

Lab 6. Final Report

System integration and Enhancement

EE405 Electronic Design Lab

20154920 송우태

22 June 2016

Table of Contents

Purpose	p. 2
Procedures	p. 2
Results:	p. 2 - 13
Discussion Questions	p. 13 - 21
Reference	p. 22

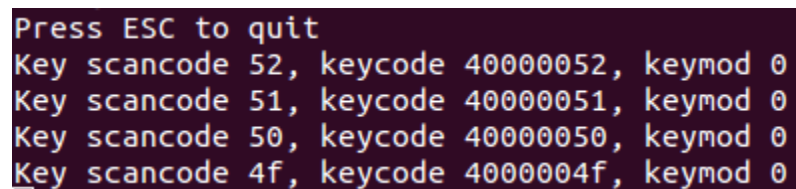
Purpose:

The purpose of lab 6 is to integrate all the functions that we implemented in previous labs and add our enhancements to the robot to create the ultimate robot.

Procedures**A. In-Depth Test of SDL2 Event****B. Test Multi-Tasking****C. System Integration****D. System Enhancement****Results:****A. In-Depth Test of SDL2 Event****Test_Keysym.cpp**

The source code is provided. I compiled and ran the program.

After printing out `e.key.keysym.scancode`, `e.key.keysym.sym`, and `e.key.keysym.mod`, I got the following results:

A terminal window with a dark background and light-colored text. The first line says "Press ESC to quit". The next four lines show key events: "Key scancode 52, keycode 40000052, keymod 0", "Key scancode 51, keycode 40000051, keymod 0", "Key scancode 50, keycode 40000050, keymod 0", and "Key scancode 4f, keycode 4000004f, keymod 0". A cursor is visible at the end of the last line.

```
Press ESC to quit
Key scancode 52, keycode 40000052, keymod 0
Key scancode 51, keycode 40000051, keymod 0
Key scancode 50, keycode 40000050, keymod 0
Key scancode 4f, keycode 4000004f, keymod 0
```

Examining the result, I found that `e.key.keysym.scancode` variable prints out the ASCII code of the key that is pressed. Therefore, I found that `e.key.keysym.scancode` is the variable that is useful for key input.

KeyControlSDL.cpp

In KeyControlSDL.cpp, I printed out e.key.keysym.scancode as character. As a result, I got the following result:

```
Key pressed: 1 Ascii#(in Hex): 31
Key pressed: 2 Ascii#(in Hex): 32
Key pressed: 3 Ascii#(in Hex): 33
Key pressed: q Ascii#(in Hex): 71
Key pressed: w Ascii#(in Hex): 77
Key pressed: e Ascii#(in Hex): 65
Key pressed: a Ascii#(in Hex): 61
Key pressed: s Ascii#(in Hex): 73
Key pressed: d Ascii#(in Hex): 64
Key pressed: z Ascii#(in Hex): 7a
Key pressed: x Ascii#(in Hex): 78
Key pressed: c Ascii#(in Hex): 63
```

As you can see in the above image, you can see that the ASCII codes matches the keys that is printed.

B. Test Multi-Tasking

Sourcecodes: Camera_PC.c, Robot_PC.c, commander.cpp

In this problem, commander is the program that controls both Camera_PC.c and Robot_PC.c.

Robot_PC prints out the TMR command and Camera_PC.c captures an image and sends the image to commander.cpp. When commander.cpp receives an image, its thread displays the image on the SDL window. Commander.cpp takes in user input (raw), and when a user types a command, commander.cpp sends TMR commands to Robot_PC if the key is related to TMR and sends camera commands to Camera_PC.c if the key is related to Camera (p and v). When Robot_PC receives the command datagram from commander, it prints the command. This program is dummy program that will be implemented in part C. In part C, this program will parse the data and actually control the Robot using functions that I implemented in the previous labs.

I made two sockets in Commander.cpp: one for Camera_PC.c and one for Robot_Pc.c. These three programs transmit and receive data using UDP packets. Port number for Robot is 4960 and Robot number for Camera is 4950.

C. Integration

Sourcecodes: Codes that run on Bealgebone: Acquire_Triple_PWM.sh, Capture2.c, Send_UDPs, Control_TMR_Bone.c

Codes that run on PC: commander2.cpp

Acquire_Triple_PWM: a shell file that acquires the PWM. (used in lab3)

Capture2.c: This is the file that actually captures an image. It is called in Send_UDPs.c program to capture images.

Send_UDPs.c: This program calls capture2.c program to capture and send images to the receiving host every 1 second. It has an infinite loop that keeps sending the receiving host images every 1 sec. It calls the Capture2.c program by using system function:

system("./Capture2_Bone -M -c 1 -o > image.jpg"). (When Capture2.c is compiled, its name is given as Capture2_Bone) When the system call is run, an image is captured and saved with the name image.jpg. Then, I open the image using: fopen function with "

rb" parameter. I need to open the image with "rb" parameter because image files may contain

binary characters. After opening the image, I found the size of image by using fseek and ftell

functions. After, I send the image to the receiving host using UDP function sendto. Before

sending the image to the receiver, I sent the size of the image to the receiver first so that the

receiver can prepare a buffer to store the image. After sending the size of image, I created a

buffer that can store the image. Then, I perform the task of reading the image into the buffer and

sending the buffer to the receiving end at the time in a while loop. The while loop runs until the entire image is read. As a result, I could transfer an image that exceeds the maximum datagram size, which is $2^{16}-1$. After sending the image, I used `usleep` function with parameter 1000000 to stop the program for a second to prevent the overhead of the process. When you run this file, you need to give two parameters: IP address and port number of the receiving host.

Control_TMR_Bone: This program receives TMR commands and controls the Beaglebone's TMR. (Used in lab 3) This program runs infinite loop. Inside the infinite loop, it receives datagram from `commander2.cpp`. It behaves like UDP server. It creates UDP socket and binds the port number 4950 to it. As a result, `commander2.cpp` can connect to this program using the port number 4950. After receiving the message, it parses the message using `strtok` and stores the values in appropriate variables. There are 7 variables: `id`, `time_elapsed`, `vx`, `vy`, `vw`, `ll`, and `lr`. `ID` represents the command number, `time_elapsed` represents when the command is received, `vx` represents velocity in x axis, `vy` represents velocity in y axis, `vz` represents rotation velocity, `ll` represents left led's toggle state, and `lr` represents right led's toggle state. After storing the values, the exact same code from lab 3 is ran. It opens GPIO, converts `wv` to `wv`, and assigns run and duty value according to `wv`. For `ll` and `lr`, same code from lab 2 is ran. It opens GPIO for LED, and gives the value of 1 if the toggle is on, value of 0 if the toggle is off.

commander2.cpp: This is the ultimate program that combines every other programs. It takes IP address of Beaglebone as an argument. it is divided into two parts: main loop and threaded loop. The threaded loop takes in images and displays them. The main loop deals with TMR control. In main loop, it first initializes SDL and waits for SDL events. The SDL event is the keyboard input. q, w, e, a, s, d, z, x, and c.

Key	Action
q	left LED toggle
w	go forward (+vw)
e	Right LED toggle
a	go left (-vy)
s	stop
d	to right (+vy)
z	rotate counter clock wise
x	go backwards (-vx)
c	rotate clock wise
p	take picture
v	show video

When these keys are pressed, switch statement evaluates the input value and processes them.

This is implemented in lab 3.

P and v keys are treated differently from other keys. Keys other than p and v are related to TMR control. When p and v are press camera toggle is activated. Let me explain how p and v works after explaining the threaded fuction.

In threaded function, it makes UDP socket that Beaglebone's Send_UDPs can send images to. It binds port number 33834 to it. It first receives the size of the image before receiving the actually image. After it receives the size of the image, it makes a buffer that the image will be stored. The maximum size of the datagram is $2^{16}-1$. However, high resolution image can easily exceed that

size. As a result, we first check the size of the picture. If it exceeds the maximum size of the datagram, it receives the datagram in two chunks. As a result, it can resolve images with high resolution as desired. After receiving the image in to buffer that was created before, it opens a file with parameter "wb". Again, you need to open the file with b option because an image file can contain binary data. After opening the file, we write the buffer to the file using fwrite function.

Going back to the main function, when p is pressed, an image is stored to computer. An image named image.jpg is received from Beaglebone every one second. (the job of threaded function) I implemented the saving of an image by renaming this image to different name. There is a counter and each image is named differently with number concatenated to the name of the image. For example, the first image that is store is name image1.jpg, second being image2.jpg, and so on. When v is pressed, the images are shown on SDL. This is done by calling loadMedia function. When video is displaying, you need to keep displaying the image using function loadMedia. This is done by using infinite loop. As a result, there is a problem that you cannot control the robot inside this infinite loop. As a result, we, again, polled events inside this infinite loop. This makes the code lengthy but does a great job. After getting input from the user, commander2.c makes a message that will be sent to the Beaglebone. If the input is related to TMR, it is sent to Control_TMR_Bone. If the input is p or v, it is internet, and does not send message. The way to distinguish whether an input is related to TMR is the camera toggle. If the camera toggle is on, the command is related to camera (internal) and does not send message, and if the camera toggle is off, the command is related to TMR and message is formulate and sent to Control_TMR_Bone.c.

After running commander2.c on PC and Control_TMR_Bone and Send_UDP.c on Beaglebone, I could confirm that the program runs flawlessly. The robot properly moved according to the key input and image saving and video displaying worked flawlessly. Also, the video and movement worked at the same time. The images were clear and had no broken pixels because I sent the size of the image before I send the actual image.

D. System Enhancement

Source code: commander3.cpp (the source code is attached)

I wanted to make an exploration robot. As a result, I added two sensors: temperature sensor and light sensor for the enhancement. Also, I implemented controlling the robot using an Xbox 360 control that would allow me to control the robot more precisely and it would make it more fun to control the robot.

After a little bit of research, I found that I could use my Xbox 360 controller on SDL.

Therefore, I installed Xbox controller driver on my Linux laptop.

First, I added the repository that contains Xbox controller driver using command:

```
sudo add-apt-repository ppa:grumbel/ppa
```

After, I updated and installed Xbox controller driver by using command:

```
sudo apt-get update && sudo apt-get install xboxdrv
```

After installing Xbox controller driver, I was able to use my Xbox 360 controller on my Linux laptop.

Inputs from Xbox controller is handled by SDL_Event. In SDL_Event data fields, there are two fields that an Xbox controller can use: SDL_JoyAxisEvent and SDL_JoyButtonEvent.

SDL_JoyAxisEvent detects input from joystick. If you look at the image at the end, joystick corresponds to 3 and 1. I decided to move RoboCam using left joystick, which is 3. So, when I move my joystick to left, RoboCam also should move to left.

In SDL_JoyAxisEvent, there are two data fields that I had to worry about: axis and value. Axis corresponds to whether the movement was horizontal or vertical. Value corresponds to which direction and how strong you moved the joystick. If axis is 0, the movement of joystick is in x direction; if axis is 1, the movement of joystick is in y direction. For example, if you move the joystick to right, axis will be 0 and value will be positive. If you move the joystick to left, axis will be 0 and value will be negative. The chart for joystick movement is shown below:

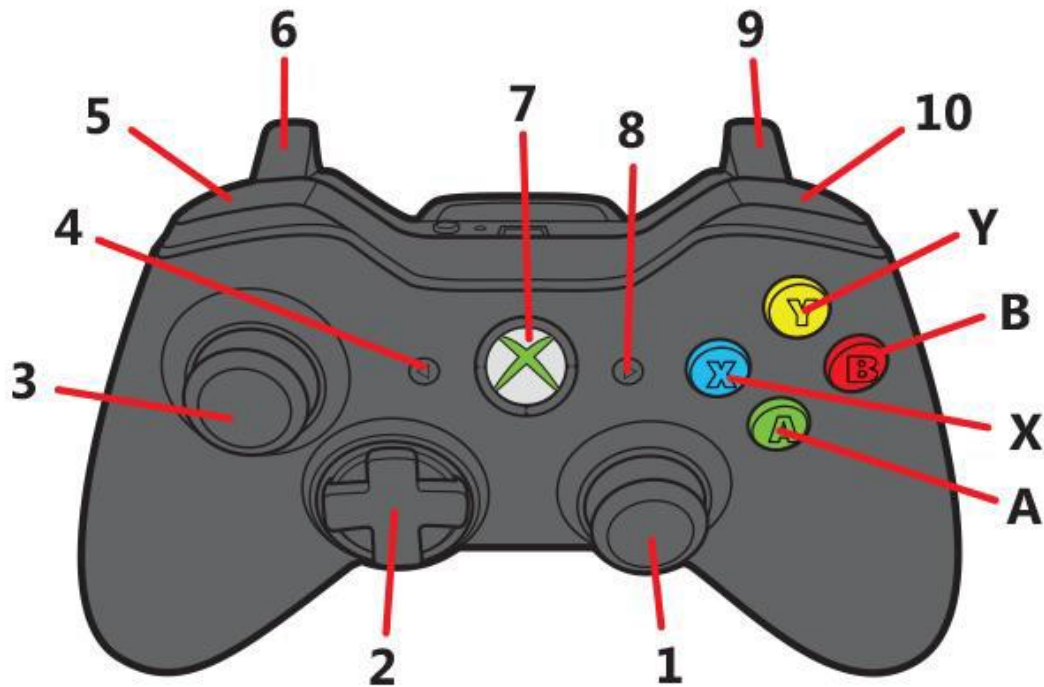
	-> (right)	down	<- (left)	up
direction	0	1	0	1
value	positive	positive	negative	negative

By examining the direction and value, I was able to know the movement of the joystick.

While SDL_JoyAxisEvent deals with joysticks, SDL_ButtonEvent deals with the buttons of the controller. There are many buttons on the controller, but I am only using 7 buttons: LT (6), LB (5), RT (9), RB (10), B, A, and Y. In SDL_ButtonEvent, the data field button is used to determine which button is pressed. The button values for the buttons are listed in the chart below:

	LT	RT	LB	RB	B	A	Y
button value (uint_8)	6	7	4	5	1	0	3

What I used for	Rotate CCW	Rotate CW	Left LED toggle	Right LED toggle	Stop RoboCam	Take a picture	Video toggle
-----------------	------------	-----------	-----------------	------------------	--------------	----------------	--------------



By examining the button value, I was able to assign tasks for each button.

When I integrated my code with this Xbox controller events, I had problem. The problem was that the joystick event happens too fast and loads many joystick events to stack. For example, when I move my joystick to left, more than 10 SDL_JoyAxisEvent are loaded into stack. I solved this issue by using states. I assigned states to each behavior. For example, I assigned LEFT state when I move the joystick to left. More than 10 events are going to be loaded into stack. When I poll events, if the polled event has same state as the previous polled event's event, I ignore the event.

After successfully implementing JoyAxisEvent and JoyButtonEvent, I was able to control RoboCam with my Xbox 360 controller.

For this enhancement, I upgraded commander2.cpp to commander3.cpp. Commander3.cpp is pretty much same as the commander2.cpp except that It implements the joystick control. I implemented this using by first calling SDL_Joystick for initializing the Joystick variable, called SDL_Init with SDL_INIT_JOYSTICK as a parameter to initialize the SDL Joystick, and used SDL_JoystickOpen function to use the joystick. After calling those three functions, I can now use the Xbox 360 controller in SDL. In commander2.cpp, the switch statement that checks the keyboard input is changed to code that checks the joystick input. Also, as I mentioned above, the states were added that deals with hundreds of joystick events.

For the temperature sensors and light sensor, I used python code to implement them.

I installed python library that controls Beaglebone's GPIO. After, I imported the library's ADC which control analog signals. Assigned pin for each sensors and ran infinite loops. In the infinite loops each pin reads in analog value from sensors. After reading the sensors, I converted the analog input to Celsius for temperature sensor and converted the analog input to scale (0-1) for light sensors. After converting the values, I printed the values to standard output stream.

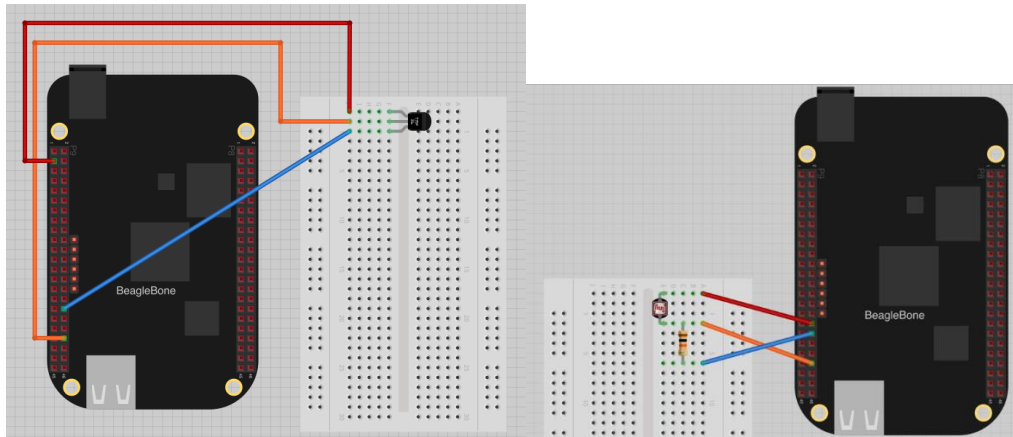
```

debian@beaglebone: ~
mv=729 C=22 F=73
mv=729 C=22 F=73
mv=729 C=22 F=73
mv=729 C=22 F=73
mv=729 C=22 F=73
mv=729 C=22 F=73
mv=731 C=23 F=73
mv=729 C=22 F=73
mv=729 C=22 F=73
mv=729 C=22 F=73
mv=729 C=22 F=73
mv=729 C=22 F=73
^CTraceback (most recent call last):
  File "tnp36.py", line 14, in <module>
    time.sleep(1)
KeyboardInterrupt
root@beaglebone:/home/debian/nfs_client/6/enhancements# python tnp36.py
mv=729 C=22 F=73
mv=729 C=22 F=73
mv=729 C=22 F=73
mv=729 C=22 F=73
mv=731 C=23 F=73
0.828556 1.477000
0.818889 1.477000
0.816111 1.469000
0.828556 1.477000
0.824444 1.464000
0.824444 1.464000
0.822778 1.461000
0.822778 1.461000
0.828000 1.476000
0.815556 1.468000
0.815000 1.467000
0.818556 1.459000
0.812222 1.462000
0.813333 1.468000
0.818333 1.473000
0.828556 1.477000
0.832222 1.490000
0.831667 1.497000
0.828000 1.476000
0.818111 1.469000
0.817222 1.471000
0.818333 1.473000

```

The left picture shows the temperature information in Celsius and Fahrenheit and the right picture shows the brightness information. You can find more information (result) of enhancement in discussion part.

The hardware wiring of sensors are shown below:



left: temperature sensor; right: photoresistor

The thing to note here is that the temperature sensor and photoresistor sensor use analog input of Beaglebone.

65 possible digital I/Os

P9				P8			
DGND	1	2	DGND	DGND	1	2	DGND
VDD_3V3	3	4	VDD_3V3	GPIO_38	3	4	GPIO_39
VDD_5V	5	6	VDD_5V	GPIO_34	5	6	GPIO_35
SYS_5V	7	8	SYS_5V	GPIO_66	7	8	GPIO_67
PWR_BUT	9	10	SYS_RESETN	GPIO_69	9	10	GPIO_68
GPIO_30	11	12	GPIO_60	GPIO_45	11	12	GPIO_44
GPIO_31	13	14	GPIO_40	GPIO_23	13	14	GPIO_26
GPIO_48	15	16	GPIO_51	GPIO_47	15	16	GPIO_46
GPIO_4	17	18	GPIO_5	GPIO_27	17	18	GPIO_65
I2C2_SCL	19	20	I2C2_SDA	GPIO_22	19	20	GPIO_63
GPIO_3	21	22	GPIO_2	GPIO_62	21	22	GPIO_37
GPIO_49	23	24	GPIO_15	GPIO_36	23	24	GPIO_33
GPIO_117	25	26	GPIO_14	GPIO_32	25	26	GPIO_61
GPIO_125	27	28	GPIO_123	GPIO_86	27	28	GPIO_88
GPIO_121	29	30	GPIO_122	GPIO_87	29	30	GPIO_89
GPIO_120	31	32	VDD_ADC	GPIO_10	31	32	GPIO_11
AIN4	33	34	GNDA_ADC	GPIO_9	33	34	GPIO_81
AIN6	35	36	AIN5	GPIO_8	35	36	GPIO_80
AIN2	37	38	AIN3	GPIO_78	37	38	GPIO_79
AIN0	39	40	AIN1	GPIO_76	39	40	GPIO_77
GPIO_20	41	42	GPIO_7	GPIO_74	41	42	GPIO_75
DGND	43	44	DGND	GPIO_72	43	44	GPIO_73
DGND	45	46	DGND	GPIO_70	45	46	GPIO_71

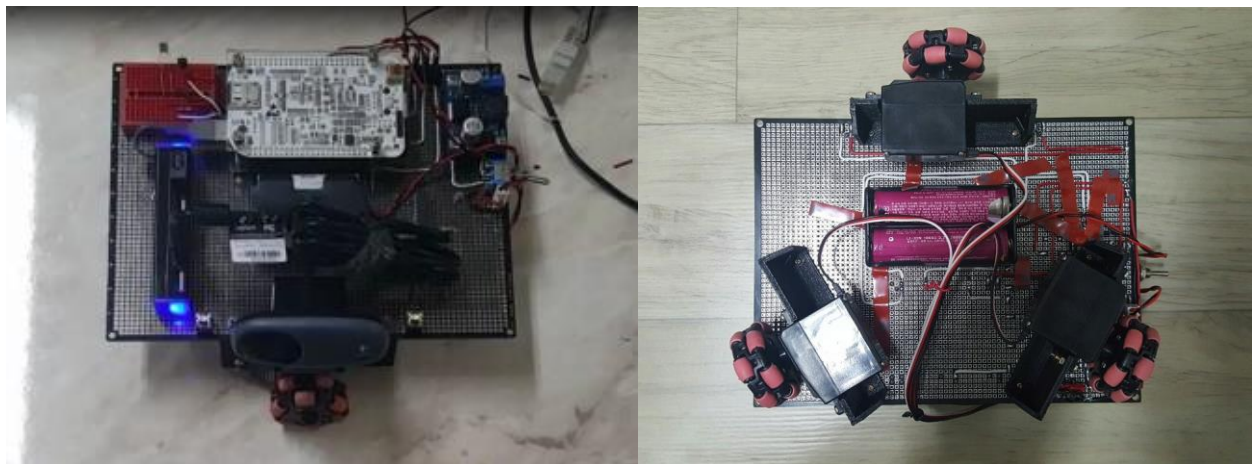
As you can see in the table above, there are 7 analog inputs GPIO on Beaglebone: AIN0, 1, 2, 3, 4, 5, and 6. These pins can read in analog signals. As a result, we used AIN1 for temperature signal and AIN3 for photoresistor.

Discussion

1) Discussion on result of integration:

In this lab, I integrated all the functions that I implemented in the previous labs. As a result of the integration, the Beaglebone can now run all the functions that I implemented in previous labs at

the same time. In lab 2, we controlled LED using resistors, Darlington transistor array, and LEDs. In lab 3, we controlled the TMR using batteries. In lab 4, we implemented wireless connection between Beaglebone and computer. In lab 5, we took pictures and videos using webcam. By integrating all these functions, we made an ultimate robot that can turn LEDs on and off, move and rotate, take pictures, and communicate with computer wirelessly. When implementing this ultimate robot, many things are used. Such as GPIO and unix file coding for LED and TMRs, socket programming (UDP) for wireless transmission, and video coding for taking pictures. As a result of integration, the robot can move, turn led on and off, take picture, and display video at the same time. The picture of our robot is shown below:



left: top view of the robot; right: bottom view of the robot.

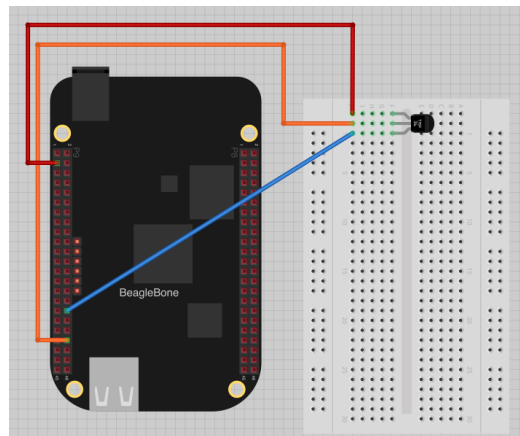
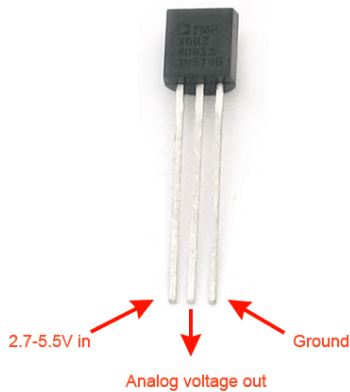
It is pretty hard to show the implementation of integration in pictures and in words. As a result, I am attaching videos of our robot.

2) Proposal for Enhancement:

In this lab, on top of the integration, I added some enhancement to the robot. I wanted to make an exploration robot and I added three things to our robot: temperature sensor (TMP 36), light

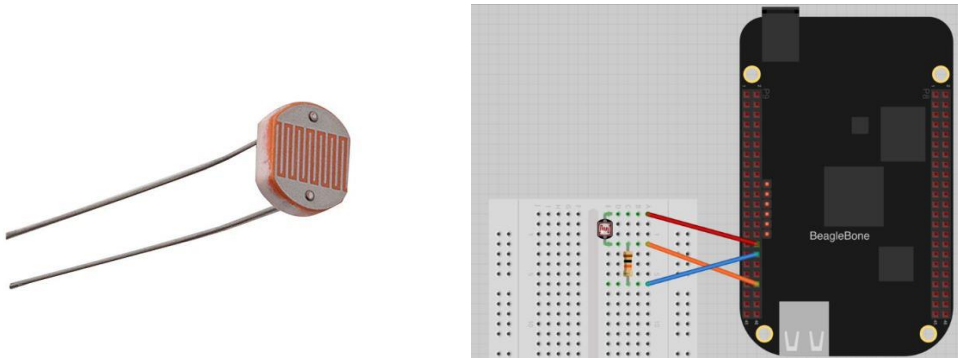
sensor (photo resistor), and joystick control. An exploration robot goes to dangerous places that humans cannot go such as a place that has a lot of radiation, a place that is on fire, or small crevices that humans cannot get into. At the dangerous places that humans cannot go, it fetches information of the place. I added two sensors for this task: temperature sensor and light sensor. So the robot uses those two sensors to get temperature information and the brightness information of the place. I also added joystick control feature to control the robot because it allows more fine control.

I used TMP 36 for temperature sensor because it is cheap and easy to implement and it looks like the picture below:



We provide voltage from Beaglebone to the leftmost pin of the TMP 36 sensor, connect ground to the right most pin, and connect analog input to the middle pin. Then, we get the temperature information from the middle pin in voltage. The analog input is scaled to a value between 0 and 1. If we multiply it by 1800, subtract it from 500, and divide it by 10, we get the temperature value of Celsius. We can do conversion to get the temperature in Fahrenheit.

I used photoresistor for light sensor. It is also cheap and easy to implement. A picture of it is shown below:



One end is connected to analog input and the other is connected to ground. When there is a lot of light, the resistance decreases and more voltage will be supplied to the analog input; when it is dark, resistance increase and less voltage will be supplied to the analog input. As a result, we can see the value of the analog input which is scaled to be a value between 0 and 1. We can do useful things with it. Such as if it is dark (say the value is less than 0.5), we can turn on the LED so we can take a better picture of the place.

Lastly, I added joystick control for the concise control of the robot. It is pretty hard to control the robot with keyboard because it is unfamiliar. However, with joystick, it is easy to visualize how the robot will move. As a result, I used an Xbox 360 controller for the robot control.



With this controller, I can move the robot with the left joystick and do various things with the buttons on the right and in the back. A button takes a picture, Y button pops up the video, B

button stops the robot, LB button toggles the left LED, RB button toggles to right LED, LT button rotates the robot counter clock wise, and RT button rotates the robot clock wise.

I implemented this using by first calling `SDL_Joystick` for initializing the Joystick variable, called `SDL_Init` with `SDL_INIT_JOYSTICK` as a parameter to initialize the SDL Joystick, and used `SDL_JoystickOpen` function to use the joystick. After calling those three functions, I can now use the Xbox 360 controller in SDL. There are two events that I used for control.

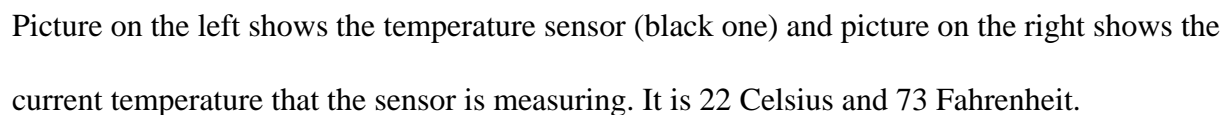
`SDL_JoyAxisEvent` for joysticks and `JoyButtonEvent` for all other buttons. For `JoyButtonEvent`, the variable `button` is the important and the only variable that I used. When I press a button, an event is created with `button` value that corresponds to the button that I pressed. The table of `button` value is shown below:

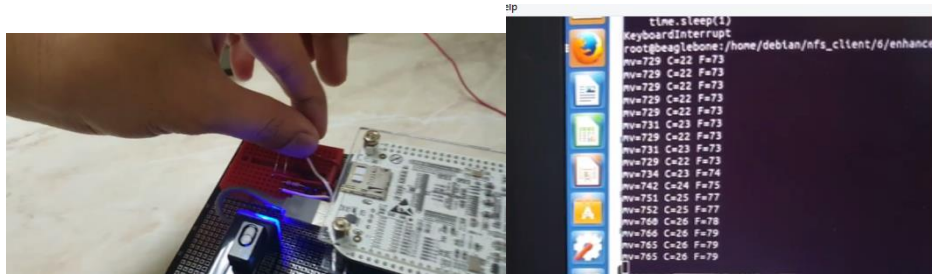
Button	Jbutton.button (button value)
A	0
B	1
X	2
Y	3
LB	4
RB	5
LT	6
RT	7

So, if I press button A, I get an `SDL_JoyButtonEvent` with the `Jbutton.button` value of 0. By looking up the table, I could assign appropriate behaviors when a button is pressed using if statements.

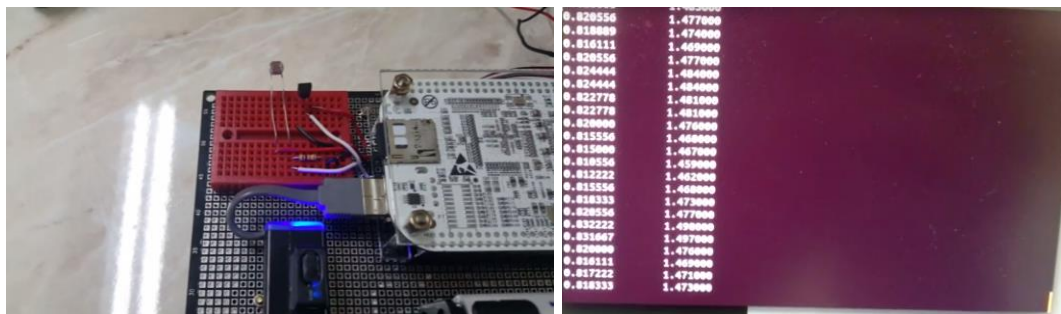
`SDL_JoyAxisEvent` is used for the movement of the robot. When I move the joystick, `SDL_JoyAxisEvent` is stacked inside the `SDL_Events`. There are two important variables in `SDL_JoyAxisEvent`: `axis` and `value`. `Axis` represents whether the movement is in X axis (value 0) or in Y axis (value 1). When I move the joystick to left and right, the value of `axis` is 0. When I

As a result of the enhancement, my robot can now measure temperature and brightness using TMP36 sensor and photoresistor. Also, I can control it more precisely using the Xbox 360 controller.

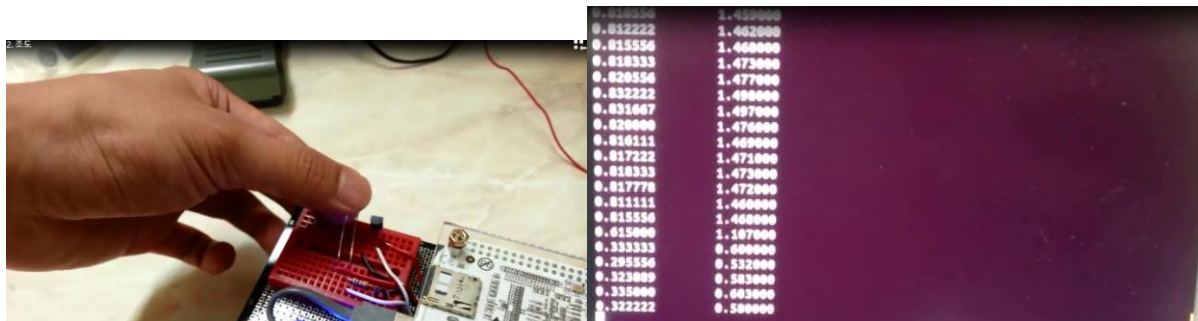




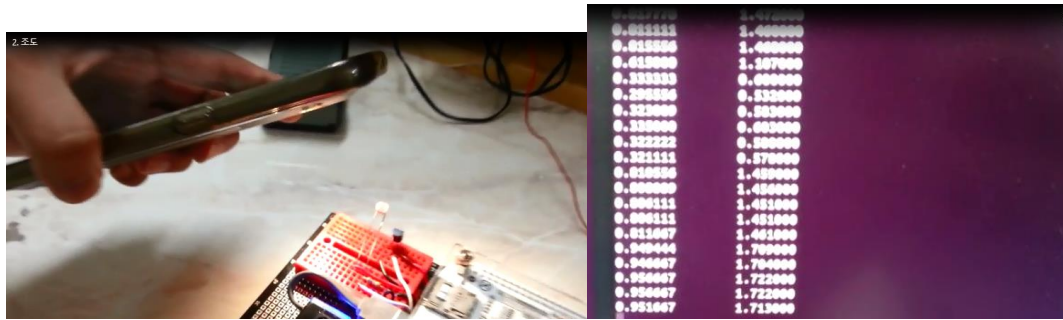
By touching the sensor, we could see that the temperature is increased to 26 Celsius and 79 Fahrenheit. We also tested decreasing of temperature by placing a cold drink next to the sensor and confirmed that the temperature sensor is working properly.



Picture on the left shows the photoresistor (tall one) and the picture on the right shows the temperature information. The left column shows the scaled voltage (range: 0-1) and the right column indicates the analog input voltage. As you can see in the picture, the value is about 0.82 which means that the room is pretty bright.



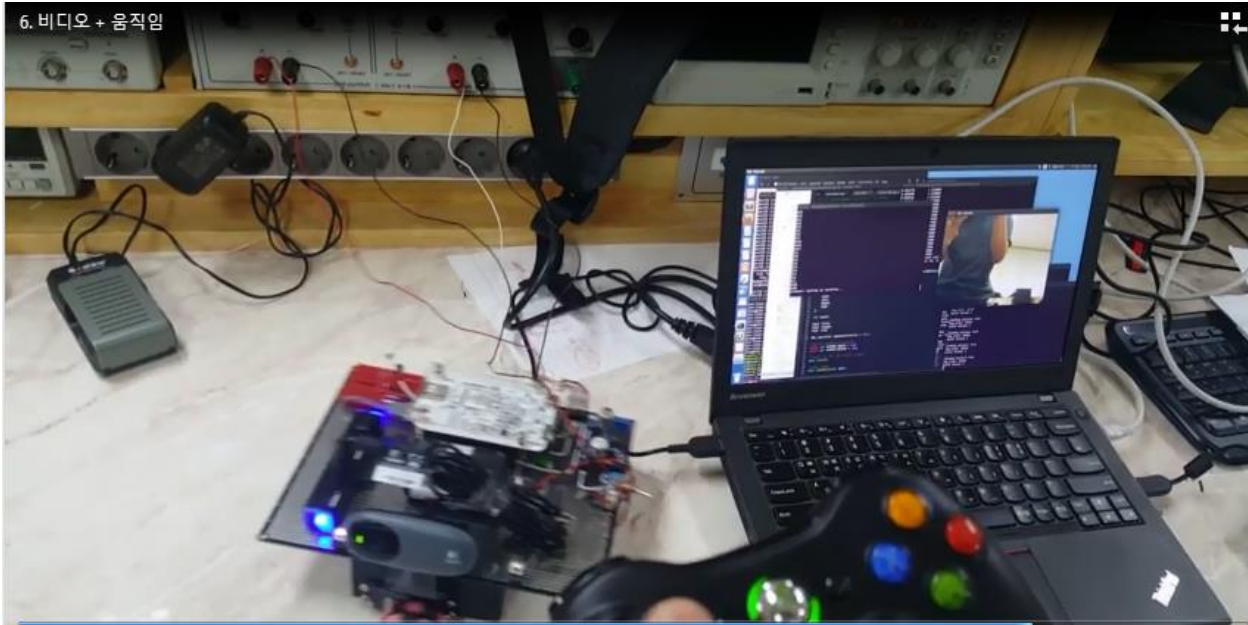
After covering the light sensor, we can see that the value decreased to 0.32.



After lighting the photoresistor with flash, we can see that the value increased to 0.95.

As a result, we could see that the light sensor is working properly.





Controlling the robot with the Xbox 360 controller worked flawlessly. It is pretty hard to show the movement of the robot using the Xbox 360 controller with images so please look at the videos that we attached.

4) Suggestion for RoboCam:

Overall, it was pretty good experience learning embedded system and actually implementing things that we learned. However, I think there is very little resources for students. For example, we do not know what the TAs want in the reports. I hope that there is a clear guideline for grading. I think it would be also good to return the score and feedback to students early so that students can learn what they did wrong and correct them for the next report.

DesignLab_RoboCam_Lecture6_E.pdf

DesignLab_RoboCam_Lecture6_E.pdf

DesignLab_RoboCam_Lab6_E.pdf

http://www.analog.com/media/en/technical-documentation/data-sheets/TMP35_36_37.pdf

<https://en.wikipedia.org/wiki/Photoresistor>

<https://www.libsdl.org/release/SDL-1.2.15/docs/html/joystick.html>