Rudder Controller & Wiper Controller
Embedded System Design, Lab 2

Ben Lorenzetti

September 25, 2015

Contents

1 Objectives and Problem Description

The primary objective of this lab was learning to interface with two new types of peripherals: a servo motor and potentiometers. Several new functions in PBASIC are introduced for these types of actuators and transducers.

1.1 Rudder Controller

Control the angular position of a tiny ship's rudder (i.e. the horn of your servo) based on the input from a steering wheel (i.e. your potentiometer). The full range of motion of the ship's rudder should be $45^o < \theta < 135^o$ and it should be controlled linearly by the full range of the potentiometer.

1.2 Wiper Controller

Instead of controlling angular position, as was done for the rudder controller, use the potentiometer to control the angular velocity of the servo. The servo represents a single windshield wiper, and should rotate continuously between 15^o and 160^o. The full range of the potentiometer should be used to linearly control the wiper speed from 0 degrees/sec up to the maximum speed.

2 Procedure

2.1 Assignment: Ch. 4 Servo Motor

To prepare for this lab, we were instructed to read chapters 4 and 5 in the Parallax What's a Microcontroller book and complete the example activities.

The parallax servo motor has three connectiong: power, ground, and signal. The signal (white) connection is used to control the angular position of the horn, from 0^o to 180^o. The servo expects a square wave control signal, where the period is approximately 20 ms and the width of the high pulse carries angular displacement information.

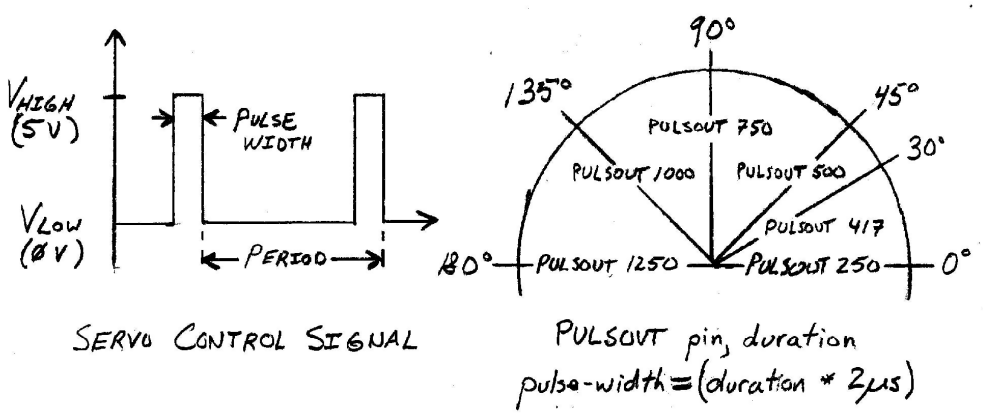

Figure 1: Servo Control Signal Timing

The formula relating pulse width and angular displacement is

$$t_{p-width}(\theta) = \begin{cases} \frac{500\mu s}{45^o}\theta + 500\mu s & 10^o < \theta < 170^o \\ 0\mu s & \text{servo off} \end{cases} \quad (1)$$

The function for doing pulse width modulation in PBASIC is `PULSOUT pin, duration`, where `pin` is the I/O port number and `duration` is the length of time high. Of course, since this Stamp and PBASIC, the board isn't quite fast enough so `duration` is in units of $2\mu s$ and you have to complete the pulse width modulation yourself by `PAUSE`ing for 20 ms, or however long the period should be approximately.

To avoid dealing with PBASICs strange limitations, we can rewrite equation 1 as

$$\text{duration} = \frac{50}{9} * \theta + 250, \quad 10^o < \theta < 170^o \tag{2}$$

Chapter 4 also provided two more expressions in PBASIC that are helpful with servo motors:

```
i VAR Word
FOR i = 0 TO 150
...
LOOP
}
```

and `duration = duration MIN 350 MAX 1150`.

2.2 Assignment: Ch.5 Potentiometers

Not surprisingly, the Basic Stamp is too primative to have an analog-to-digital converter, so you can only measure analog inputs with a function called `RCTIME`, and then only if you construct an RC decay circuit–and again only if the input signal is stable for a long enough charging time and can source enough current.

The function prototype is `RCTIME pin, state, duration`, where `pin` is the I/O pin number (0-15), `state` is the expected charged status of the pin (0 or 1), and `duration` is a variable to store the decay time–in units of $2\mu s$. The decay time is taken when the voltage on `pin` falls below $1.4V$, if the original charge `state` was 1.

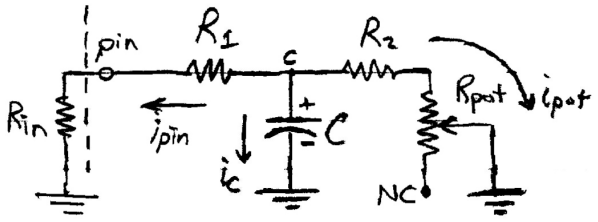

Figure 2: Input Circuit for Reading a Potentiometer with `RCTIME`

An input source that is well suited to `RCTIME` is a rotary potentiometer. A circuit combining a potentiometer and capacitor is shown if figure 2.

If we assume infinite input resistance once `RCTIME` switches the pin direction to input, then $i_{pin} = 0$ and $V_{in} = V_c$. Taking Kirchoff's Current Law at node c allows us to solve for input voltage as a function of time and initial condition across C.

$$i_{pin} + i_c + i_{pot} = 0$$

$$0 + C\frac{dV_{in}}{dt} + \frac{V_{in}}{R_2 + R_{pot}} = 0$$

$$\frac{-V_{in}}{R_2 + R_{pot}} = C\frac{dV_{in}}{dt}$$

$$\frac{-1}{C(R_2 + R_{pot})} * dt = \frac{1}{V_{in}} * dV_{in}$$

$$\frac{-t}{C(R_2 + R_{pot})} + A = ln(V_{in})$$

$$e^{ln(V_{in})} = e^{\frac{-t}{C(R_2+R_{pot})}}$$

$$V_{in} = e^A * e^{-t/C(R_2+R_{pot})}$$

$$V_{in}(t) = V_0 e^{-t/\tau} \qquad \tau = (R_2 + R_{pot})C \tag{3}$$

Using equation 3, the decay time required to discharge from V_{HIGH} to V_{LOW} is

$$V_{LOW} = V_{HIGH} e^{-t_{discharge}/\tau}$$

$$t_{discharge} = \tau * ln\left(\frac{V_{HIGH}}{V_{LOW}}\right)$$

For $V_{HIGH} = 5V$, $V_{LOW} = 1.4V$, $C = 0.1\mu F$, this evaluates to

$$t_{discharge} = \left[\frac{6.365}{10^2} * \frac{2\mu s}{\Omega}\right] * (R_2 + R_{pot}) \tag{4}$$

A $10k\Omega$ potentiometer with $R_2 = 10k\Omega$, leads to input domain $10k\Omega < R_2 + R_{pot} < 20k\Omega$ and output range $636.48 * 2\mu s < t_{discharge} < 1272.97 * 2\mu s$. When we consider that the PBASIC Stamp board only has an integer resolution of $2\mu s$, then the expected result from a call to `RCTIME, pin, 1, duration` is in the set

$$\texttt{duration} = \{637, 638, 639, ...1272, 1273\}$$

when used with a 0.1 μF capacitor.

Actual testing showed with the circuit shown in figure 3 showed an experimental range of

$$\texttt{duration} = \{619, 620, 621, ...1288, 1289\} \tag{5}$$

2.3 Rudder Controller

The circuit contructed for the rudder controller is shown in figure 3. In the potentiometer input circuit, a 220 Ω resistor is used to prevent a current inrush greater than the maximum I/O pin current for the microcontroller.

$$R_{limiter(MIN)} = \frac{V_{CC} - V_{SS}}{I_{I/O(MAX)}} = \frac{5V}{25mA} = 200\Omega$$

The transient discharge of this potentiometer circuit was solved in equation 3, assuming that the charged voltage was nearly 5V. To ensure that the potentiometer's parallel path to ground does not significantly affect the charging time or the charging potential, resistor R_2 was selected to be 10 $k\Omega$ such that $R_2 + R_{pot} >> R_1$ for all values of R_{pot}. With this assumption, charging current through R_{pot} can be neglected and the charging time is approximately

$$\tau_{charge} = 220\Omega * 0.1\mu F = 0.022ms$$

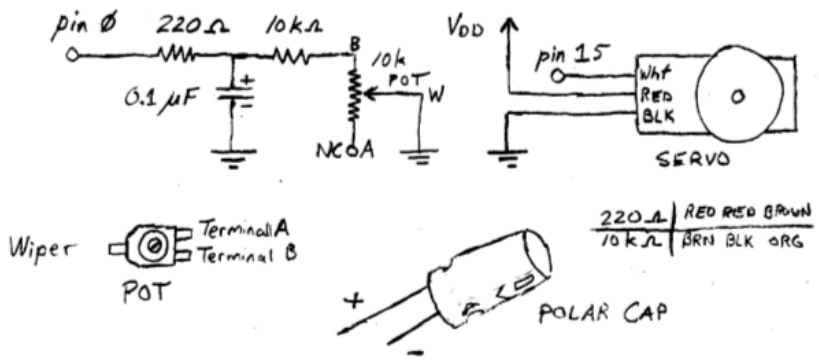

Figure 3: Rudder Controller and Wiper Controller Circuit

Based on τ_{charge}, a charging `PAUSE` of 1 ms is more than adequate.

From theory (equation 4) and testing (set 5), the domain of input discharge time values is 619–1289. The output range should drive the servo between 45^o–135^o. Using equation 2, the output range should be 500–1000. The mapping should be linear:

$$y_2 - y_1 = m(x_2 - x_1)$$

$$1000 - 500 = m(1289 - 619)$$

$$m = 0.746 = \frac{191}{256}$$

$$y_2 = 0.746(x_2 - 619) + 500$$

The microcontroller cannot do floating point arithmetic, but it does have a fractional (middle) multiplication operator that allows you to multiply by fractions of 256. Thus the final mapping equation is

$$t_{p-width} = \frac{191}{256}(t_{discharge-time} - 619) + 500 \tag{6}$$

where both times are in units of $2\mu s$. This is implemented:

```
pulse_width = ((discharge_time - 619) */ 191) + 500
```

Once the potentiometer circuit's discharge time has been mapped to a pulse width, all that remains is to output the pulse and loop indefinitely with an appropriate delay period to complete the pulse- width modulation and input monitoring. The flowchart for the implentation is shown in figure 4.

2.4 Wiper Controller

The wiper controller used the same circuit as the rudder controller (figure 3); the only difference between the two is the implementation. The rudder controls angular position, while the wiper implementation controls angular speed.

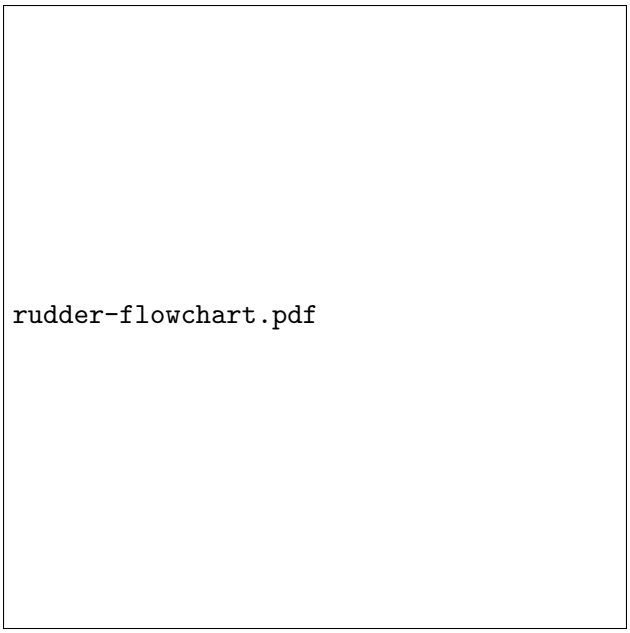

Figure 4: Rudder Controller Implementation Flowchart

The maximum frequency at which the pulse width, and thus angular position, can be changed is the inverse of the pulse width period.

$$f_{max} = \frac{1}{T} = \frac{1}{20ms} = 50Hz$$

At this frequency (or a integer fraction of this frequency), angular velocity can be controled by varying the magnitude of change in pulse width per period.

$$\omega \propto \frac{\Delta(\text{pulse width})}{T} \tag{7}$$

The maximum value for Δpulse_width comes from the full range of motion, which is specified for this project as $15^o < \theta < 160^o$. From equation 2, this corresponds to `333 < duration < 1139`. So the maximum potential swing per period is

$$1139 - 333 = 806 * [2\mu s]$$

For smoothness, we will reduce the maximum swing per period by a factor of 16 (somewhate arbitrarily), leaving us with 50 possible speeds. Now, like was done for the rudder controller, the discharge_time domain of 619–1289 needs to be mapped linearly to 50 speeds, starting at speed 0.

$$50 - 0 = m(1289 - 619)$$

$$m = 0.0746 \approx \frac{19}{256}$$

$$\Delta\text{pulse\_width}_{(\text{per period})} = \frac{19}{256}(t_{\text{discharge}} - 619) \tag{8}$$

where again the pulse width, discharge time, and 619 constant are in units of 2 μs.

We have the the pulse width interval the wiper's range of motion, we have a function for potentiometer input from the rudder project, we have a mapping between the potentiometer discharge time and rate of change in pulse width; all that is needed is an implementation. The implementation flowchart is more complicated than the rudder controller because the wiper has to continuously rotate between two extremes and change directions, in addition to monitoring the pot. and varying speed. How this is done is shown in the implementation flowchart in figure 5.

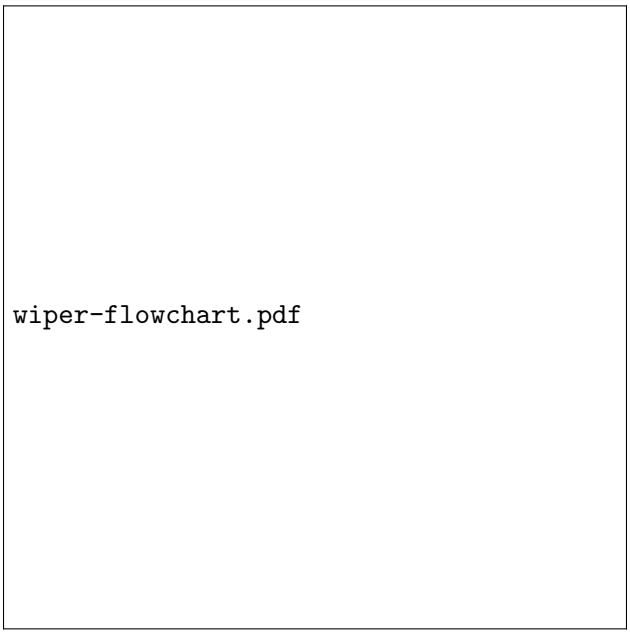

Figure 5: Wiper Controller Flowchart

3 Expected Results

I expected both circuits to behave according to the problem statements.

4 Experiment and Design Revisions

4.1 Rudder Controller

When I first built this project's implementation, the controller worked for 90% of the potentiometer's input range. However, when the discharge time was on the lower range of its possible values, the rudder would suddenly shift from near 45^o to the 135^o position. After adding a `DEBUG` statement, I saw the following transition occuring:

```
discharge_time = 622, pulse_width = 502
discharge_time = 621, pulse_width = 501
discharge_time = 620, pulse_width = 500
discharge_time = 619, pulse_width = 500
discharge_time = 618, pulse_width = 1000
```

What was occuring was that the `discharge_time` from the RC circuit was coming back slightly lower than the minimum value I was expecting. Thus the subraction `(discharge_time - MAPPING_X1)`

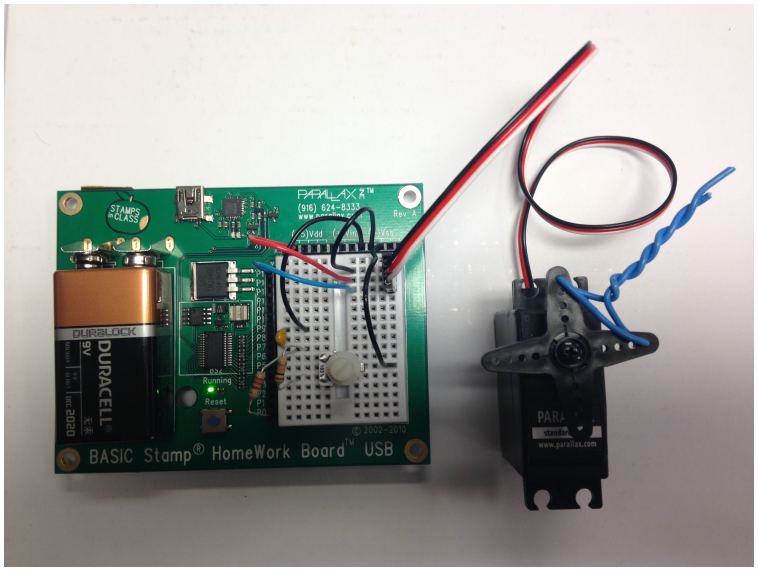

Figure 6: Rudder Position Controller Board

resulted in a negative number, except the variable was not declared as signed so the result of the two's-complement arithmetic was the maximum possible value for the 16-bit Word. Later in the code is the function `MAX 1000`, so that is where the 1000 is coming from.

The fix for this bug was very simple, simply treating the `discharge_time` variable with `MIN MAPPING_X1` prior to calculating the `pulse_width`.

4.2 Wiper Controller

When I first implemented this circuit the program worked but the servo was extremely jerky. At first I thought this meant my speeds were too high, but it turned out it was only due to a `DEBUG` statement in the loop that was throwing off the pulse width timing.

Removal of the `DEBUG` statement fixed the error.

5 Observations

Both circuits eventually worked as expected, and photos of the working boards are shown in figure 6 and figure 7.

6 Discussion

7 Exercises

There were no exercises given.

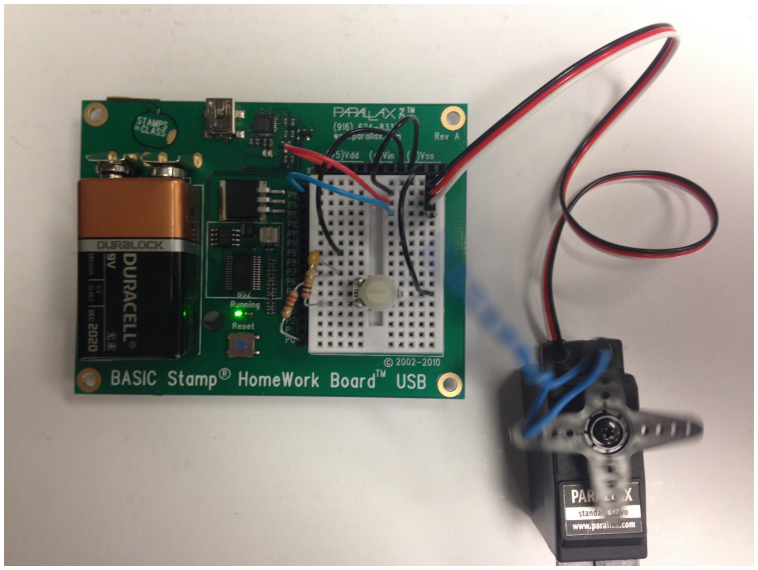

Figure 7: Wiper Speed Controller Board

8 Implementation Code

8.1 Rudder Controller

```
' Rudder-Controller.bs2
' {$STAMP BS2}
' {$PBASIC 2.5}

POT_PIN CON 0
SERVO_PIN CON 15
CHARGE_STATE CON 1
CHARGE_TIME CON 1
MAPPING_M CON 191
MAPPING_X1 CON 619
MAPPING_Y1 CON 500

discharge_time VAR Word
pulse_width VAR Word

DO
  ' Set POT_PIN to output the charging voltage and wait CHARGE_TIME
  DIRS = DIRS | (1 << POT_PIN)
  ' If CHARGE_STATE is high, set OUTS Pot. bit to high
  OUTS = OUTS | (CHARGE_STATE * (1 << POT_PIN))
  ' If CHARGE_STATE is low, set OUTS Pot. bit to low
  OUTS = OUTS & (~ ( (CHARGE_STATE ^ 1) * (1 << POT_PIN)))
  PAUSE CHARGE_TIME

  ' Discharge the capacitor through the Potentionmeter and record time
  RCTIME POT_PIN, CHARGE_STATE, discharge_time

  ' Map the discharge_time input to pulse_width output
  discharge_time = discharge_time MIN MAPPING_X1
  pulse_width = MAPPING_Y1 + ((discharge_time - MAPPING_X1) */ MAPPING_M)
  pulse_width = pulse_width MIN 500 MAX 1000

  ' Generate Pulse-Width Waveform for Servo
  pulse_width = pulse_width MIN 500 MAX 1000
  PULSOUT SERVO_PIN, pulse_width
  PAUSE (19 - CHARGE_TIME)
LOOP
```

8.2 Wiper Controller

```
' Wiper-Controller.bs2
' {$STAMP BS2}
' {$PBASIC 2.5}

POT_PIN CON 0
SERVO_PIN CON 15
CHARGE_STATE CON 1
CHARGE_TIME CON 1
MAPPING_M CON 19
MAPPING_X1 CON 619
MAPPING_Y1 CON 0
FIFTEEN_DEG CON 333
ONE_HUNDRED_SIXTY_DEG CON 1139

discharge_time VAR Word
pulse_width VAR Word
delta_pulse_width VAR Word
direction VAR Bit

pulse_width = 0
direction = 0

DO
  ' Set POT_PIN to output the charging voltage and wait CHARGE_TIME
  DIRS = DIRS | (1 << POT_PIN)
  ' If CHARGE_STATE is high, set OUTS Pot. bit to high
  OUTS = OUTS | (CHARGE_STATE * (1 << POT_PIN))
  ' If CHARGE_STATE is low, set OUTS Pot. bit to low
  OUTS = OUTS & (~ ( (CHARGE_STATE ^ 1) * (1 << POT_PIN)))
  PAUSE CHARGE_TIME

  ' Discharge the capacitor through the Potentionmeter and record time
  RCTIME POT_PIN, CHARGE_STATE, discharge_time

  ' Map the discharge_time input to the delta_pulse_width per period
  discharge_time = discharge_time MIN MAPPING_X1
  delta_pulse_width = MAPPING_Y1 + ((discharge_time - MAPPING_X1) */ MAPPING_M)

  ' Change the pulse_width based on the delta_ variable and the current direction
  IF (direction = 0) THEN Invert_Delta_Variable
  Invert_Return:
  pulse_width = pulse_width + delta_pulse_width

  ' Change Directions if Angular Displacement is at Min or Max
  IF (pulse_width <= FIFTEEN_DEG) THEN Invert_Direction
  IF (pulse_width >= ONE_HUNDRED_SIXTY_DEG) THEN Invert_Direction
  Invert_Direction_Return:

  ' Generate Pulse-Width Waveform for Servo
  pulse_width = pulse_width MIN FIFTEEN_DEG MAX ONE_HUNDRED_SIXTY_DEG
  PULSOUT SERVO_PIN, pulse_width
  PAUSE (19 - CHARGE_TIME)
LOOP

Invert_Delta_Variable:
  ' Two's Complement
  delta_pulse_width = 1 + (~delta_pulse_width)
GOTO Invert_Return

Invert_Direction:
  direction = direction ^ 1
GOTO Invert_Direction_Return
```