KOBUKIKART

EECS 149/249 Project Milestone Report

Morgan Fong, Nikhil Gahlot, Rohan Konnur
UC Berkeley

11/21/17

ABSTRACT

The motivation and goal for our project was the bring to life some of our favorite childhood games. This project designs and implements a real-life version of a Mario Kart race. Mario Kart involves different users racing their own kart's around a track track as fast as possible. In our version, the race cars will be represented by Kobukis, the player controls will be represented by WiiMotes, and the track and obstacles will be represented by tape. Each player controls his/her own race car (Kobuki) via a WiiMote steering wheel as they race around a predefined track. The track contains tape of different colors that represent different power-ups in Mario Kart (bananas and mushrooms). We use light sensors to recognize the different obstacles, the track, and the finish line. We introduced realistic game sound effects and costumes in order to better replicate the Mario Kart experience.

NOTES

We stuck pretty close to what we had initially decided on building from our original project charter. The main change we have made is switching from a permanent track that would have had wooden elements on it, to having a track that is purely on the ground and guided by strips of tape. The color sensor that we planned on initially using for object recognition is also being used to determine when the robot goes outside the parameters of the tracks. This change was made with time restrictions as well as material considerations in mind.

PROJECT REQUIREMENTS

Each Kobuki-Wiimote pair will work in isolation from other Kobuki-Wiimote pairs. As shown in Figure 1, the Wiimote will communicate its accelerometer readings with the BeagleBone controller via Bluetooth. The BeagleBone will also receive input from the light sensors which will indicate either hitting the edge of the track or detecting an item. Based on these inputs, the BeagleBone will send commands to the Kobuki.

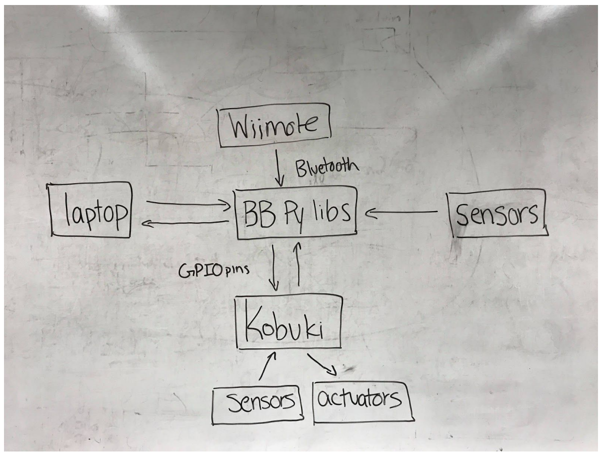

Figure 1: Software stack

Figure 2 shows the high-level view of how the Kobuki will interact with the real world as it drives. Within the Drive state, the Kobuki will react to user input (ie. Wiimote control). Upon detecting an item (banana or mushroom), it will handle each item's respective action, then return to the drive state. Upon detecting the track edge, it will perform a simple obstacle avoidance, then return to the drive state.

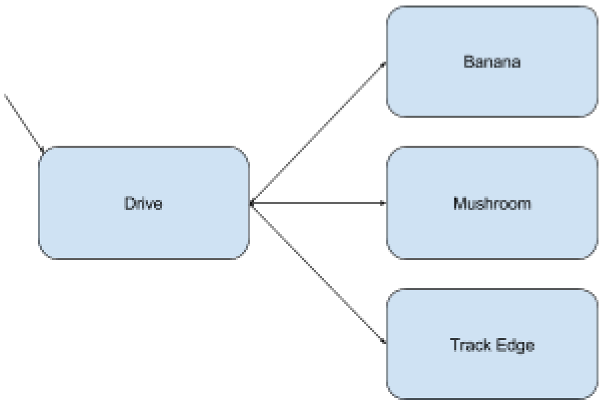

Figure 2: High-Level Kobuki Control Schema

SOFTWARE

BLUETOOTH

The bluetooth portion of the project was done with two key libraries which interfaced with 1) the BeagleBone side of the bluetooth connection and

2) the Wiimote side of the connection.
The bluetooth library which worked with the BeagleBone side of the handshake was called BlueZ. This API allowed the bluetooth to be enabled on the BeagleBone and then to connect to any open signals in the nearby area. This made the process very simple as a lot of the manual work was handled. On the BeagleBone, we wrote a python script which initiated the connection between the BeagleBone and the Wiimote and then proceeded to collect data from the Wiimote. The data was given in three values in terms of the "x", "y", and "z" axis of the Wiimote. We took these numbers and found a single unifying value that represented the angle of the Wiimote. This was done by assigning weights to the different axes based on the amount they changed as the Wiimote rotated through its rotation angle. These values turned out to be 0, 1, and 0 for "x", "y", and "z" respectively. We then found that the unifying value was between 100 and 150 and we wanted to find a single turning angle for the amount that the Wiimote was turned. We found an affine model as such:

$$y = 3.6*x - 360$$

This gave us a "real" turning angle between 0 and 180 degrees. The next step was to use this turning angle to actual turn the car and that was done by weighting the turning wheel with the turning angle so that the car would turn proportional to how much we turn the Wiimote. The library that worked the bluetooth connection on the Wiimote side was the Cwiid library which was built for interfacing Wiimotes with microcontrollers over bluetooth. This again did a lot of the heavy lifting for the connection and we decided on a scheme where the Wiimote would go forward when the "A" button is pressed and backwards when the "B" button is pressed. This is all done by recording the state using the functions as:

wm.state['buttons"] or wm.state["acc"][0, 1, or 2]

These values are recorded at every time step and an action is made based on this.

DRIVE CONTROL

The drive control program is written in Python and run on the BeagleBone Black via an Ubuntu operating system. Communication with the Kobuki was done through a UART port with the following specifications: *115200 BPS, Data bit: 8 bit, Stop bit: 1 bit, No Parity* (1). We chose to use the Adafruit BBIO library to simplify communication because it is written in python and is very well documented (2). With the library, writing a data packet to a serial port can be simplified into one line:

ser.write(<byte array data packet>)

Drive control for the Kobuki is based describing the right and left wheel speed of the Kobuki. With the right and left wheel speed, we calculate the turning radius and speed necessary to achieve the desired action. The speed and radius are then transformed into the correct data-packet format, as described in the Kobuki User Guide. For the specific calculations, we relied heavily upon the calculations done by the TA's in their lab files (3).

The drive control is created with a polling-based scheme to gather user input. For each iteration of our main drive loop, the drive program polls the bluetooth interface for a user input. If a user-input is present, the drive controller translates the accelerometer data into the proper format, as described above. If there is no data present, the Kobuki maintains course by transmitting the last data that was present.

LIGHT SENSOR

Item detection is currently implemented through a TCS34725 light sensor and programmed via the existing Python library by Adafruit (4). The Python library provides a simple abstraction to retrieve the color values sensed, thus the calibration of the light sensors can simply record the "magnitude" of the three color values: red, blue, and green.

$$|color| = \sqrt{r^2 + g^2 + b^2}$$

Each time the sensor data is sampled, we can easily check if the data is similar to any of the recorded color values and detect an object.

HARDWARE

BLUETOOTH

The hardware for this part of the project was fairly standard and big picture. The main parts were the BeagleBone Black board and the Wiimote. There was no other hardware involved. We used a Bluetooth interface to connect and communicate between the two pieces of hardware.

KOBUKI / BBB

The Kobuki and BeagleBone Black are interfaced via a UART Connection. The TX/RX UART ports on the Kobuki are connected to pins 24/26 on the BeagleBone Black (UART1). The BeagleBone Black is powered by a separate 5V portable battery pack via a USB adapter. This was done because the power output of the Kobuki did not match with the requirements of the board.

LIGHT SENSOR / I2C MULTIPLEXER

The light sensor (TCS34725) and I2C Multiplexer (TCA9548A) can both be directly wired to the BeagleBone Black via existing I2C bus pins. As the BeagleBone Black has only one I2C bus and the light sensor is only accessible through a single I2C address, an I2C Multiplexer is needed to enable access to multiple light sensors on the same controller for more fine-grained object detection. The integration of the light sensors and multiplexers with the BeagleBone Black is still a work in progress as the parts only recently arrived.

NEXT STEPS AND FUTURE CONSIDERATIONS

The next steps for this project involve finishing off the bluetooth control by making the program start on a button press, rather than only running by running a python script on the BeagleBone. The steps after that include setting up the control for the car so that it is also no longer wired and can roam free. We want to integrate all the individual software and hardware components as well such as using the ProtoBuffs to communicate any of the C code needed to make the light sensors work and integrating the light sensors input into the control of the car. After that has been completed, the main technology for the project will have been completed.

The final step would then be to build a sample track with objects strewn across the field and to conduct a sample race across it. This will be the final test of whether or not the the project is a succes.

CONCLUSION

This project gave us a very real-life experience at designing and implementing a project. From creating a project charter, to setting project milestones and dealing with setbacks, we learned a great deal about the challenges of real world development. Throughout the process we inevitably ran into setbacks with hardware components and software design choices, but due to our thorough planning and frequent revisions we were able to overcome these challenges.

ACKNOWLEDGEMENTS

Professor Edward A. Lee
Professor Prabal Dutta
GSI Meghan Clark
GSI Marcell Vazquez-Chanlatte
GSI Rohit Ramesh

PROJECT DEMONSTRATION

https://www.youtube.com/watch?v=ZuHS-MprLv4

REFERENCES

1.http://yujinrobot.github.io/kobuki/enAppendixProtocolSpecification.html
2.https://learn.adafruit.com/setting-up-io-python-library-on-beaglebone-black/using-the-bbio-library
3.http://leeseshia.org/lab/
4.https://github.com/adafruit/Adafruit_Python_TCS34725/blob/master/Adafruit_TCS34725/TCS34725.py