPM. The motor drives an output shaft connected to a 25 tooth gear, that gear drives a 5 tooth gear. The 5 tooth gear is connected to a shaft upon which a 9 tooth gear is connected and driving a 3 tooth gear. What is the total gear ratio and at what RPM will the output shaft spin?*

# Solution…

- ratio 1 = 5 teeth/ 25 teeth = 1:5
- ratio 2 = 3 teeth/ 9 teeth = 1:3
- Now we multiply the ratios together and we get a total gear ratio of 1:15
- Our output shaft spins 15 times faster or at 375 RPM.

# Homework…

- Gear ratio problems posted on the course forum by Thursday.

# Tonight's Challenge

- Modify your robots code so that it uses proportional control.
- The input from the IR sensors should be used.
  - Process the data from the IR sensors and input that into a control loop.
  - Lower numbers result in higher speeds to that motor (we have already done this part).
  - Each motor should be controlled separately. The output of the **LEFT** sensor controls the **RIGHT** motor etc…
- Behavior = The robot should avoid obstacles.
  - As the left sensor gets close to an object the right wheel is slowed and the left wheel is sped up, turning the robot right.
  - Opposite occurs when the right sensor approaches and object.

```
#define LEFT_MOTOR 2 //left motor plugged into port 2
#define RIGHT_MOTOR 0 //Right motor plugged into port 0

#define LS 0  // Left IR sensor plugged into port 0
#define RS 1   // Right IR sensor plugged into port 0

void main()
{
   int left_speed;  // holds our speed
   int right_speed;

   while(!b_button())  // loop until b button pressed
     {

       /* The two lines below assign the output of the analog sensors
to variables that control the motors speed.
Since the sensors return a value from 0-255 we must scale this value
to 0-100 by dividing by 2.55.

Since the IR sensors return large values for close objects we must make them negative
and add 100 to get the desired range of 0-100.
*/
       left_speed= -(int)( (float)analog(RS)/2.55) + 100;
       right_speed= -(int)( (float)analog(LS)/2.55) + 100;

       motor(LEFT_MOTOR,left_speed);
       motor(RIGHT_MOTOR, right_speed);
   }
   printf("DONE!!\n");
}
```