2015 BOEBot Code Template File Analysis

The source code or Arduino "sketch" given to you in the file BOEBot2015.zip provides the following functionality:

- A complete real-time control loop
- Wheel encoder reader, odometry calculations
- Wheel servo routines
- Go-to-goal PID controller
- Ultrasonic Rangefinder routines
- Infrared beacon decoding routines
- Sensor turret scanning routine

In addition to the sketch itself, a custom library called BotPing is part of the package. It is in the file BotPing.zip. It is a modified version of the NewPing library that allows the Infrared Receiver to share the same interrupt service routine.

As-is, the template provides *go-to-goal* functionality. Using dead-reckoning (odometry) the robot will attempt to visit waypoints compiled into the program from the file called robot_goals.h. The format is a list of tuples as follows:

```
//  x      y      speed  error  end-flag
{  0.0,  12.0,    60,   2.5,   0},
{-12.0,  24.0,    40,   3.0,   1},
```

Each set of numbers in curly brackets represents a waypoint. The robot will visit each waypoint in turn, and after the waypoint with a "1" in the end-flag field, it will stop. The coordinate system of the robot is based upon its initial position which is at position x,y equal to (0,0). The positive y-axis is the direction the robot is initially facing, and the positive x-axis is to the right. The x,y units in the table is inches. The speed parameter is percentage of full power that will be applied to the wheels, and can vary from 0 to 100%. However, you have to reserve power in order to execute turns properly, so 100% is not a viable number to use. The error parameter is the distance in inches from the goal the robot is considered to have reached to goal. Due to accumulating drift, the error value should increase in value from waypoint to waypoint.

Other than *go-to-goal*, no other robot behavior is implemented. It is up to you to add blend in behaviors based upon the sensor inputs.

You may have to add new features to the robot that will extend the go_to_table. For example, in a race you might want to robot to be able to navigate backward. The forward/backward flag can be added to the table.

Important global variables

Name	Type	Description
WCount_Left	long	Left wheel encoder raw tick count
WCount_Right	long	Right wheel encoder raw tick count
bot_x	double	Absolute x coordinate of current bot pose (inches)
bot_y	double	Absolute y coordinate of current bot pose (inches)
bot_sin	double	sin(theta) where theta is the angle of the current bot pose
bot_cos	double	cos(theta) where theta is the angle of the current bot pose
go_to_x	double	X coordinate of goal as set by go_to_goal() function
go_to_y	double	Y coordinate of goal as set by go_to_goal() function
go_to_speed	int	Desired speed in driving to the current goal as set by go_to_goal() function
bot_heading	double	Angle of bot pose in radians.
go_to_tolerance	double	If the bot is less than this distance to the goal, it is considered to have reached the goal.
go_to_done	char	If the bot has reached its goal, this is set to 1 in the go_to_goal_PID() function. It is cleared to 0 by the template code whenever a new goal is assigned.
ping_in_progress	char	This flag is set whenever a sonar ping is sent. It is cleared when an echo is received,
us_ping_result	unsigned long	The time in microseconds it took the echo to return. Divide by US_ROUNDTRIP_IN to get the distance in inches.
isr_ir_available	char	Flag set to 1 when an IR frame was received. It is cleared by the ISR at the beginning of every frame. Clear this flag whenever you read the isr_ir_value. Currently the template code polls this flag in the 100 ms section of the loop. You might consider moving it to the 10 ms section since IR frames may arrive at 25 ms intervals.
isr_ir_value	char	This is the numeric value of the IR frame received. This value is cleared at the beginning of a received frame.

BotPlot and Debug Serial Output

The template code can be configured to periodically output information for the BotPlot graphical display, or a human-readable status line. Examine the commented out code in the main control loop.

The go_to_goal_PID() Function

The go_to_goal_PID() function found near end of the template is called once every loop() iteration which is every 10 ms. The function serves as the controller of the robot. It takes the current bot pose, desired speed and goal position and adjusts the left and right servo power using the set_speeds_calibrated() function as necessary to keep the bot headed toward its goal. A full PID controller would probably be superfluous, but you can tune the Ki and Kd to non-zero values. For more background information, you can view the Coursera videos 1.6 to 1.8.

Main Loop Structure

```
Loop() {

Poll_encoders(); // Check wheel encoders and update robot pose
go_to_goal_PID(); // Perform PID control to target next waypoint
if (go_to_done){} // If next waypoint is reached, bump to next
if (loop_counter_n == EVERY_N_LOOPS) { // do only 10 times a second
   // print turret angle, echo distance and IR code
   // initiate next ping
   // Commented out: debug/telemetry print only 2x a second
   // Turn turret servo
}
// Do other stuff to stretch loop interval to 10 ms.

}
```

Some things to think about

1. Where in the loop would you insert code to implement other behaviors?
2. How would you "pause" the robot?
3. How would you implement obstacle avoidance? Dynamic waypoint insertion?
4. How would you modify the go_to_goal_PID function to allow navigating in reverse?
5. Which PID coefficient would you make non-zero to reduce errors in the current P-only implementation?
6. What happens if Kp is too large? If it is too small?
7. How would you pause the sensor turret so you can take distance and beacon readings in a fixed direction?
8. The odometry calculation is independent of the go_to_goal PID calculation. How would you temporarily disable the PID controller to have the robot temporarily move under the control of some other sensor-based function?