June 11, 2015 at 23:52

1. Introduction. This is the firmware portion of the propulsion system, featuring piruett turning.

This will facilitate motion by taking "thrust" and "radius" pulse-width inputs and converting them to the appropriate motor actions.

Both pulse-width inputs will have some dead-band to allow for full stop.

The pulse-width from the receiver is at 20 ms intervals. The time ranges from 1000–2000 ms, including trim. 1500 ms is the width for stopped. The levers cover ±0.4 ms and the trim covers the balance.

Math for radius...I think this is right:

Where: t is track r is radius v is value f is factor

min r is 1 max r is 127

get this value for v [1] [127] [255]

r=factor*abs(127-v)

For non-zero r Inside=(2r-t)/2r Outside=(2r+t)/2r

Port motor pulse will be applied to ???, starboard will be at ???. The median time will be subtracted from them for a pair of signed values thrust and radius. The value will be scaled.

The thrust and radius will be translated to power to the port and starboard motors. When near median the motors will be disabled. The motors will also be disabled when there are no input pulses. Each motor need direction and power so that's 4 signals of output.

The radius control will also be the rotate control, if thrust is zero.

Adding the two signal of input, I need more I/O than the trinket has. So—I now have a $10 Pro Trinket with far more capability. It has an ATmega328.

The ATmega328 has a fancy 16 bit PWM with two comparators, Timer 1. This has an "Input Capture Unit" that may be used for PWC decoding. That's an elegant solution

One of the other timers will do more than fine for the two motors.

For the PWC measurement, this app note, AVR135, is helpful: www.atmel.com/images/doc8014.pdf

In the datasheet, this section is helpful: 16.6.3

Since I have two signals, maybe the best way to use this nice feature is to take the PWC signals into the MUX, through the comparator and into the Input Capture Unit. First pick the thrust, set for a rising edge, wait, grab the time-stamp and set for falling edge, wait, grab the time-stamp, do modulus subtraction, switch the MUX, set for rising, reset the ICR, wait...

place-holder code below ==========

Extensive use was made of the datasheet, Atmel "Atmel-8271I-AVR- ATmega-Datasheet_10/2014".

⟨ Include 4 ⟩
⟨ Types 5 ⟩
⟨ Prototypes 6 ⟩

2. `"F_CPU"` is used to convey the Trinket Pro clock rate.

```
#define F_CPU 16000000UL
```

3. Here are some Boolean definitions that are used.

```
#define ON 1
#define OFF 0
#define SET 1
#define CLEAR 0
```

4. ⟨ Include 4 ⟩ ≡

```
#include <avr/io.h>      /* need some port access */
#include <util/delay.h>      /* need to delay */
#include <avr/interrupt.h>      /* have need of an interrupt */
#include <avr/sleep.h>      /* have need of sleep */
#include <stdlib.h>
#include <stdint.h>
```

This code is used in section 1.

5. Here is a structure to keep track of the state of things.

⟨ Types 5 ⟩ ≡

```
typedef struct {
  uint8_t portOut;      /*  */
  uint8_t starOut;      /*  */
  uint16_t thrust;      /*  */
  uint16_t radius;      /*  */
} statestruct;
```

This code is used in section 1.

6. ⟨ Prototypes 6 ⟩ ≡

```
void ledcntl(uint8_t state);      /* LED ON and LED OFF */
```

This code is used in section 1.

7. My lone global variable is a function pointer. This lets me pass arguments to the actual interrupt handlers. This pointer gets the appropriate function attached by the `"ISR()"` function.

8. Here is *main*( ).

```
int main(void){
    ⟨ Initialize the inputs and capture mode 20 ⟩⟨ Initialize pin outputs 17 ⟩
```

9. Of course, any interrupt function requires that bit “Global Interrupt Enable” is set; usually done through calling sei().

```
sei( );
```

10. Rather than burning loops, waiting 18 ms for something to happen, the “sleep” mode is used. The specific type of sleep is ‘idle’. In idle, execution stops but timers continue. Interrupts are used to wake it.

```
⟨ Configure to idle on sleep 18 ⟩ledcntl(OFF);
ADMUX |= (~(1 << MUX2) | ~(1 << MUX1) | ~(1 << MUX0));      /* Set to channel 0 */
```

11. This is the loop that does the work. It should spend most of its time in *sleep_mode*, comming out at each interrupt event caused by an edge.

```
for ( ; ; )      /* forever */
{
```

12. Now we wait in “idle”.

```
sleep_mode( );
```

13. If execution arrives here, some interrupt has been detected.

```
static char toggle = 0;
{
  if (toggle) {
    ledcntl(ON);
    TCCR1B &= ~(1 << ICES1);      /* wait for falling edge */
  }
  else {
    ledcntl(OFF);
    TCCR1B |= (1 << ICES1);      /* wait for rising edge */
  }
  toggle = toggle ? 0 : 1;
}
}     /* end for */
return 0;      /* it's the right thing to do! */
}     /* end main() */
```

14. Here is a simple function to flip the LED on or off.

```
void ledcntl(uint8_t state)
{
  PORTB = state ? PORTB | (1 << PORTB5) : PORTB & ~(1 << PORTB5);
}
```

15.

16. These are the supporting routines, procedures and configuration blocks.
Here is the block that sets-up the digital I/O pins.

17. ⟨ Initialize pin outputs 17 ⟩ ≡
```
{   /* set the led port direction; This is pin #13 */
  DDRB |= (1 << DDB5);
}
```
This code is used in section 8.

18. ⟨ Configure to idle on sleep 18 ⟩ ≡
```
{
  SMCR &= ~(1 << SM2);
  SMCR &= ~(1 << SM1);
  SMCR &= ~(1 << SM0);
}
```
This code is used in section 10.

19. To enable this interrupt, set the ACIE bit of register ACSR.

20. ⟨ Initialize the inputs and capture mode 20 ⟩ ≡
```
{
  ADCSRB |= (1 << ACME);      /* Conn the MUX to (-) input of comparator */
  ADMUX |= (1 << MUX0);      /* Set bit MUX0 of register ADMUX */
  ADCSRA &= ~(1 << ADEN);      /* Turn off ADC to use its MUX (per 23.2) */
  DIDR0 |= ((1 << AIN1D) | (1 << AIN0D));      /* Disable digital inputs */
  ACSR |= (1 << ACBG);      /* Connect the + input to the band-gap reference */
  ACSR |= (1 << ACIC);      /* Enable input capture mode */
  TIMSK1 |= (1 << ICIE1);      /* Enable input capture interrupt */
  TCCR1B |= (1 << ICNC1);      /* Enable input capture noise canceling */
  TCCR1B |= (1 << CS10);      /* No Prescale. Just count the main clock */
}
```
This code is used in section 8.

⟨ Configure to idle on sleep 18 ⟩ Used in section 10.
⟨ Include 4 ⟩ Used in section 1.
⟨ Initialize pin outputs 17 ⟩ Used in section 8.
⟨ Initialize the inputs and capture mode 20 ⟩ Used in section 8.
⟨ Prototypes 6 ⟩ Used in section 1.
⟨ Types 5 ⟩ Used in section 1.