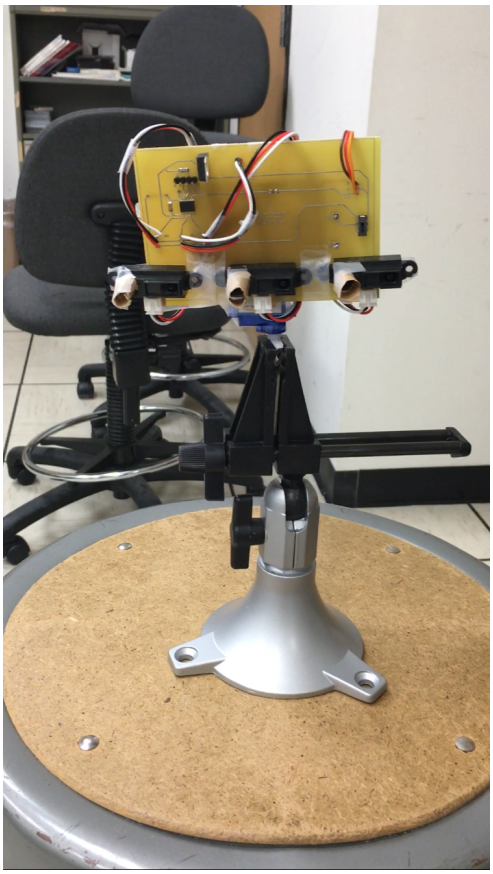
Motion Follower

ELEC 327 Final Report

Justin Pensock

May 4, 2016



The Concept:

For my final project, I created a “pseudo-autonomous” robot that can ‘see’ an object and follow it around a 360 degree plane. I did this using a servo motor controlled by brief PWM pulses and distance sensing using SHARP IR sensors.

The Design:

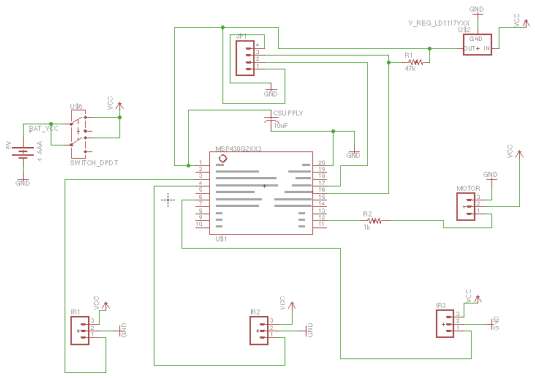
The design of the robot consists of two parts: the body and the head.

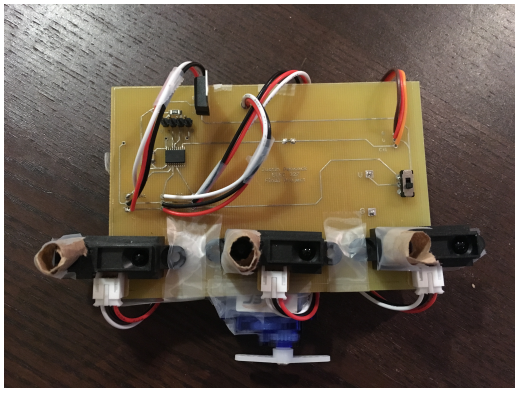
The Body

The body was simply a (stationary) clamp connected to the rotating arm of the motor, which was connected to the head. Whenever the motor rotated, the clamp kept the arm stationary and the bulk of the motor (connected to the head) was what rotated around.

The Head

(the schematic and assembled PCB seen below):





At the base of the head is the blue servo motor. This micro-servo is a standard continuous rotation servo, controlled by PWM pulses. These pulses, ranging from 1-2ms depending on which way you want to rotate it, drive the motor and control the position you want to set the motor at. To move full speed forward set the motor to 180 degrees, full speed backward corresponds to 0 degrees, and stop is 90 degrees. For my project,

when I wanted to rotate to the left, I wanted the motor to go to 65 degrees (a slow backwards rotation) and when I wanted to rotate right, I drove it with a pulse corresponding to 95 degrees. It is connected to pin 2.4 on the MSP430 via wires routed across the back of the head, with a 1k resistor to protect the control input wire of the motor.

Above the motor are the three IR sensors. These sensors have two components: the IR emitter and the IR detector and they work by emitting an IR signal that bounces off objects and is detected by the detector. Based on the reading of the detector, the sensor outputs an analog voltage signal – ranging from 3 V when an object is close (within 10 cm) and 0.4 V when an object is far (up to a maximum of 80 cm). I had to place those "cones" around the detectors to cancel out IR noise and to ensure it was only detecting IR signals bouncing back directly at it (i.e. an object was directly in front of it). Each of these 3 IR sensors is connected to an ADC analog input on the MSP430 to convert the analog signal into quantitave values that can be compared.

The motor and IR sensors are powered off a 6 V rail from 4 AAA batteries in a battery pack attached to the back (not shown above), and the MSP430 is powered off the constant 3.3 V output of a voltage regulator, also connected to the battery pack.

The Software Architecture:

The motor is controlled by PWM using Timer A1, sourced from SMCLK. I could not use the VLO to control the motor as its 12 KHz frequency could not be separated nicely into the PWM degree "steps" needed to drive the motor's rotation, thus I used the DCO configured to run at 1 MHz. This limited the system to only go into LPM1 when I was polling the sensor data instead of the more efficient LPM3.

For the IR sensors, each ADC analog input is converted at every iteration of the main loop (A1, A2, and A4) using LPM1 and the ADC ISR and their values are compared to each other. The idea is that whichever has the max value, rotate the motor in that direction (or not at all if it's the middle sensor). However, the actual system is a bit more complex than that. The ADC values corresponding to no object being detected are quite high, but if an object is detected, that value spikes. Thus I had to set a minimum threshold value such that, if the maximum value of the readings is below that threshold, treat it as no object being detected. There was also another problem that an object was being detected but it was equally close to 2 sensors. To solve this I had to specify a distance threshold such that, for an object to be detected, the maximum value had to be at least that threshold greater than the next largest value. Finally, I ran into a problem where the motor was switching directions too abruptly. To fix this, I added a buffer delay to whenever the head wanted to switch directions, not allowing the switch until that delay had passed.

After that logic to determine the direction, I then change the PWM register value accordingly and delay a few cycles to allow the motor to turn. This all occurs within the main loop.

I had played around with the idea of only polling the ADC inputs in the main loop and having the watchdog timer (WDT) generate interrupts, in which I would set the PWM and drive the motor accordingly. However, I realized the motor control must be synchronous with the ADC recordings, as we would not want find the direction based on the max ADC value when the MSP430 was in the critical section of recording the ADC values (i.e. it had recorded values 1 and 2, but was waiting to poll the third sensor). In this case, I could have my hand over sensor 3 but the new value 1 could be higher than the old value 3 and the head would rotate the wrong way. I even toyed around with the use of locking movement until ADC had completed its recording but this would have wasted WDT interrupts and system resources, so I decided to shy away from an asynchronous approach and have the PWM be set immediately after the sensor inputs had been processed.