Gesture Control Car

Mahdiul Chowdhury (BU Username: mahdiulc)
Zachary Safran (BU Username: zsafran)
David Abadi (BU Username: dabadi)

Abstract:

A remote control car that was controlled using a Gumstix Verdex Pro Microcontroller, and the 3-axis accelerometer position data from a TI-Chronos watch was created. The x axis position data was used to set the width of the PWM signal controlling the speed and direction of the car, while the y axis position data was used to set the width of the PWM signal controlling whether or not the car was turning. Additionally, an Arduino was used as an external ADC that would process the data from an IR rangefinder and set one of its pins that was connected to a GPIO of the Gumstix high when an obstacle was within ~33cm of the front of the car.

1. Introduction

Remote controlled vehicles are used to assist search and rescue missions and can also be used as sacrificial lambs to deactivate explosives. In this project a Gumstix Verdex Pro Microcontroller, and the data from a TI EZ430-Chronos Watch were used to control the motion of a Exceed RC Maxstone Rock Crawler Car. This car, its NiMH rechargeable battery, and Electronic Speed Control unit (ESC) were provided to us by Prof. Little, along with some sample Arduino code [1][2]. The speed of the car was controlled by using an Adafruit 16-Channel 12-bit PWM/Servo Driver, PCA9685, breakout board to send PWM signals to the ESC [3]. Another PWM signal from the breakout board was used to control the angle of the servo in charge of turning the car. The Gumstix communicates with, and sends commands to the PWM breakout board over I2C, from its pins 117 (SCL) and 118 (SDA). In order to determine the correct I2C commands to send to the PWM breakout board and understand how it works the Adafruit Arduino library and PCA9685 datasheet were referenced and the relevant Arduino functions were converted to C code[6][7]. The Gumstix received accelerometer orientation data from the Chronos watch through the USB RF receiver access point that came with the Chronos watch. The EC535 TA Rushi provided us with some sample code to interface our user level program with the Chronos watch access point[5]. One last feature of our Gesture Control Car is obstacle detection. Obstacle detection was implemented by using an Arduino as as an external ADC that would process the data from an IR rangefinder and set one of its pins that was connected to a GPIO of the Gumstix high when an obstacle was within ~33cm of the front of the car. A blue LED is also lit up to provide some feedback to the user when an obstacle is detected. Overall, we were able to use these components to build a remote control car with obstacle detection that could be controlled by turning one’s wrist.

2. Design Flow

Components project includes:

1. Chronos Watch
2. Gumstix Board
3. PWM Breakout Board
4. Electronic Speed Controller and Servo
5. IR Range Finder
6. Arduino
7. Battery (5v and 7.2v)
8. Rock Crawling Car

Task project includes:

1. I2C communication between Gumstix and PWM Breakout Board
2. PWM signal generator for Servo and ESC
3. RF communication between Gumstix and Chronos watch
4. GPIO level reading using Gumstix and Arduino
5. Sensor Reading (IR Range Finder) using Arduino

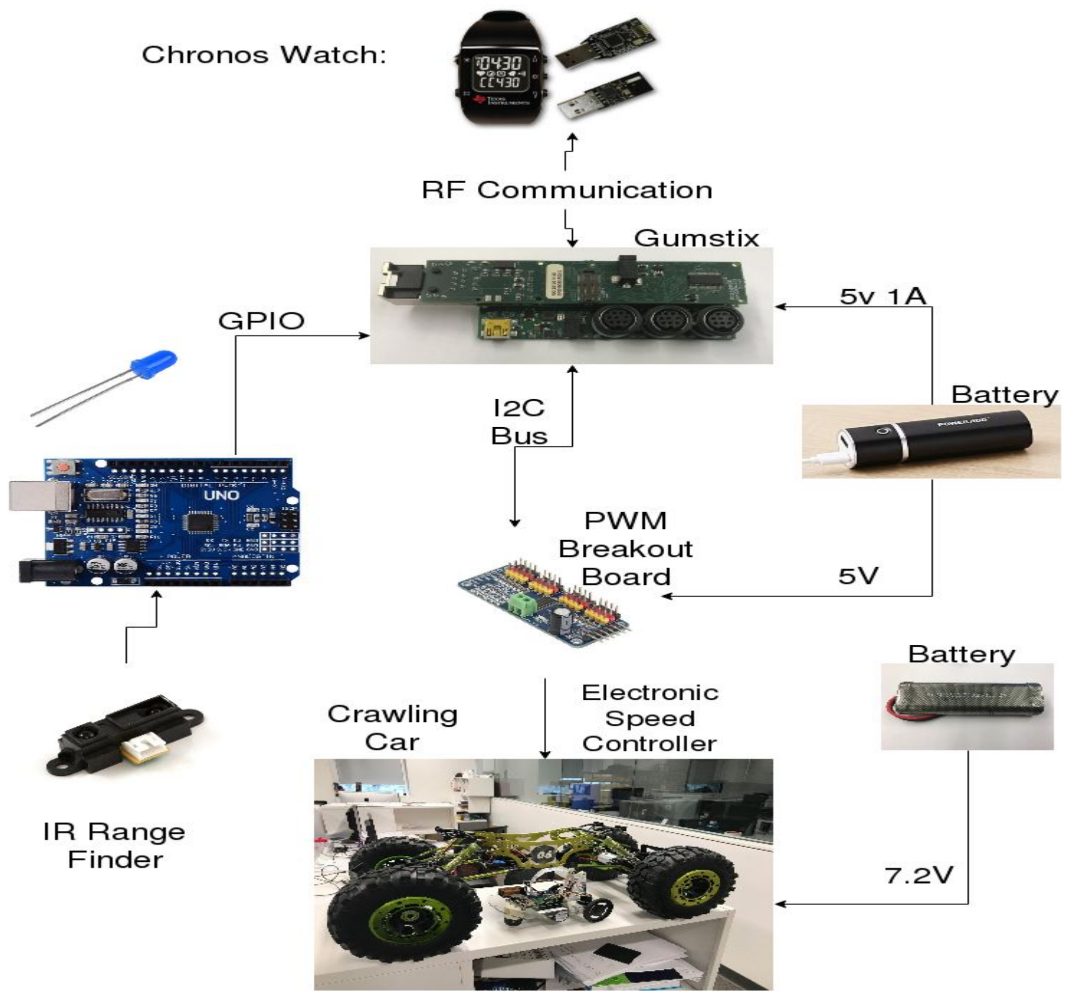


Figure one: Data Flow

Description:

Chronos watch [4] interfaces with Gumstix board using RF communication, to constantly send the 3-axis accelerometer data. Gumstix board is directly connected with a access point, USB serial port, which is receiving the data repeatedly from the chronos watch. We have a user level program run in Gumstix board opening the device port "/dev/ttyACM0" and sending request to load the data in the registers. After reading the chronos watch accelerometer data, we are using I2c bus to send proper PWM signal for forward, backward, neutral and angle to the electronic speed controller and servo. In addition to that Gumstix board is reading the pinout 101 to check whether the range finder detects any obstacles within the range of ~30 cm.

Work Distribution:

a. David Abadi (33.3%):
    i. Worked on Chronos Watch interfacing with the Gumstix board.
    ii. Incorporating the accelerometer data to pwm signal.
    iii. Worked on writing the Kernel Module and modified the user level program to stop when there was an obstacle in front.

b. Mahdiul Chowdhury (33.3%):
    i. Worked on calculating the PWM signal.
    ii. Incorporating the Chronos data to control the servo and ESC.
    iii. Worked on wiring the hardwares for the project.
    iv. Worked on IR range finder to avoid the obstacles.

c. Zachary Safran (33.3%):
    i. Researched I2C and how to work with Chronos Watch
    ii. Converted the Adafruit Arduino PWM Library to C to work with the Gumstix.
    iii. Worked with Mahdiul to use an oscilloscope and the sample arduino crawler car control code to calculate the correct values to use with the setPwm() function.
    iv. Tried but failed to convert the Adafruit ADS1X15 library to C [9][10]. The difference between with communicating with the ADS1115 and the PCA9685 is that the ADS1115 has 16 bit registers while the PCA9685 has 8 bit registers [11].

3. Project Details

B. User-level Program (See /code/ul/car_control.c for source code)
    a. In the user level program, we are first opening the access point for the Chronos Watch and pass a start string to make the access point connect to the watch (lines 92-107). To get the data from the Chronos Watch we first have to write the string “\xFF\x08\x07\x00\x00\x00\x00” and then we can read from the access point. From the watch we get four 8 bit values. We get the x, y, z values from the accelerometer and a data validation value (named u in our code). If u is equal to 1 then we got valid data from the watch [5]. According to the x and y values from the watch we are able

to set the desired values for the PWM duty cycles for the ESC and Servo (lines 149-162). Also by reading from the Kernel Module, MyModule, every time we read from the watch we are able to see if there is an object in front of the car and prevent the car from going forward even if the user is signaling with the watch to go forward.

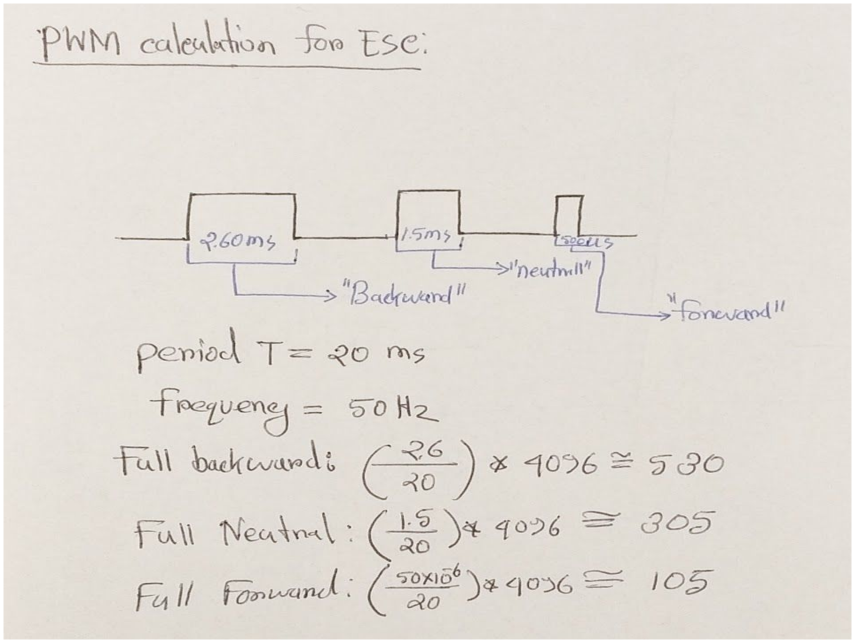

PWM calculation for Esc:
2.60ms
1.5ms
500us
"Backward"
"neutrall"
"forward"
period T = 20 ms
frequency = 50 Hz
Full backward: $\left(\frac{2.6}{20}\right) * 4096 \cong 530$
Full Neutral: $\left(\frac{1.5}{20}\right) * 4096 \cong 305$
Full Forward: $\left(\frac{50 \times 10^{-6}}{20}\right) * 4096 \cong 105$

Figure two: PWM signal calculation

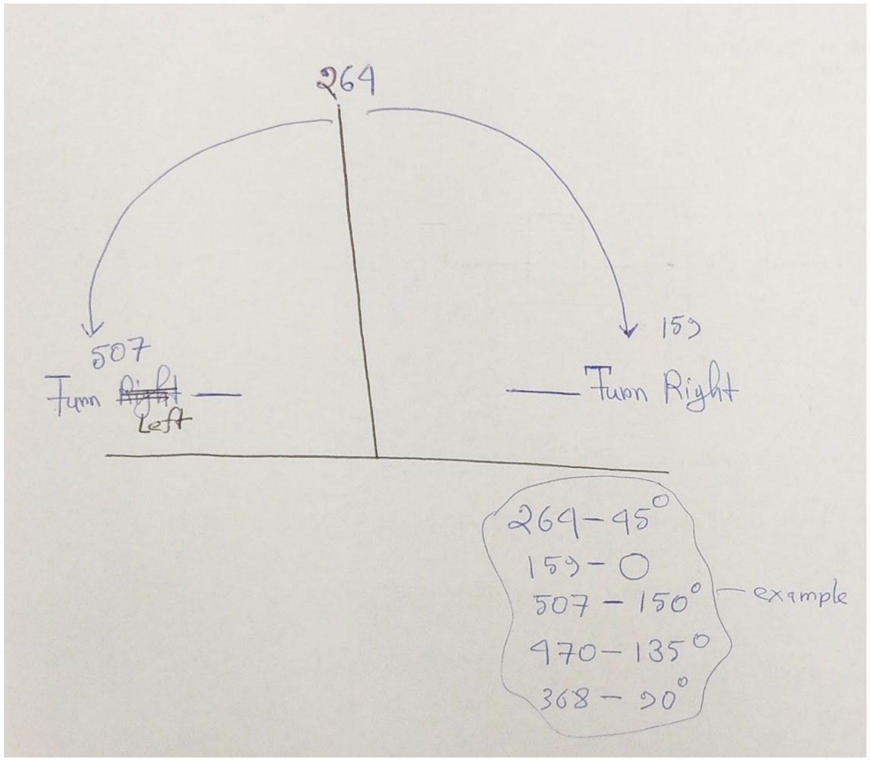
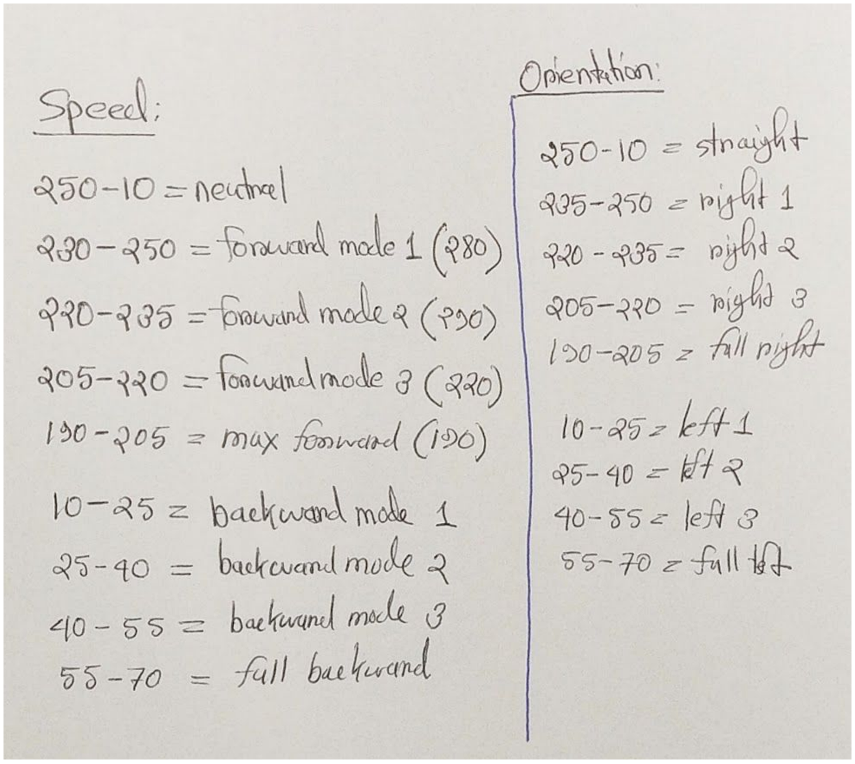


Figure three: PWM signal for servo

Figure two and three is describing how we calculated the pwm signal for electronic speed controller and servo. In figure two, using oscilloscope we calculated the period and time on phase of duty cycle. We divided the time on phase by period and then multiply it with 4096 steps of the PWM driver, which gives us the exact value of PWM for full forward, full backward and full neutral. In figure three, we calculated the PWM range for servo control, for example decreasing the value 260 to 159 will turn the car in right direction and increasing the value from 264 to 507 will turn the car in the left direction.



Speed:

250-10 = neutral
230-250 = forward mode 1 (280)
220-235 = forward mode 2 (290)
205-220 = forward mode 3 (220)
190-205 = max forward (190)

10-25 = backward mode 1
25-40 = backward mode 2
40-55 = backward mode 3
55-70 = full backward

Orientation:

250-10 = straight
235-250 = right 1
220-235 = right 2
205-220 = right 3
190-205 = full right

10-25 = left 1
25-40 = left 2
40-55 = left 3
55-70 = full left

Figure four: First testing for PWM signal

Figure four is showing how we calculated the range to control the car based on the x and y value of the chronos watch. Increasing x value means, the car has to speed up from lower to higher. That is why we have multiple mode in forward and backward direction. We used the same methodology for the orientation of the car, increasing y value means, the car has to increase the angle of turning right or left. After we calculated the range of PWM signal, we wrote a function "calibration" to calibrate the ESC based on our calculated value. To calibrate the the ESC for forward, backward and neutral motion, we send 530,105 and 305 which is full backward, full forward and full neutral signal. This helps the ESC to adjust with the new signal we generated, without calibration the ESC might burn out as it can not handle the signal.

b. The functions pwmBegin(), pwmReset(), setPWMFreq(), setPWM(), write8(), and read8() were converted from the their counterparts in the Adafruit Arduino PWM Library [6]. The functions write8(file, reg_addr, data) and read8(file, reg_addr) respectively implement the I2C operations of writing to and reading from the registers in the PCA9685 defined by the reg_addr input argument. In both write8() and read8() the first thing done is open the i2c driver "/dev/i2c-0" with read and write permissions. Then, the I2C_SLAVE address of the PCA9685 was set using the ioctl() function.

The smbus functions i2c_smbus_write_byte_data(file, reg_addr, data) and i2c_smbus_read_byte_data(file, reg_addr), which are defined in the i2c-dev.h, were used in write8() and read8() to write and read a byte of data to, or from, the reg_addr register of the PCA9685 [12]. After this conversation is complete the "/dev/i2c-0" file is closed. The pwmBegin() function calls pwmReset() and setPwmFreq(file, 54). PwmReset() writes 0x80 to register 0x00 to reset all of the PWM channels of the PCA9685. SetPwmFreq(file, 54) first reads from 0x00 to get the current configuration of PCA9685. Then this config is & with 0x7F and | with 0x10 and the result is written to 0x00 to make the PCA9685 go to sleep. Once the PCA9685 is asleep the following formula is used to calculate the prescale value that should be written to the Prescale register to set the frequency of the output PWM signals.

```
Input freq = 54;
prescaleval = (25000000/ (4096 * freq * 0.9)) - 1;
```

While testing, we determined that an input freq of 54 would result in the desired output PWM frequency of ~50Hz. After writing this value to the prescale register, the PCA9685 is awoken by writing its original config back to register 0x00.

Lastly, the function setPwm(file, num, on, off), is used to set the PWM output of the PCA9685 channel "num". In this function, write8() is called four times to first write the least significant byte(LSB) and the most significant byte(MSB) of the "on" input value to the proper registers, and then write the LSB and MSB for the "off" input value to the proper registers of the PCA9685. If on = 0, the duty cycle of the resulting PWM signal can be calculated using:

```
Duty cycle% = (off/4096)*100;
```

C. MyModule (See /code/km/mymodule.c for source code)

The Kernel Module is read only module in which we set the GPIO pin 101 to be an edge triggered interrupt. When the interrupt happens we check the level of the pin, if it is high we set the global variable "stop" to 1 else we set it to 0. Every time we read from the Kernel Module we just output the value of the variable "stop".

D. Arduino Program

The arduino code reads the analog data from IR range finder. Arduino is connected with analog pin “A0”. Once it’s reads the value from the sensor, we know that there is an obstacle in front of the car. However, there is a digital pin (Pin 2) in the arduino which becomes high once the car comes close to ~33 cm of the obstacles.

4. Summary

Our project consisted of a car that a person could control using hand gesture and also have his/her hands completely free to do other stuff. There many useful situation where we can use this car to perform task such monitoring areas, search and rescue and transportation. We were able to implement this by using a Chronos Watch, Gumstix Board, an Arduino, a PWM breakout board and an IR range finder.

Remaining challenges of our project could be implementing path planning algorithm and obstacle avoidance feature using a camera. We could have more sensors around the car to avoid collisions not only when the car is going forward but also when it is going backward or turning. Also, one challenge that we could not solve due to the time constraint was having an external ADC, ADS1115, connected directly to the Gumstix Board instead of using the Arduino’s ADC to read from the IR range finder.

References

[1] “Crawler Car Setup”
https://github.com/EC544-BU/EC544_demos/wiki/Guide:-Crawler-Setup
[2] “Sample Arduino Crawler Car Control Code”
https://github.com/EC544-BU/EC544_demos/blob/master/demos/crawlerOscillator/crawlerOscillator/crawlerOscillator.ino
[3] “Adafruit PWM breakout” https://www.adafruit.com/product/815
[4] “Chronos Watch Wiki Page”
http://processors.wiki.ti.com/index.php/EZ430-Chronos?DCMP=Chronos&HQS=Other+OT+chronoswiki#PC.2FChronos_Communication
[5] “Chronos Watch Provided Code”
https://github.com/MahdiulChowdhury/ENGEC535_GestureControlCar/blob/master/watchController.c
[6] “Adafruit Arduino PWM Library”
https://github.com/adafruit/Adafruit-PWM-Servo-Driver-Library

[7] “PCA9685 Datasheet” https://cdn-shop.adafruit.com/datasheets/PCA9685.pdf
[8] “ADS1115 Info Page”
https://learn.adafruit.com/adafruit-4-channel-adc-breakouts/arduino-code
[9] “ADS1115 Arduino Library” https://github.com/adafruit/Adafruit_ADS1X15
[10] “ADS1115 Convertion Attempt”
https://github.com/MahdiulChowdhury/ENGEC535_GestureControlCar/blob/master/ZS/IR_Rangefinder_adc/adc_43019versions/ADS1115_v43019.c
[11] “ADS1115 Datasheet”
https://github.com/MahdiulChowdhury/ENGEC535_GestureControlCar/blob/master/ZS/IR_Rangefinder_adc/ads1115.pdf
[12] “SMBus Reference”
https://github.com/MahdiulChowdhury/ENGEC535_GestureControlCar/blob/master/ZS/IR_Rangefinder_adc/ads1115.pdf

Our Gesture Control Car Project Github link:
https://github.com/MahdiulChowdhury/ENGEC535_GestureControlCar