The Hardware Abstraction Layer
Ethan Y. Myers
Wesley Y. Myers
Cedar Brook Academy

The Hardware Abstraction Layer

1. Introduction

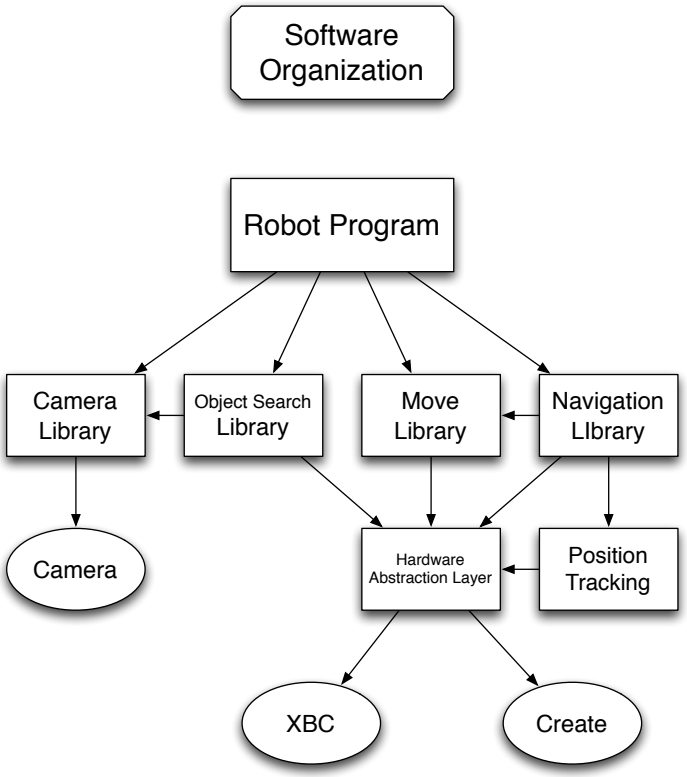
The Hardware Abstraction Layer (HAL) is a layer of software that hides the differences in hardware from the software above it. It allows the same code to be run on different types of hardware. In the 2007 Botball game [1], there was only one type of hardware, the XBC [7]. In the 2008 Botball game [2], a whole new platform was introduced, the Create [3]. The Create came with its own motors and sensors, making the Create a platform. The addition of the Create meant that teams now have two different platforms to deal with. From a software perspective, these platforms have different interfaces for their movement functions.

This is Cedar Brook Academy's (CBA) fifth year in Botball. Over the course of the years, we have developed extensive code libraries. We are able to reuse and further develop our code every year. The addition of the Create meant that all of the functions that dealt with moving the robot would have to be changed because the XBC and Create have different motor commands. This is exacerbated by the fact that our movement functions heavily use the XBC's Back Electromotive Force (BEMF) [7], and the Create does not have an equivalent mechanism. We did not want to maintain two separate code bases, so we decided to abstract the movement commands by creating a hardware abstraction layer (HAL). The HAL handles the differences in the movement commands. The HAL allows us to maintain one set of code for two separate platforms. By doing this, the HAL allows us to still be able to use our code with the Create.

The HAL is actually two separate libraries, a library for the XBC, and a separate library for the Create. These two libraries hold common function names, but within each function, the code is specific to the hardware. For example, when calling a motor function for the Create, the code will call the HAL function and use create_drive_direct, while calling a motor function for the XBC will use move_at_velocity (mav). The calling function will use the same call in both instances. This is the basic idea of how the HAL works.

2. CBA's Software Libraries

Our programming is simplified by the use of extensive software libraries that we have constructed during our previous Botball competitions. These libraries provide functions such as position tracking, navigation, movement commands, object searching, and camera commands. The five main libraries of software are the Position Library, the Navigation Library, the Move Library, the Object Search Library, and the Camera Library. All of these libraries, except for the Camera Library, directly or indirectly use the HAL.



2.1 Position Library

The position tracking library [6] tracks the position and direction of the bot as it moves throughout the board. It does this by monitoring the wheel encoders on the bot. It assumes that the encoder variables are directional, which the BEMF provides. By using the encoders, the software is able to determine how far the robot has gone and what direction it is pointing in. However, it is necessary to set the robot's initial position and direction at the beginning by calling set_position. The software can determine the robot's position by calling get_position, which provides x-y coordinates and the Bot's heading. The software also can update the robot's position in the middle of a match by calling set_position. Doing this allows the software to adjust for errors that build up during the course of the match.

2.2 Navigation Library

The navigation library [4] uses the position library and movement library. In this library, there is a set of functions that move the bot to a desired position or direction. The most used function in the navigation library is go_to_point. This function tells the bot to go to a specific x-y coordinate on the board. Another important function in this library is rotate. With this function, the software is able to tell the robot to turn to a specific direction.

2.3 Movement Library

The movement library holds the more basic movement functions. The major function in move_lib is go_straight. With this function, we are able to move the robot straight in either the forward direction or backward direction at any speed. Go_straight also monitors the sensors and informs the caller, so that an action can be taken.

2.4 Object Search Library

The object search library [5] contains functions that move the bot to an object based on one or two colors. It uses the camera to find and go to the object. The major function in the object search library is go_to_object, which moves the robot to a colored object. Another important function is center_object, which centers the robot on an object based on the color.

Another useful function, go_to_target, is similar to go_to_object except that it moves the robot to objects that are composed of two colors.

2.5 Camera Library

The camera library is not like the other libraries in that, it is not as large as the others. The camera library contains camera control functions. The camera library essentially prepares the camera for use. It sets the various camera parameters such as white balance and reads the camera when it is ready.

3.0 Differences in the XBC and Create Movement Function Calls

The Create has many unique calls for itself. Around a fourth of those calls deal with movement. For each of the movement functions for the Create, there is a similar function for the XBC, though it may not be exactly the same. The Create call, create_drive_straight, is the same as mav for the XBC, except that create_drive_straight controls both motors, while mav controls only one motor. This is the main difference in the function calls. The Create also has additional movement calls such as create_spin_CW, create_spin_CCW, and create_drive_direct. Create_drive_direct allows each motor to be assigned a different speed. This function is the primary Create function used by the HAL_create_lib. These all can be simulated on the XBC use the mav function. Both platforms use a motor command with zero speed to stop their motors.

3.1 Differences Between the Create Raw Encoders and XBC BEMF

The XBC uses what is called Back EMF, which provides the direction and total distance a motor has moved. In the XBC, get_motor_position_counter can be called to get the position of a wheel. Using this, you can calculate the distance the wheel has traveled. The XBC also tells the direction the wheel has traveled, i.e. forwards or backwards, by adding to the counter when moving forward, and subtracting from the counter when moving backwards. Determining direction is a matter of computing the difference between two subsequent returns of get_motor_position.

The Create raw encoder counter[1] is a summation of encoder tics. In other words, it is a count of the number of times the encoder is tripped. Since it only counts up, the counter cannot provide direction information. Many of our movement functions are dependent upon BEMF directional properties, so we had to come up with a solution to map the Create encoders to the BEMF equivalent.

Another major difference between the two counters is the number of bits in each counter. The XBC uses 32-bit counters, which is unlikely to roll over during a match. However, the Create only uses 16-bit counters. Our tests indicate that this counter can roll over several times in a match. When the Create counter rolls over, it goes from 65535 to 0, and then starts to count

[1] The Create's raw encoders can be accessed by calling get_raw_encoders, which is found in create_lib.ic.

up again. This was an obstacle because our code did not account for roll over because we were using the XBC, so the HAL had to be coded to deal with this.

3.2 Emulating Back EMF in the Create HAL

The two major barriers with simulating the Back EMF in the Create was that, it does not determine direction, and secondly, it rolls over quickly. Since we could not simply use the Create counters by themselves, we had to create two 32-bit variables, right_position and left_position. These two variables hold the position of the right motor and left motor respectively.

To simulate the BEMF counter directionality, directional state for each wheel must be maintained. Depending upon the directional state, this value is either added or subtracted. By creating two variables called right_motor_state and left_motor_state, we are able to maintain the direction the motor is going. These variables have three possible states: forward, backward, or stopped. The major issue is change of direction. To maintain accuracy, the motors must be completely stopped before changing direction. Otherwise, the encoder values may be incorrectly tabulated because of drift

To handle the roll over, we simply have to do the math. When updating the right_position and left_position, we first calculate the delta for each motor and add it to the position. Because the counters always count up, the current encoder value is always greater than the previous encoder value, unless it rolls over. At the roll over point, the new encoder value will be less than the previous. When this occurs, two distances are calculated and then added. The distance it took to get to 65536 or 0 is calculated. Then that distance is added to the current encoder value to get the total delta, which is then added to the appropriate counter.

4.0 Functions in the CBA HAL

In our HAL library, only the movement functions are abstracted. Move_robot and stop_robot provide the abstract movement functions, and get_wheel_position, generate_position, and clear_wheel_position provide the abstract BEMF information. The abstract movement functions are easy to do for the XBC since they are a direct translation. Additional work had to be done in the equivalent create HAL functions because the directional information had to be accounted for in the movement functions.

4.1 Move_robot

The abstract move_robot function is easily implemental using mav in the HAL_XBC. The HAL_create move_robot function requires more effort. The motor command is implemental using create_drive_direct. However, the BEMF emulation requires the HAL_create library to detect when the motor direction is reversed. In this instance, the motor has to be stopped (using stop_robot) before the direction state is changed. Only after the robot is stopped, will the new power be applied to the motors.

4.2 Stop_robot

For the stop_robot function, we use the appropriate set of commands to bring the robot to a full stop. From our experience in Botball, we know that powering the motors to off does not prevent the robot from drifting. Drift can be a huge problem when precision is needed. For the HAL_XBC lib the freeze command is used. For the HAL_create library, the motors are reversed using a low power. In order to do this, the motor state must be used to determine the reverse direction. Then the reverse power is maintained until the robot is stopped, which is determined when the delta encoder values become zero.

4.3 Get_wheel_position

For the get_wheel_position function, we use the get_motor_position_counter function for the XBC. The get_motor_position_counter returns the encoder value for the motor specified. With the Create, the right_position or left_position value will be returned to the caller. As stated in section 3.2, these two values hold a running value for the encoders for the right and left motors.

4.4 Clear_wheel_position

Clear_wheel_position is generally only used during the beginning of the match. This command sets the encoder values to 0. In the XBC, this command is clear_motor_position_counter. While in the HAL Create library, it is defined as setting right_position and left_position as 0.

4.5 Generate_position

The generate_position function is an internal function of the HAL_create library. It is started as a background process during the initialization phase. This process emulates the BEMF of the XBC. It periodically checks the Create's raw encoder values and compares them with the previous encoder values to generate right_position and left_position. The conversion process is discussed in section 3.2.

5. Summary

This paper has discussed the notion of a Hardware Abstraction Layer. The HAL insulates the software from hardware differences. We have discussed mainly movement functions, but the HAL can be applied to other functions as well, such as sensor differences. For example, the Create has its own bump sensor. Interfacing with this bump sensor will be different from interfacing with another touch sensor. The HAL can hide these differences. Therefore, HAL is extremely useful for software reuse.

6. References

[1] Anonymous, “Botball 2007 Teacher’s Workshop Game Review,” KIPR, 2007.

[2] Anonymous, “Botball 2008 Teacher’s Workshop Game Review,” KIPR, 2008.

[3] Anonymous, “iRobot Create Open Interface,” iRobot Corporation, 2007.

[4] Myers, Ethan, Myers, Wesley, “Navigation Using Position Tracking,” 2007 National Conference on Education Robotics 10-13 July 2007.

[5] Myers, Ethan, Myers, Wesley, “The Object Search Library,” 2007 National Conference on Education Robotics, 10-13 July 2007.

[6] Myers, Wesley, Myers, Ethan, "Position Tracking Using the XBC," Proceedings of the 5th Annual Conference on Educational Robotics, 7-10 July 2006, 157-163.

[7] R. LeGrand, K. Machulis, D. Miller, R. Sargent and A.Wright, “The XBC: a Modern Low-Cost Mobile Robot Controller,” *Proceeding of IROS 2005*, IEEE Press, 2005.