G54UBI final report (2014/15)

Title: Automatic Toy Pickup Robot

Student ID: 4219232
Date: 2015/01/07

Summary

The aim of the project is to provide convenient tools for getting pieces of toys in areas where hands are hard to reach. The prototype would be a movable robot that can perform a range of different tasks based on the input from grove pi sensors. This version of prototype focuses on the following functionalities:

1. Collision detection: The robot could detect a collision between the robot and other objects.
2. Falling prevention: The robot could detect a potential cliff to avoid falling.
3. Exploration: The robot could be guided with light to explore specific areas.

The three functionalities have been well designed and successfully implemented using ultrasonic range sensors, light sensor, button and vibration sensor. However, this prototype cannot by itself perform a complete exploration and pickup task, e.g. move forward and backward, turn left or right, pick up object in front and etc. All unimplemented functionalities were performed by. The main focus would be collision detection, falling prevention, and exploration.

The rest of the report would firstly introduce the background and motivation of the project, some related work in the past; then, followed with design and implementation of the robot. Thirdly, tests and results would be presented and evaluated. All testing data and source code are available in appendix.

Background and motivation

The idea of the proposed project is to design and develop an automatic toy pick-up robot to help family pick up toys scattered throughout the house, especially in areas where hands are hard or dangerous to reach.

Using the system with the light mode, which would be introduced in the design chapter, is very interesting and attractive for children. By using this system, children would gradually get into the habit of storing toys away after playing.

To summarize, this system: 1, helps to protect children from picking up toys in dangerous areas; 2, alleviates the workload of storing toys.

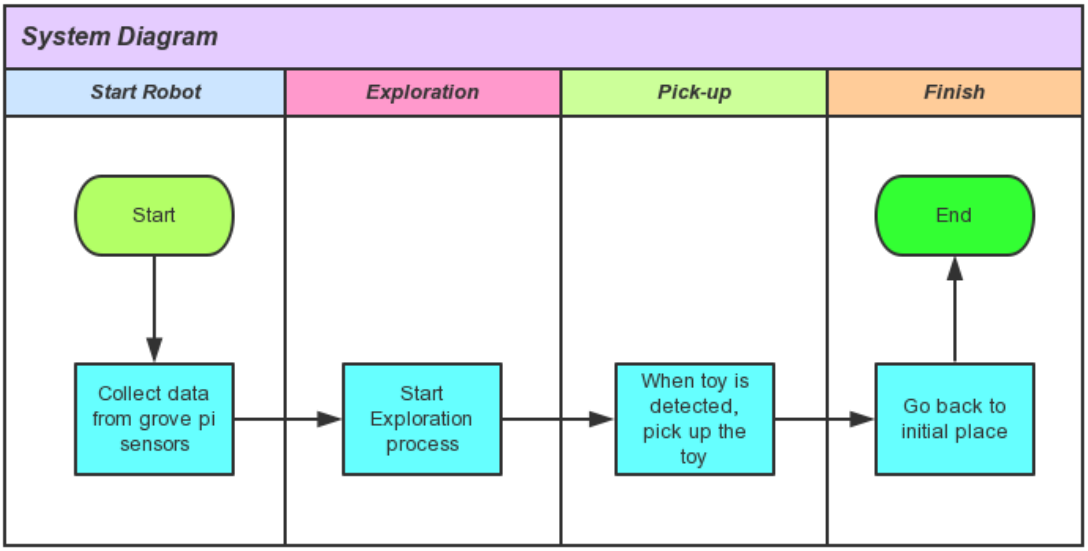
Related work

In terms of similarity, this auto toy pick-up robot shares lots of similarities with automatic floor cleaning machine. Although the goal differs, few functions are the same. According to Liu and Wang

(2013), the auto floor cleaning machine has utilized several emitters and photon detectors to detect obstacles in the field of emissions, so that the robot could be redirected to avoid the detected obstacles. Samsung has developed a system for the machine to locate itself efficiently using photos taken by the camera at the top of the machine. In order to perform similar detections, ultrasonic rangers were used by Carullo (2001). Inspired by Khing(2000), several ultrasonic rangers could be combined to monitor key values of the environment. In the UBI course, a template coursework introduced in the lecture utilized light sensor to detect the darkness of lab. Similarly, a light sensor was used in this project to determine the work region for the robot.

Design

System Design

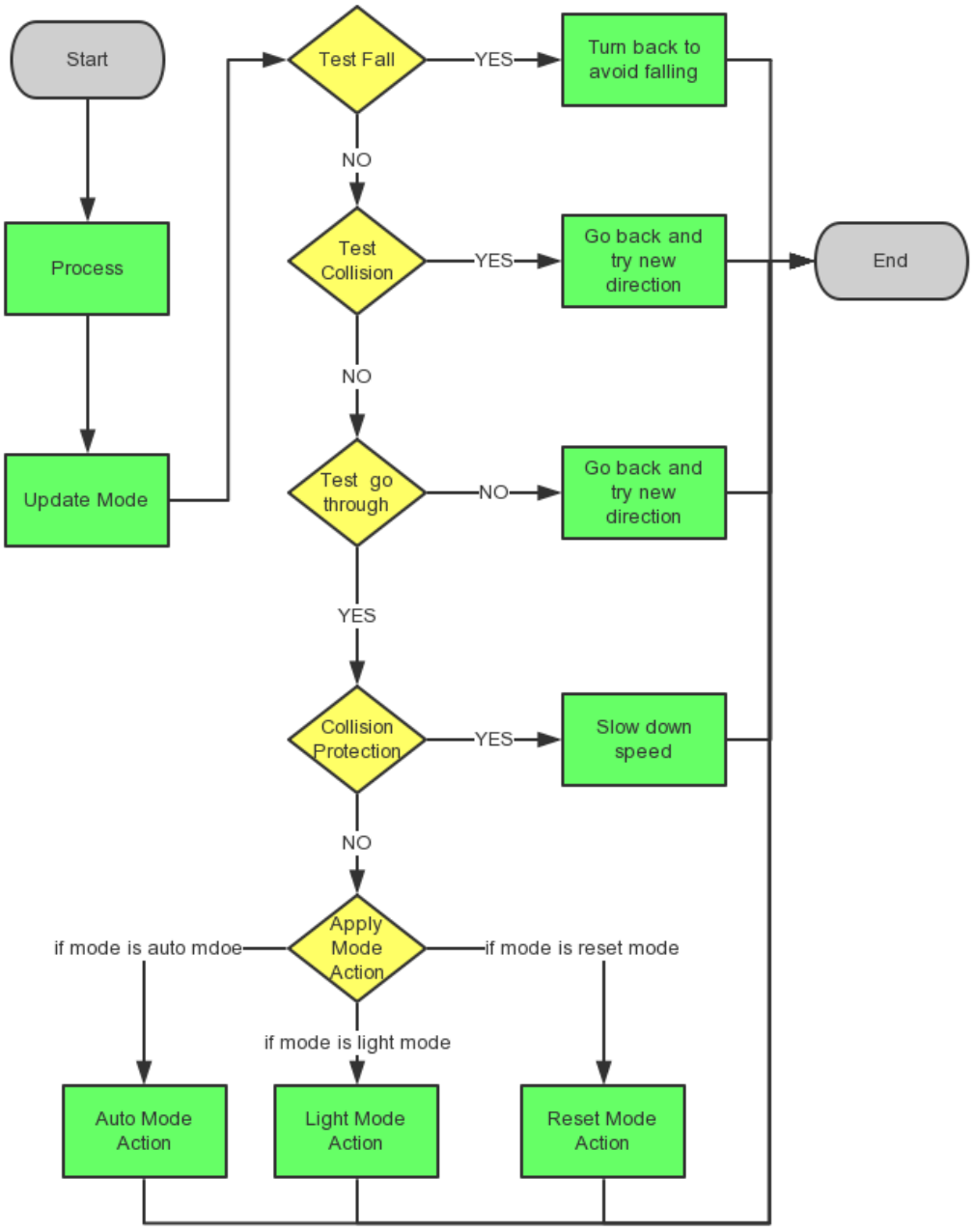


Graph 1: system design

At each iteration, the robot would receive a set of data collected by the prove pi sensors and then react to the environment accordingly. When the toy is detected, the robot would automatically pick up the toy and return to the initial place. This project would mainly focus on the Exploration function.

The system contains 3 different modes in exploration phase: auto mode, light mode and reset mode. Auto mode is the default mode for the robot to explore the environment; when light mode is on, the system would allow user to use flashlight to guide the robot to specific place; when the reset mode is on, the system would automatically navigate to the initial place. When the robot is turned on, the mode is by default the auto mode. User could press the button on the robot to change the mode to light mode in order to guide the robot to specific area. When the guidance is done, the robot would automatically switch to auto mode to start performing tasks.

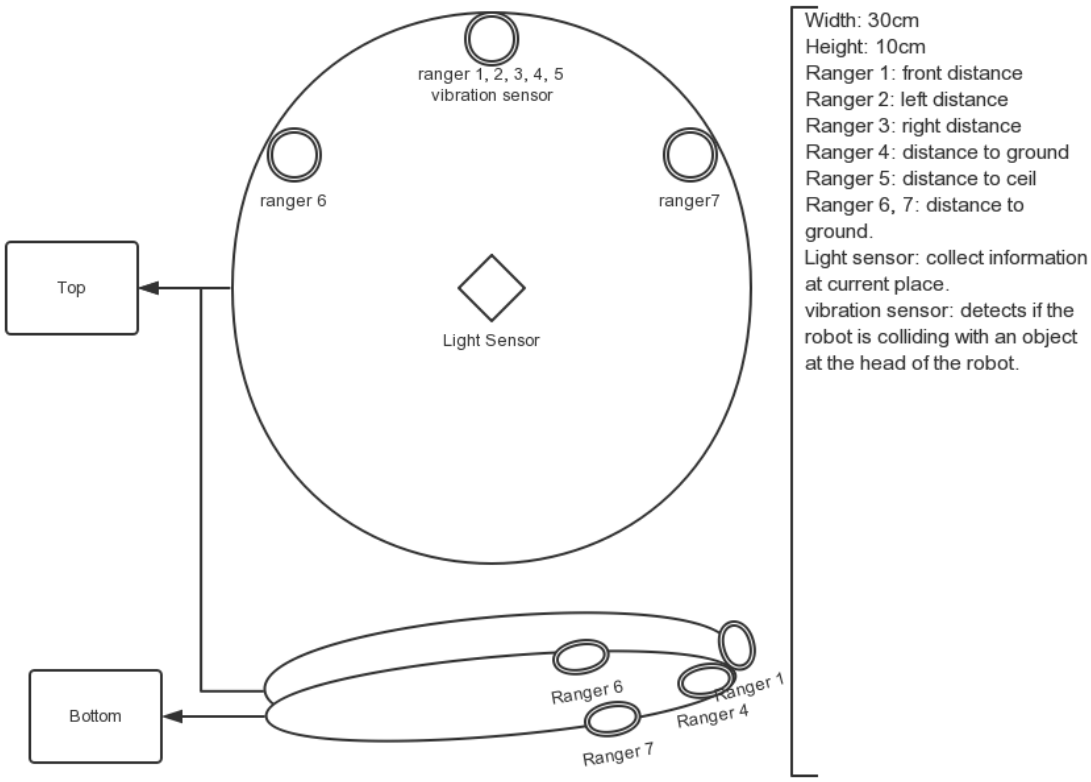
Exploration Design



Graph-2: implemented robot workflow

The workflow of the automatic pick-up robot is presented in graph-2. The robot would receive all necessary input at the initial state. When the initialization process is finished, the robot would update its mode accordingly. Then a sequence of tests would be performed: 1, Test fall, to test if the robot is going to fall if continues moving at the current direction; 2, Test collision, to test if the robot is colliding with other objects; 3, test go through, to test if the robot could pass the current path. A collision protection would be performed if there were near objects.

Robot Design



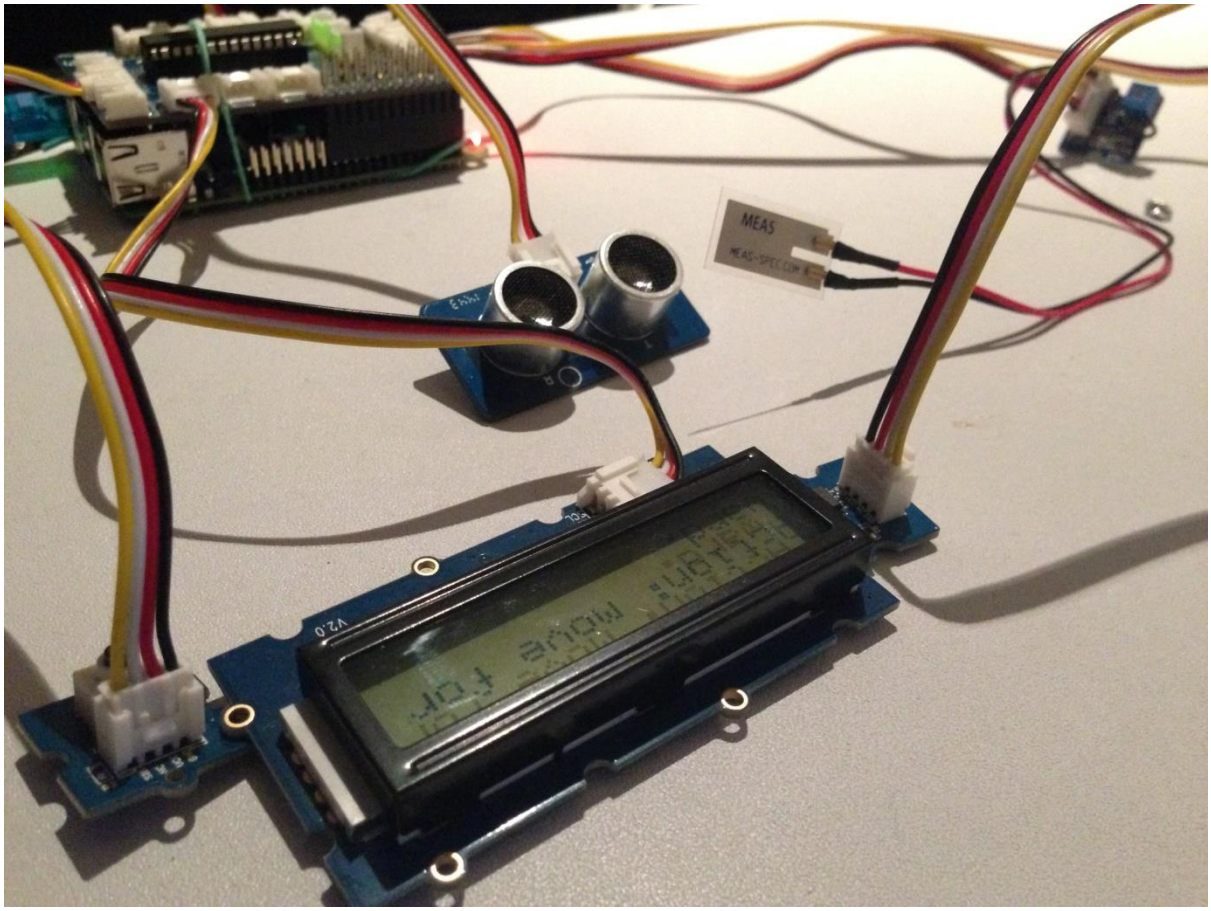
Graph 3: robot design

The design of the robot is presented as graph 3. The sensors are labelled as above. There is one light sensor at the top of the robot to collect light information. One vibration sensor is placed at the front of the robot to detect collisions with other objects. Three rangers (ranger 4, 6, 7) are place at bottom to detect the distance to ground. Ranger 2 and 3 would detect the distance to left and right respectively. All these sensors work together to provide information for the robot to react to the environment.

Implementation

Grove pi setup

The sensors used in this project were shown in the picture below. Due to the limitation of sensors, only one ranger was used to simulate all different situations. The sensors used are: ultrasonic ranger sensor, vibration sensor, light sensor, LCD display and button.



Grove pi data initialization

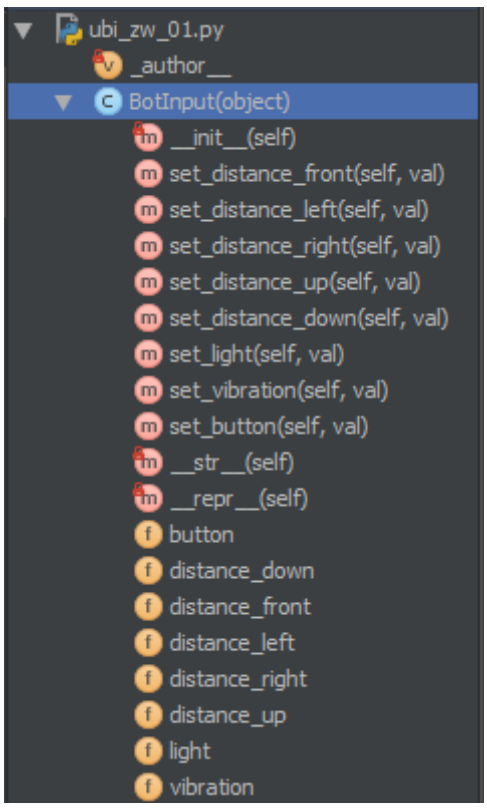
The first step of the system is to initialize sensors and get data from the sensors. As shown in the screenshot below. The program would read the data from sensors and store the data in object “BotInput”.

```
314  # initialize the sensors
315  sensors = {
316      "button": (SensorShim.DIGITAL, 8),
317      "light": (SensorShim.ANALOG, 0),
318      "vibration": (SensorShim.ANALOG, 1)
319  }
320  sensorObj = SensorShim(sensors)
321
322  def get_bot_input():
323      """
324      Get all necessary information from the grove pi sensors.
325      """
326      bot_input = BotInput()
327      bot_input.distance_front = ultrasonicRead(2)
328      #bot_input.distance_up = ultrasonicRead(3)
329      #bot_input.distance_down = ultrasonicRead(4)
330      #bot_input.distance_left = ultrasonicRead(5)
331      #bot_input.distance_right = ultrasonicRead(6)
332      bot_input.button = sensorObj.getValue("button")
333      bot_input.light = sensorObj.getValue("light")
334      bot_input.vibration = sensorObj.getValue("vibration")
335      return bot_input
336
337  def main():
338      bot = Bot()
339      frequency = 5
340      time_gap = 1.0 / frequency
341      while True:
342          # get all necessary data from grove pi and pass the information to bot for processing
343          bot.process_input(get_bot_input())
344          time.sleep(time_gap)
```

```
class BotInput(object):
    """
    This class is designed to hold input from grove pi.
    Particularly
    1, the distance values from rangers.
    2, light values from light sensor
    3, button values from button
    4, vibration values from vibration to detect collisions.
    """
    def __init__(self):
        self.distance_front = 1000
        self.distance_up = 1000
        self.distance_down = 0
        self.distance_left = 1000
        self.distance_right = 1000
        self.light = 1000
        self.button = 0
        self.vibration = 0

    def set_distance_front(self, val):
        """
        The value to represent the distance from bot to front object.
        """
        self.distance_front = val

    def set_distance_left(self, val):
        """
        The value to represent the distance from bot to left object.
        """
        self.distance_left = val
```



Data Processing

All source code is available at appendix. This chapter would mainly focus on explaining some important implementations.

General process

All grove pi information was stored in the object “BotInput”. At each iteration, the program would generate the “BotInput” object and pass it to “Bot” for processing using method “process_input”. The screenshot below showed that the only public method for the “Bot” class is the “process_input”, and all behaviours were defined inside the “Bot” class.

When the bot received the new input from grove pi, it would firstly store it into memory. The code was shown below. The comments showed the detailed process of the method “process_input”

```
124     def process_input(self, bot_input):
125         """
126         The only open method for the bot to process the input from grove pi.
127         the process could be generally described as:
128         1, keep log of the current values
129         2, the bot will update the mode
130             a, if the button is pressed
131             b, if the time for the current mode runs out.
132         3, detects if the bot is going to fall
133             a, if yes, the bot will turn back, and find a new direction to try.
134         4, detects if the bot is colliding with other objects
135             a, if yes, the bot will turn back, and find a new direction to try.
136         5, detects if the bot can pass through the path and move forward.
137             a, if no, the bot will turn back, and find a new direction to try.
138         6, detects if the bot is about to hit objects in front
139             a, if yes, the bot will slow down the speed.
140         7, apply actions according to current mode and input.
141         """
142         self.__update_log(bot_input)
```

As the comments indicated, the process procedurally checks all possible situations, and if any of the situations applied, a “handle” method would be called to handle such situation.

```
150         self.__update_mode(bot_input)
151 
152         if self.__test_fall(bot_input):
153             self.__handle_fall()
154             return
155         elif self.__test_collision(bot_input):
156             self.__handle_collision()
157             return
158         elif not self.__test_go_through(bot_input):
159             self.__handle_not_go_through()
160             return
161 
162         self.__apply_collision_protection(bot_input)
163 
164         if self.mode == Bot.MODE_AUTO:
165             self.__apply_auto_mode()
166         elif self.mode == Bot.MODE_LIGHT:
167             self.__apply_light_mode()
168         elif self.mode == Bot.MODE_RESET:
169             self.__apply_reset_mode()
170 
```

Method: update mode

This method considered two situations that would lead to changes in mode: 1, time limitation; 2, button pressed. The screenshot showed the code in detail.

```
188     def __update_mode(self, bot_input):
189         """
190         This method handles mode changes.
191         1, test the time, if longer than time limit, switch to other modes accordingly.
192         2, if the button is pressed, reset the time, and switch to other modes accordingly.
193         """
194         # calculate passed time
195         passed_time = timeit.timeit() - self.time
196         # test if the time is out of limit
197         if self.mode == Bot.MODE_LIGHT and passed_time >= Bot.TIME_LIGHT_LIMIT:
198             # change mode and reset values.
199             self.mode = Bot.MODE_AUTO
200             self.low_light = self.log[-1].light
201             print "<time out> Bot mode changed to", self.mode
202             self.__set_display_lcd("<time out> Bot mode changed to " + self.mode)
203             return
204         elif self.mode == Bot.MODE_AUTO and passed_time >= Bot.TIME_AUTO_LIMIT:
205             # change mode and reset values.
206             self.mode = Bot.MODE_RESET
207             print "<time out> Bot mode changed to", self.mode
208             self.__set_display_lcd("<time out> Bot mode changed to " + self.mode)
209             return
210
```

```
211         # test if the button is pressed
212         if bot_input.button and not self.log[-2].button:
213             # update the time
214             self.time = timeit.timeit()
215             # change mode according to current mode.
216             if self.mode == Bot.MODE_AUTO:
217                 self.mode = Bot.MODE_LIGHT
218                 self.low_light = self.log[-1].light
219             else:
220                 self.mode = Bot.MODE_AUTO
221                 self.low_light = self.log[-1].light
222             print "Bot mode changed to", self.mode
223             self.__set_display_lcd("Bot mode changed to " + self.mode)
224
```

Method: test go through

In the design, it was stated clear that the ultra-range sensors were set at the head of the robot, thus if the detected distance if longer than the radius of the robot, the robot would pass the path.

```
275     def __test_go_through(self, bot_input):
276         """
277         test if the bot can pass through the path
278         """
279         return bot_input.distance_left > Bot.RADIUS and bot_input.distance_right > Bot.RADIUS
```

Method: test fall

It is considered that if the detected distance from the bottom of the robot to the ground is longer than half height of the robot, the robot may not climb back from that place.

```
281     def __test_fall(self, bot_input):
282         """
283         test if the bot would fall if continue moving forward.
284         """
285         return bot_input.distance_down > Bot.HEIGHT/2
```

Method: test collision

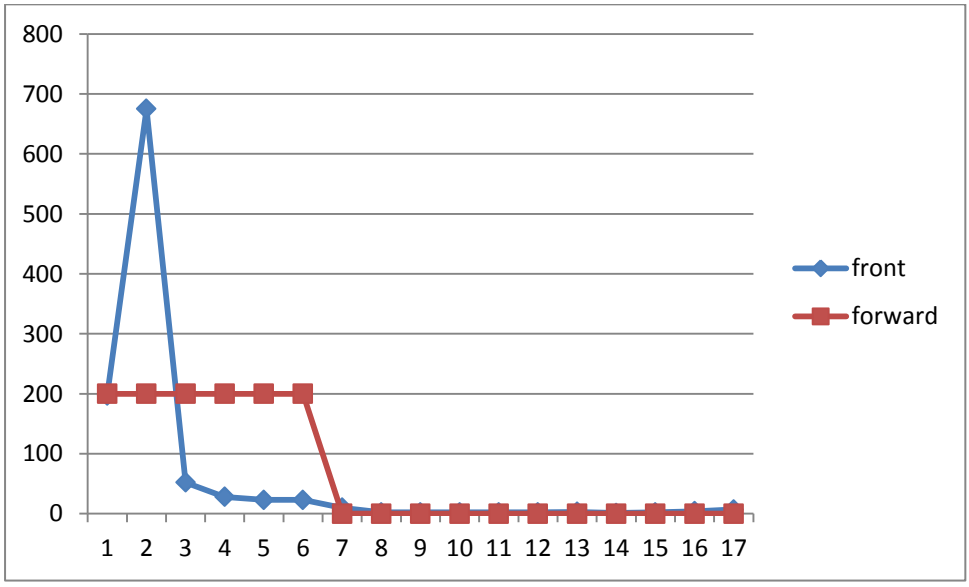
According to lots of experiments, the vibration sensor would generate high values, e.g. higher than 1000, when it was touched, therefore, the implementation defined a constant “Bot.COLLISION” as a threshold to determine if the sensor was touched. Due to the fact that the sensor was set at the head of the robot, if it was touched, it would indicate that the robot collided with some other object.

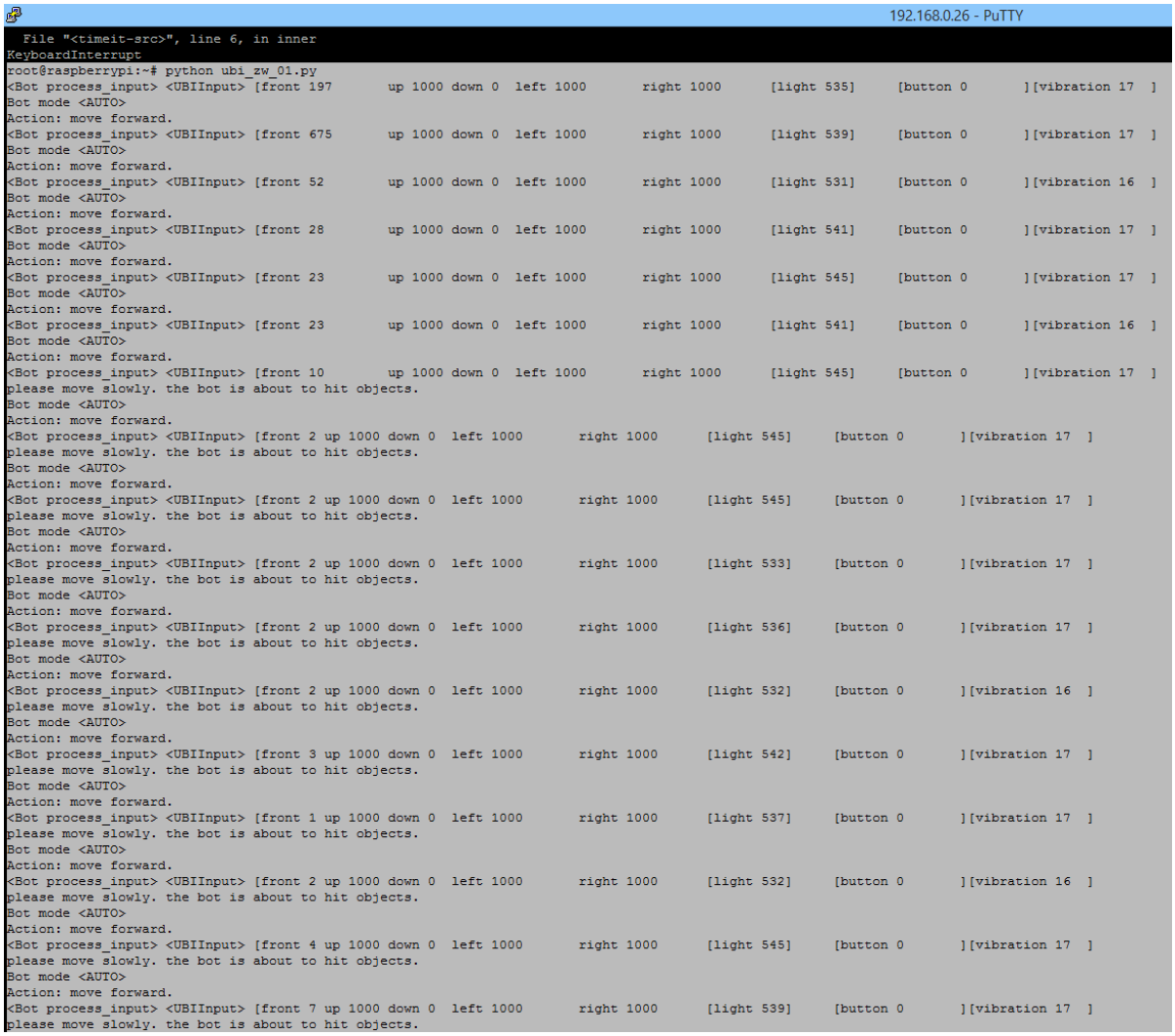
```
287     def __test_collision(self, bot_input):
288         """
289         test if the bot is colliding with objects.
290         """
291         return bot_input.vibration > Bot.COLLISION
```

Testing

Front distance test

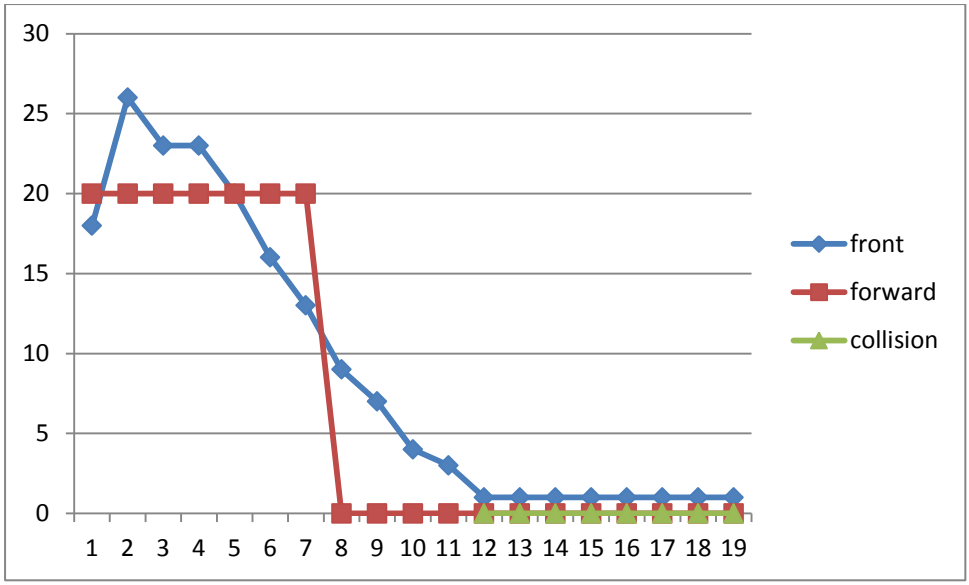
The front ultrasonic ranger is to detect the distance in the front. The purpose is to tell the robot to slow down the speed when there is near object. The simulation was done by holding the ultrasonic ranger, and moving it towards a wall. The screenshot shows the data and reaction of the robot. The relevant outputs were summarized in the chart below. There is one exception value in front at index 2, with value 675. As documented in the ultrasonic ranger, when exception happens, the output would be over 400 or negative. When the value of “front” drops below 10, the “forward” value becomes 0 indicating to slow down the speed to protect the robot.





Collision test

Collision detection is performed by vibration sensor. When the value exceeded a certain threshold, the system would generate a message indicating that the robot is colliding with objects. The simulation was conducted by moving the robot towards the wall, until it hit the wall. The data collected was presented in the screenshot. The value of “forward” drops to 0 when the front distance is below 10, indicating that the robot slowed down speed due to near obstacle. Collision took place when the front distance was 1, and was successfully detected.

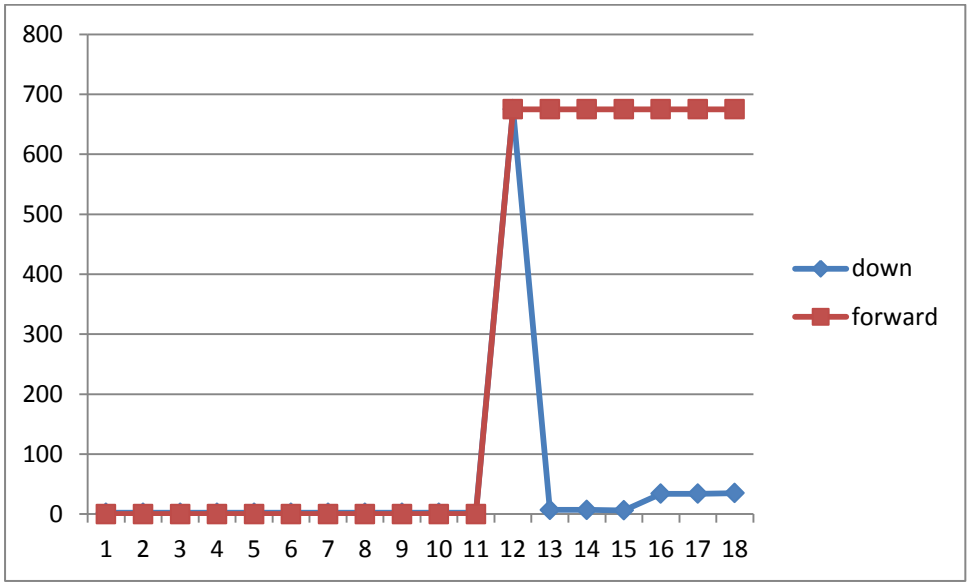


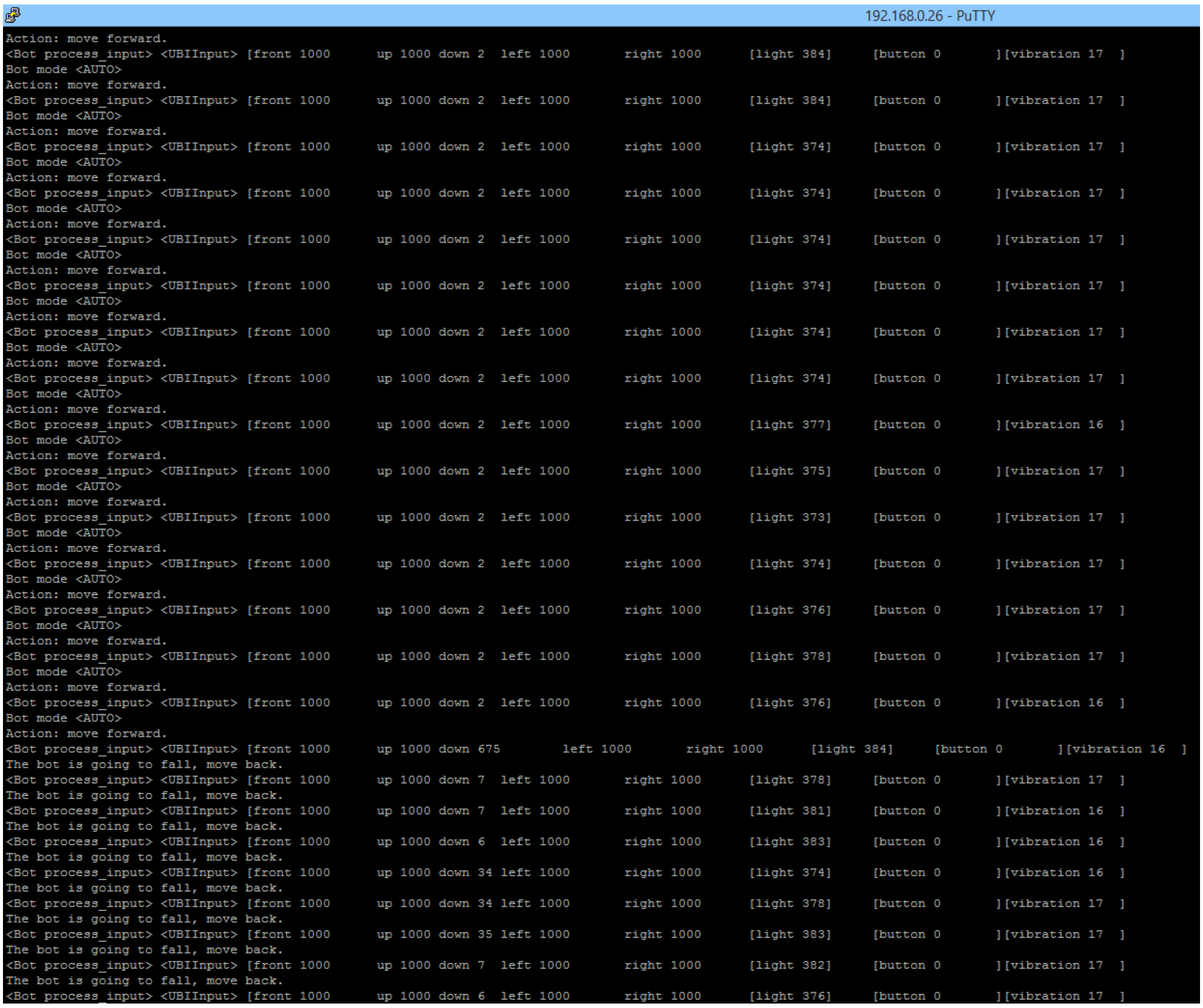
192.168.0.26 - PuTTY

```
Bot mode <AUTO>
Action: move forward.
<Bot process_input> <UBIInput> [front 18        up 1000 down 0  left 1000       right 1000      [light 381]     [button 0       ][vibration 17  ]
Bot mode <AUTO>
Action: move forward.
<Bot process_input> <UBIInput> [front 26        up 1000 down 0  left 1000       right 1000      [light 389]     [button 0       ][vibration 17  ]
Bot mode <AUTO>
Action: move forward.
<Bot process_input> <UBIInput> [front 23        up 1000 down 0  left 1000       right 1000      [light 386]     [button 0       ][vibration 17  ]
Bot mode <AUTO>
Action: move forward.
<Bot process_input> <UBIInput> [front 23        up 1000 down 0  left 1000       right 1000      [light 389]     [button 0       ][vibration 17  ]
Bot mode <AUTO>
Action: move forward.
<Bot process_input> <UBIInput> [front 20        up 1000 down 0  left 1000       right 1000      [light 382]     [button 0       ][vibration 16  ]
Bot mode <AUTO>
Action: move forward.
<Bot process_input> <UBIInput> [front 16        up 1000 down 0  left 1000       right 1000      [light 388]     [button 0       ][vibration 17  ]
Bot mode <AUTO>
Action: move forward.
<Bot process_input> <UBIInput> [front 13        up 1000 down 0  left 1000       right 1000      [light 381]     [button 0       ][vibration 17  ]
Bot mode <AUTO>
Action: move forward.
<Bot process_input> <UBIInput> [front 9 up 1000 down 0  left 1000       right 1000      [light 387]     [button 0       ][vibration 16  ]
please move slowly. the bot is about to hit objects.
Bot mode <AUTO>
Action: move forward.
<Bot process_input> <UBIInput> [front 7 up 1000 down 0  left 1000       right 1000      [light 382]     [button 0       ][vibration 17  ]
please move slowly. the bot is about to hit objects.
Bot mode <AUTO>
Action: move forward.
<Bot process_input> <UBIInput> [front 4 up 1000 down 0  left 1000       right 1000      [light 381]     [button 0       ][vibration 16  ]
please move slowly. the bot is about to hit objects.
Bot mode <AUTO>
Action: move forward.
<Bot process_input> <UBIInput> [front 3 up 1000 down 0  left 1000       right 1000      [light 383]     [button 0       ][vibration 17  ]
please move slowly. the bot is about to hit objects.
Bot mode <AUTO>
Action: move forward.
<Bot process_input> <UBIInput> [front 1 up 1000 down 0  left 1000       right 1000      [light 380]     [button 0       ][vibration 17  ]
please move slowly. the bot is about to hit objects.
Bot mode <AUTO>
Action: move forward.
<Bot process_input> <UBIInput> [front 1 up 1000 down 0  left 1000       right 1000      [light 386]     [button 0       ][vibration 16  ]
please move slowly. the bot is about to hit objects.
Bot mode <AUTO>
Action: move forward.
<Bot process_input> <UBIInput> [front 1 up 1000 down 0  left 1000       right 1000      [light 379]     [button 0       ][vibration 1022        ]
The bot collides with other objects, turn back and select new direction.
<Bot process_input> <UBIInput> [front 1 up 1000 down 0  left 1000       right 1000      [light 384]     [button 0       ][vibration 22  ]
please move slowly. the bot is about to hit objects.
Bot mode <AUTO>
Action: move forward.
<Bot process_input> <UBIInput> [front 1 up 1000 down 0  left 1000       right 1000      [light 379]     [button 0       ][vibration 1022        ]
The bot collides with other objects, turn back and select new direction.
<Bot process_input> <UBIInput> [front 1 up 1000 down 0  left 1000       right 1000      [light 380]     [button 0       ][vibration 1022        ]
The bot collides with other objects, turn back and select new direction.
<Bot process_input> <UBIInput> [front 1 up 1000 down 0  left 1000       right 1000      [light 389]     [button 0       ][vibration 1022        ]
The bot collides with other objects, turn back and select new direction.
<Bot process_input> <UBIInput> [front 1 up 1000 down 0  left 1000       right 1000      [light 379]     [button 0       ][vibration 1023        ]
The bot collides with other objects, turn back and select new direction.
<Bot process_input> <UBIInput> [front 1 up 1000 down 0  left 1000       right 1000      [light 381]     [button 0       ][vibration 1023        ]
The bot collides with other objects, turn back and select new direction.
```

Cliff detection test

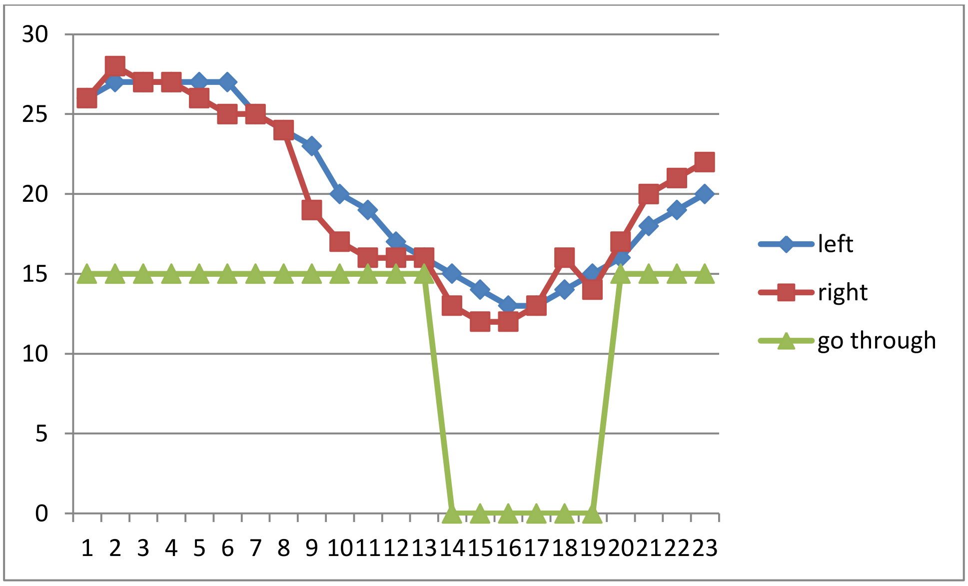
The cliff detection is realized by putting rangers on the bottom of the robot. If any of the rangers detected distances that was longer than a threshold, in this case 5, message “the bot is going to fall, move back” would be generated so the robot would turn back and avoid falling. The simulation is down by moving the robot on a table, from centre to edge. Once the edge was reached, the distance changed to 34 (the actual height of the table), and the warning messages were generated. Again, there is exception value at index 11 in “down”. It was caused due to the limitation of the sensor. The exception was well handled in the program.

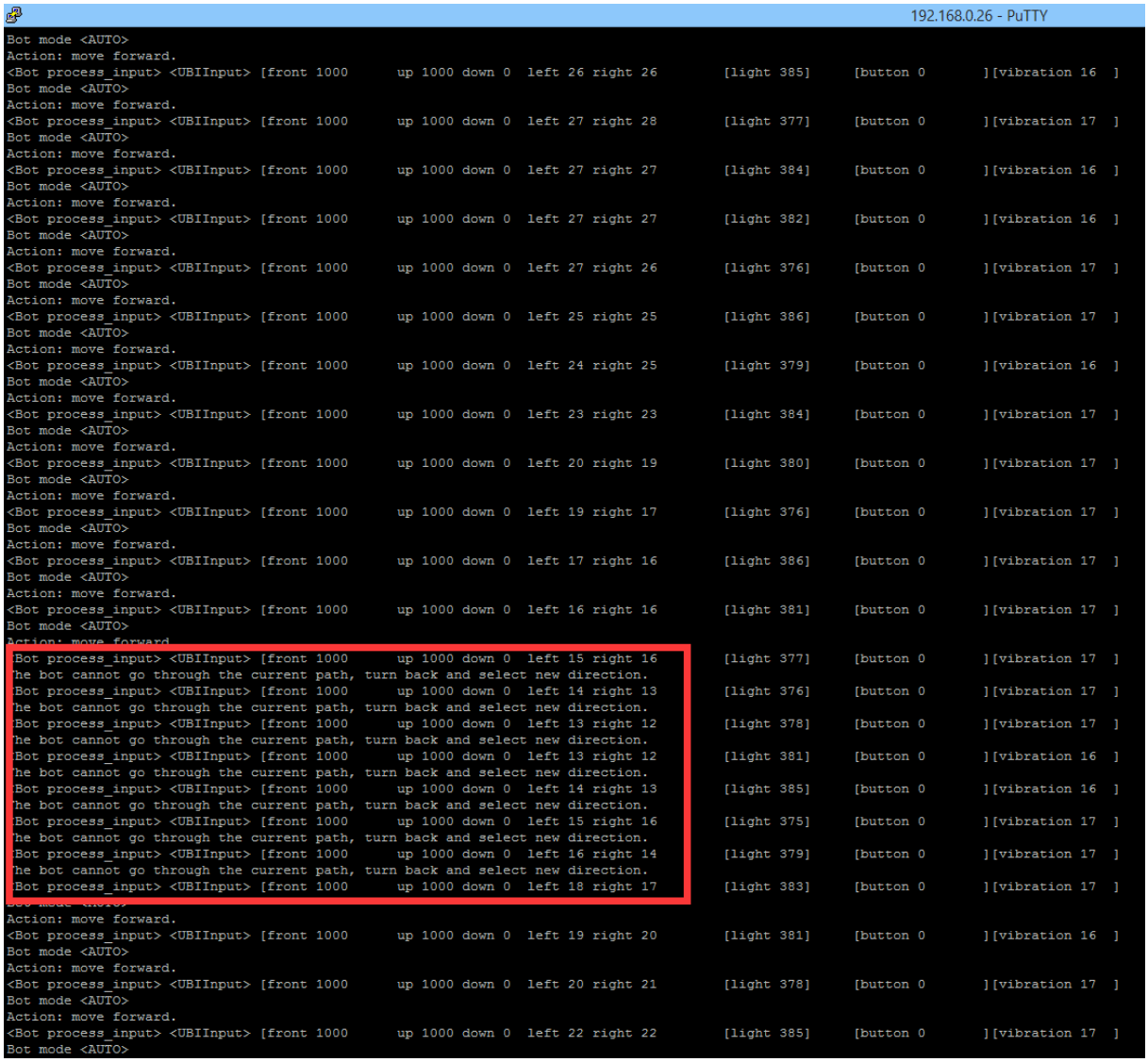




Go Through test

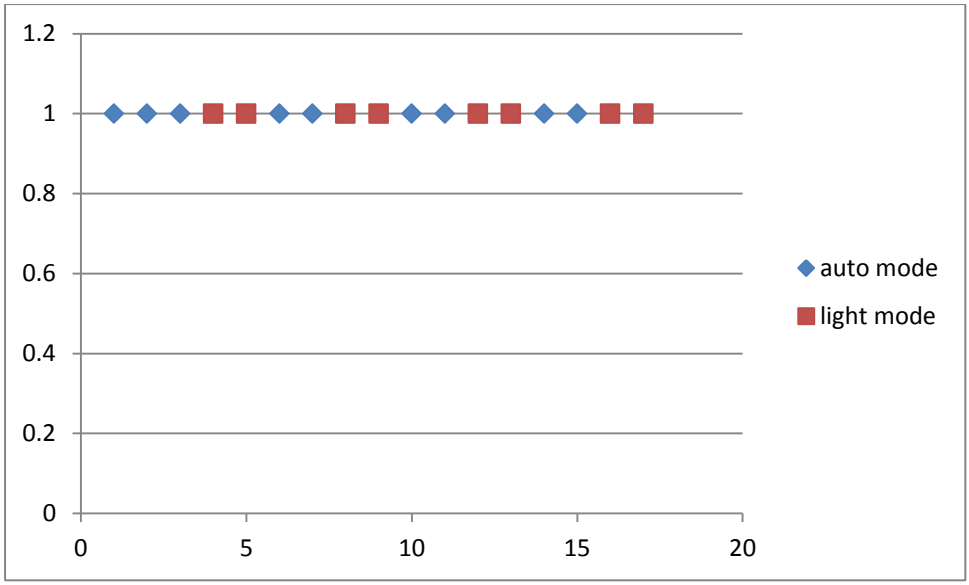
The simulation was done by putting obstacles on the path of the robot. The data was presented in the chart below. The 0 value in “go through” indicated that the robot cannot go through the path. The graph clearly showed that the when either one of the values in “left” or “right” bellows 15, the value in “go through” becomes 0, indicating that the robot cannot go through.





Change mode test

The button was pressed and released several times in this test. According to the chart, it is clear that the mode switch function was implemented efficiently.

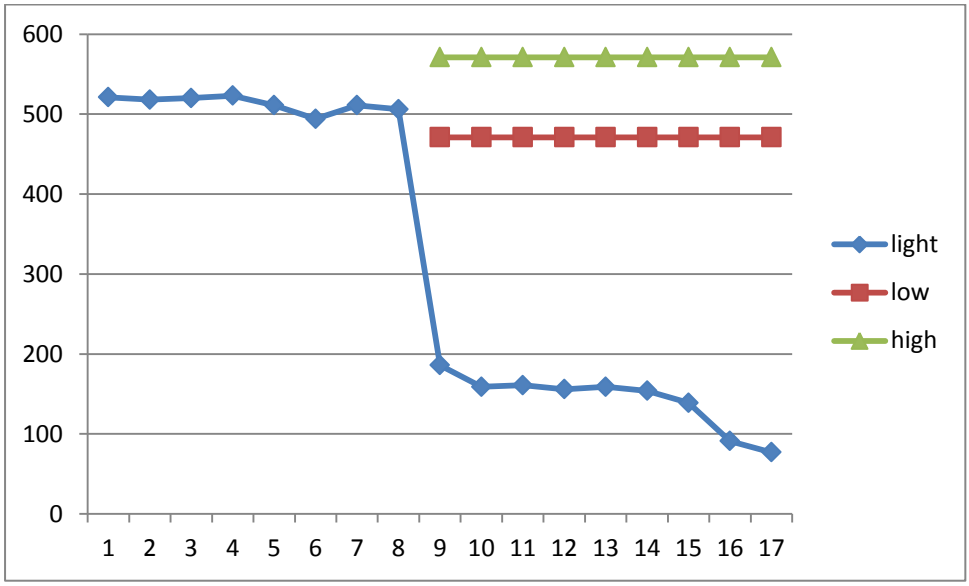


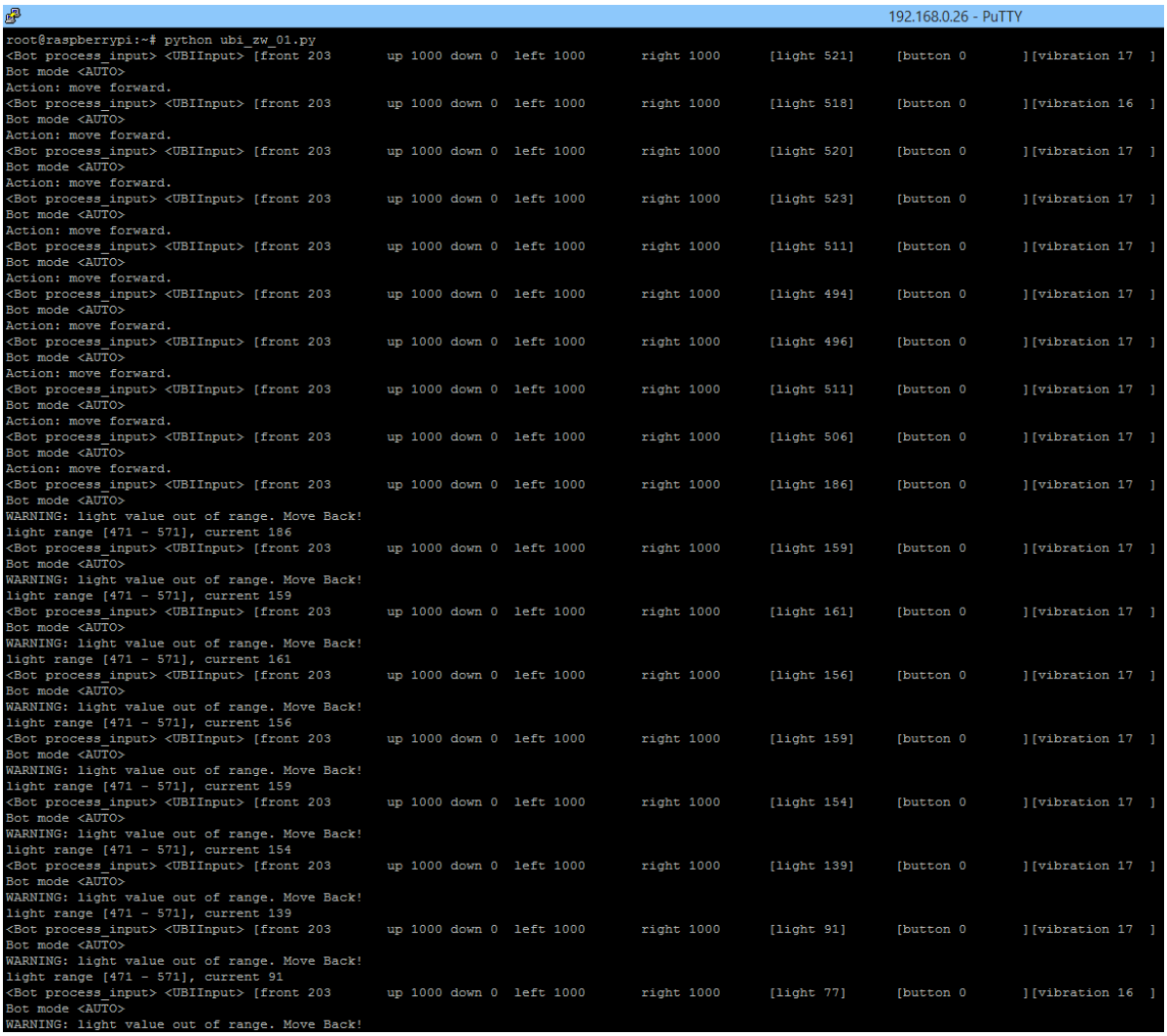
192.168.0.26 - PuTTY

```
root@raspberrypi:~# python ubi_zw_01.py
<Bot process_input> <UBIInput> [front 203       up 1000 down 0  left 1000       right 1000      [light 438]     [button 0       ][vibration 17  ]
Bot mode <AUTO>
Action: move forward.
<Bot process_input> <UBIInput> [front 203       up 1000 down 0  left 1000       right 1000      [light 437]     [button 0       ][vibration 17  ]
Bot mode <AUTO>
Action: move forward.
<Bot process_input> <UBIInput> [front 203       up 1000 down 0  left 1000       right 1000      [light 439]     [button 0       ][vibration 16  ]
Bot mode <AUTO>
Action: move forward.
<Bot process_input> <UBIInput> [front 203       up 1000 down 0  left 1000       right 1000      [light 428]     [button 1       ][vibration 16  ]
Bot mode changed to LIGHT
Bot mode <LIGHT> Action: move to find place with lower light value.
current light value: 428 low light value:  428
<Bot process_input> <UBIInput> [front 203       up 1000 down 0  left 1000       right 1000      [light 428]     [button 0       ][vibration 16  ]
Bot mode <LIGHT> Action: move to find place with lower light value.
current light value: 428 low light value:  428
<Bot process_input> <UBIInput> [front 203       up 1000 down 0  left 1000       right 1000      [light 437]     [button 1       ][vibration 17  ]
Bot mode changed to AUTO
Bot mode <AUTO>
Action: move forward.
<Bot process_input> <UBIInput> [front 203       up 1000 down 0  left 1000       right 1000      [light 432]     [button 0       ][vibration 17  ]
Bot mode <AUTO>
Action: move forward.
<Bot process_input> <UBIInput> [front 203       up 1000 down 0  left 1000       right 1000      [light 431]     [button 1       ][vibration 17  ]
Bot mode changed to LIGHT
Bot mode <LIGHT> Action: move to find place with lower light value.
current light value: 431 low light value:  431
<Bot process_input> <UBIInput> [front 203       up 1000 down 0  left 1000       right 1000      [light 429]     [button 0       ][vibration 17  ]
Bot mode <LIGHT> Action: move to find place with lower light value.
current light value: 429 low light value:  429
<Bot process_input> <UBIInput> [front 203       up 1000 down 0  left 1000       right 1000      [light 437]     [button 1       ][vibration 16  ]
Bot mode changed to AUTO
Bot mode <AUTO>
Action: move forward.
<Bot process_input> <UBIInput> [front 203       up 1000 down 0  left 1000       right 1000      [light 429]     [button 0       ][vibration 17  ]
Bot mode <AUTO>
Action: move forward.
<Bot process_input> <UBIInput> [front 203       up 1000 down 0  left 1000       right 1000      [light 435]     [button 1       ][vibration 16  ]
Bot mode changed to LIGHT
Bot mode <LIGHT> Action: move to find place with lower light value.
current light value: 435 low light value:  435
<Bot process_input> <UBIInput> [front 203       up 1000 down 0  left 1000       right 1000      [light 439]     [button 0       ][vibration 17  ]
Bot mode <LIGHT> Action: move to find place with lower light value.
current light value: 439 low light value:  435
<Bot process_input> <UBIInput> [front 203       up 1000 down 0  left 1000       right 1000      [light 431]     [button 1       ][vibration 17  ]
Bot mode changed to AUTO
Bot mode <AUTO>
Action: move forward.
<Bot process_input> <UBIInput> [front 203       up 1000 down 0  left 1000       right 1000      [light 432]     [button 0       ][vibration 17  ]
Bot mode <AUTO>
Action: move forward.
<Bot process_input> <UBIInput> [front 203       up 1000 down 0  left 1000       right 1000      [light 436]     [button 1       ][vibration 17  ]
Bot mode changed to LIGHT
Bot mode <LIGHT> Action: move to find place with lower light value.
current light value: 436 low light value:  436
<Bot process_input> <UBIInput> [front 203       up 1000 down 0  left 1000       right 1000      [light 434]     [button 0       ][vibration 16  ]
Bot mode <LIGHT> Action: move to find place with lower light value.
current light value: 434 low light value:  434
<Bot process_input> <UBIInput> [front 203       up 1000 down 0  left 1000       right 1000      [light 432]     [button 0       ][vibration 16  ]
Bot mode <LIGHT> Action: move to find place with lower light value.
current light value: 432 low light value:  432
<Bot process_input> <UBIInput> [front 203       up 1000 down 0  left 1000       right 1000      [light 440]     [button 0       ][vibration 17  ]
```

Auto mode test

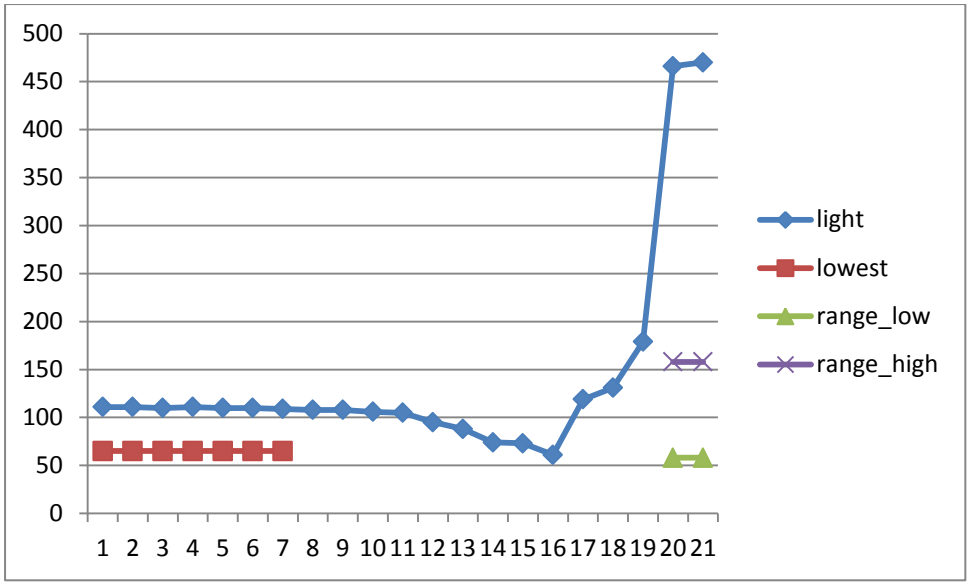
The auto mode test aims to test the behaviour of the bot with regard to light values. The screenshot revealed that the bot would generate message “WARNING…” when light value was out of range “[471 - 571]”. This was the expected result, proving the test was successful. The robot was designed to work in a certain range of light values according to initial light value when the bot was turned on. The purpose was to force it to work in a certain range to find pieces of toys. E.g. Under a sofa or bed.





Light mode test

In this test, the robot was set at the middle of the table at first place. Then a flashlight was turned on at the top of the robot. According to the message, the robot moved to find places with lower light value. Finally, the lowest value found was 65, and the final light value was 108. When the light mode ended, the robot switched back to auto mode, and started to perform exploration tasks. When the light exceeded 158, warning message was shown for the robot to move back.



192.168.0.26 - PuTTY

```
Bot mode <LIGHT> Action: move to find place with lower light value.
current light value: 111 low light value:  65
<Bot process_input> <UBIInput> [front 17        up 1000 down 0  left 1000       right 1000      [light 111]     [button 0       ][vibration 17  ]
Bot mode <LIGHT> Action: move to find place with lower light value.
current light value: 111 low light value:  65
<Bot process_input> <UBIInput> [front 17        up 1000 down 0  left 1000       right 1000      [light 110]     [button 0       ][vibration 17  ]
Bot mode <LIGHT> Action: move to find place with lower light value.
current light value: 110 low light value:  65
<Bot process_input> <UBIInput> [front 17        up 1000 down 0  left 1000       right 1000      [light 111]     [button 0       ][vibration 17  ]
Bot mode <LIGHT> Action: move to find place with lower light value.
current light value: 111 low light value:  65
<Bot process_input> <UBIInput> [front 16        up 1000 down 0  left 1000       right 1000      [light 110]     [button 0       ][vibration 17  ]
Bot mode <LIGHT> Action: move to find place with lower light value.
current light value: 110 low light value:  65
<Bot process_input> <UBIInput> [front 16        up 1000 down 0  left 1000       right 1000      [light 109]     [button 0       ][vibration 17  ]
Bot mode <LIGHT> Action: move to find place with lower light value.
current light value: 109 low light value:  65
<Bot process_input> <UBIInput> [front 203       up 1000 down 0  left 1000       right 1000      [light 108]     [button 1       ][vibration 17  ]
Bot mode changed to AUTO
Bot mode <AUTO>
Action: move forward.
<Bot process_input> <UBIInput> [front 17        up 1000 down 0  left 1000       right 1000      [light 108]     [button 0       ][vibration 17  ]
Bot mode <AUTO>
Action: move forward.
<Bot process_input> <UBIInput> [front 18        up 1000 down 0  left 1000       right 1000      [light 106]     [button 0       ][vibration 17  ]
Bot mode <AUTO>
Action: move forward.
<Bot process_input> <UBIInput> [front 17        up 1000 down 0  left 1000       right 1000      [light 105]     [button 0       ][vibration 17  ]
Bot mode <AUTO>
Action: move forward.
<Bot process_input> <UBIInput> [front 18        up 1000 down 0  left 1000       right 1000      [light 95]      [button 0       ][vibration 17  ]
Bot mode <AUTO>
Action: move forward.
<Bot process_input> <UBIInput> [front 203       up 1000 down 0  left 1000       right 1000      [light 88]      [button 0       ][vibration 17  ]
Bot mode <AUTO>
Action: move forward.
<Bot process_input> <UBIInput> [front 18        up 1000 down 0  left 1000       right 1000      [light 74]      [button 0       ][vibration 17  ]
Bot mode <AUTO>
Action: move forward.
<Bot process_input> <UBIInput> [front 15        up 1000 down 0  left 1000       right 1000      [light 73]      [button 0       ][vibration 17  ]
Bot mode <AUTO>
Action: move forward.
<Bot process_input> <UBIInput> [front 15        up 1000 down 0  left 1000       right 1000      [light 61]      [button 0       ][vibration 17  ]
Bot mode <AUTO>
Action: move forward.
<Bot process_input> <UBIInput> [front 17        up 1000 down 0  left 1000       right 1000      [light 119]     [button 0       ][vibration 17  ]
Bot mode <AUTO>
Action: move forward.
<Bot process_input> <UBIInput> [front 15        up 1000 down 0  left 1000       right 1000      [light 131]     [button 0       ][vibration 17  ]
Bot mode <AUTO>
Action: move forward.
<Bot process_input> <UBIInput> [front 16        up 1000 down 0  left 1000       right 1000      [light 179]     [button 0       ][vibration 17  ]
Bot mode <AUTO>
WARNING: light value out of range. Move Back!
light range [58 - 158], current 179
<Bot process_input> <UBIInput> [front 18        up 1000 down 0  left 1000       right 1000      [light 466]     [button 0       ][vibration 17  ]
Bot mode <AUTO>
WARNING: light value out of range. Move Back!
light range [58 - 158], current 466
<Bot process_input> <UBIInput> [front 15        up 1000 down 0  left 1000       right 1000      [light 470]     [button 0       ][vibration 17  ]
Bot mode <AUTO>
WARNING: light value out of range. Move Back!
light range [58 - 158], current 470
```

Critical reflection

The sensors are generally responsive at most of the time. However, in some experiments, when more than 5 sensors were used, especially more than 2 ultrasonic rangers, the program run slower

than expected. The solution was to set the frequency of the “read” method lower, alternatively to remove some sensors, to alleviate overload of the grove pi. Exception values from ultrasonic range sensor found were 675, followed with some negative numbers or numbers from 1 to 9. The solution in this project is that once value 675 was found, several following numbers smaller than 9 would be ignored.

The project has successfully designed and developed part of the automatic toy pick-up robot system. An interactive system for navigation was developed to detect common situations, including collisions with objects, cliff detection, determine if the robot could pass the current path or not and etc. All functionalities were thoroughly tested, and the results were satisfactory. However, such simple system may only be capable of handling simple situations. Further work is inevitably required in order to handle real world, complex environment problems. With the foundation of this project, a higher level deliberative system would be one of the possible choices. To enhance the navigation system, camera sensor could be used to take photos of the ceiling and divide search space into recognizable grid maps.

References

[1] Liu Kuotsan, Wang Chulun(2013). A Technical Analysis of Autonomous Floor Cleaning Robots. In European International Journal of Science and Technology, Vol. 2 No. 7

[2] Carullo A.; Parvis M. (2001). An Ultrasonic Sensor for Distance Measurement in Automotive Applications. In: IEEE SENSORS JOURNAL, vol. 1 n. 2, pp. 143-147. - ISSN 1530-437X

[3] H. Khing, C. Seng, & Z. Yi (2000). Multi-ultrasonic sensor fusion for mobile robots. Intelligent Vehicles Symposium, 2000. IV 2000. Proceedings of the IEEE.

Appendix A – instructions

Please connect ultrasonic rangers from D2 to D6. Connect Button to D8. Connect light sensor to A0. Connect vibration sensor to A1.

Source code

```
__author__ = 'Zongqi Wang'
from sensorshim import *
from grovepi import *
import time
import timeit
import grovelcd


class BotInput(object):
    """
    This class is designed to hold input from grove pi.
    Particularly
    1, the distance values from rangers.
    2, light values from light sensor
    3, button values from button
    4, vibration values from vibration to detect collisions.
    """
    def __init__(self):
        self.distance_front = 1000
        self.distance_up = 1000
        self.distance_down = 0
        self.distance_left = 1000
        self.distance_right = 1000
        self.light = 1000
        self.button = 0
        self.vibration = 0

    def set_distance_front(self, val):
        """
        The value to represent the distance from bot to front object.
        """
        self.distance_front = val

    def set_distance_left(self, val):
        """
        The value to represent the distance from bot to left object.
        """
        self.distance_left = val

    def set_distance_right(self, val):
        """
        The value to represent the distance from bot to right object.
        """
        self.distance_right = val
```

```
def set_distance_up(self, val):
    """
    The value to represent the distance from bot to object above.
    """
    self.distance_up = val

def set_distance_down(self, val):
    """
    The value to represent the distance from bot to object below.
    """
    self.distance_down = val

def set_light(self, val):
    """
    The value to represent the light value.
    """
    self.light = val

def set_vibration(self, val):
    """
    The value to represent the vibration value.
    """
    self.vibration = val

def set_button(self, val):
    """
    The value to represent the button value.
    """
    self.button = val

def __str__(self):
    s = "<UBIInput> "
    s += "[front {}\t".format(self.distance_front)
    s += "up {}\tdown {}\t".format(self.distance_up, self.distance_down)
    s += "left {}\tright {}\t".format(self.distance_left, self.distance_right)
    s += "[light {}]\t".format(self.light)
    s += "[button {}\t]".format(self.button)
    s += "[vibration {}\t]".format(self.vibration)
    return s

def __repr__(self):
    return self.__str__()
```

```
class Bot(object):
    # maximum log the Bot will carry.
    LOG_LENGTH = 1000
    # the threshold for the bot to determine a collision.
    COLLISION = 1000
    # the with of the bot
    WIDTH = 30
    # the height of the bot
    HEIGHT = 10
    # the radius of the bot
    RADIUS = WIDTH / 2.0
    # the bot has 3 modes, auto, light and reset.
    # auto mode: the bot will perform automatic navigation.
    MODE_AUTO = "AUTO"
    # light mode: the bot will be guided with the help of light, then change to auto mode.
    MODE_LIGHT = "LIGHT"
    # reset mode: when time is out, the bot will try to go to initial place.
    MODE_RESET = "RESET"
    # the time limit for the bot being guided by light in seconds.
    TIME_LIGHT_LIMIT = 60
    # the time limit for the bot to perform automatic navigation in seconds.
    TIME_AUTO_LIMIT = 600
    # the light range for the bot to perform automatic navigation.
    LIGHT_RANGE = 50
    # the distance for the bot to slow down if the bot detects a near object.
    WARNING_DISTANCE = 10

    def __init__(self):
        self.log = []
        self.text = ""
        self.mode = Bot.MODE_AUTO
        self.time = timeit.timeit()
        self.low_light = 1000

    def process_input(self, bot_input):
        """
        The only open method for the bot to process the input from grove pi.
        the process could be generally described as:
        1, keep log of the current values
        2, the bot will update the mode
            a, if the button is pressed
            b, if the time for the current mode runs out.
        3, detects if the bot is going to fall
            a, if yes, the bot will turn back, and find a new direction to try.
        4, detects if the bot is colliding with other objects
```

```
        a, if yes, the bot will turn back, and find a new direction to try.
    5, detects if the bot can pass through the path and move forward.
        a, if no, the bot will turn back, and find a new direction to try.
    6, detects if the bot is about to hit objects in front
        a, if yes, the bot will slow down the speed.
    7, apply actions according to current mode and input.
    """
    self.__update_log(bot_input)
    if len(self.log) < 2:
        self.low_light = bot_input.light
        return

    # the next line is used for debugging.
    print "<Bot process_input>", self.log[-1]

    self.__update_mode(bot_input)

    if self.__test_fall(bot_input):
        self.__handle_fall()
        return
    elif self.__test_collision(bot_input):
        self.__handle_collision()
        return
    elif not self.__test_go_through(bot_input):
        self.__handle_not_go_through()
        return

    self.__apply_collision_protection(bot_input)

    if self.mode == Bot.MODE_AUTO:
        self.__apply_auto_mode()
    elif self.mode == Bot.MODE_LIGHT:
        self.__apply_light_mode()
    elif self.mode == Bot.MODE_RESET:
        self.__apply_reset_mode()

def __set_display_lcd(self, text):
    self.text = text
    grovelcd.setText(text)

def __add_display_lcd(self, text):
    self.text += text
    grovelcd.setText(text)

def __apply_collision_protection(self, bot_input):
```

```
    """
    If the bot detects near objects, slow down the speed to protect the bot.
    """
    distance_front = bot_input.distance_front
    if distance_front <= Bot.WARNING_DISTANCE:
        print "please move slowly. the bot is about to hit objects."
        self.__set_display_lcd("please move slowly. the bot is about to hit objects.")

def __update_mode(self, bot_input):
    """
    This method handles mode changes.
    1, test the time, if longer than time limit, switch to other modes accordingly.
    2, if the button is pressed, reset the time, and switch to other modes accordingly.
    """
    # calculate passed time
    passed_time = timeit.timeit() - self.time
    # test if the time is out of limit
    if self.mode == Bot.MODE_LIGHT and passed_time >= Bot.TIME_LIGHT_LIMIT:
        # change mode and reset values.
        self.mode = Bot.MODE_AUTO
        self.low_light = self.log[-1].light
        print "<time out> Bot mode changed to", self.mode
        self.__set_display_lcd("<time out> Bot mode changed to " + self.mode)
        return
    elif self.mode == Bot.MODE_AUTO and passed_time >= Bot.TIME_AUTO_LIMIT:
        # change mode and reset values.
        self.mode = Bot.MODE_RESET
        print "<time out> Bot mode changed to", self.mode
        self.__set_display_lcd("<time out> Bot mode changed to " + self.mode)
        return

    # test if the button is pressed
    if bot_input.button and not self.log[-2].button:
        # update the time
        self.time = timeit.timeit()
        # change mode according to current mode.
        if self.mode == Bot.MODE_AUTO:
            self.mode = Bot.MODE_LIGHT
            self.low_light = self.log[-1].light
        else:
            self.mode = Bot.MODE_AUTO
            self.low_light = self.log[-1].light
        print "Bot mode changed to", self.mode
        self.__set_display_lcd("Bot mode changed to " + self.mode)
```

```
    def __apply_auto_mode(self):
        """
        The auto mode defines the behaviour of the bot
        normally, the bot would move forward.
        meanwhile, the light value would also be monitored.
        """
        print "Bot mode <{}>".format(self.mode)
        low = self.low_light - Bot.LIGHT_RANGE
        current = self.log[-1].light
        high = self.low_light + Bot.LIGHT_RANGE
        if low <= current <= high:
            print "Action: move forward."
            self.__set_display_lcd("Action: move forward.")
        else:
            print "WARNING: light value out of range. Move Back!"
            self.__set_display_lcd("WARNING: light value out of range. Move Back!")
            print "light range [{} - {}], current {}".format(low, high, current)
            self.__add_display_lcd("light range [{} - {}], current {}".format(low, high, current))

    def __apply_light_mode(self):
        """
        The light mode defines the behaviour of the bot.
        the bot would try to find the place with lower light value until the mode turns off.
        """
        print "Bot mode <{}>".format(self.mode), "Action: move to find place with lower light value."
        self.__set_display_lcd("Bot mode <{}>".format(self.mode) + "Action: move to find place with
lower light value.")
        if self.log[-1].light < self.low_light:
            self.low_light = self.log[-1].light

        print "current light value:", self.log[-1].light, "low light value: ", self.low_light
        self.__add_display_lcd("current light value:{} low light value: {}".format(self.log[-1].light,
self.low_light))

    def __apply_reset_mode(self):
        """
        The bot would return to initial place in reset mode.
        this mode is designed to handle exceptions.
        """
        print "Bot mode <{}>".format(self.mode), "Action: move to initial place."
        self.__set_display_lcd("Bot mode <{}>".format(self.mode) + "Action: move to initial place.")


    def __update_log(self, bot_input):
        """
```

```
    the bot would record the input from grove pi.
    in this case, the maximum length is defined to avoid wasting memory resources.
    """
    if len(self.log) >= Bot.LOG_LENGTH:
        self.log.pop(0)
    self.log.append(bot_input)

def __test_go_through(self, bot_input):
    """
    test if the bot can pass through the path
    """
    return bot_input.distance_left > Bot.RADIUS and bot_input.distance_right > Bot.RADIUS

def __test_fall(self, bot_input):
    """
    test if the bot would fall if continue moving forward.
    """
    return bot_input.distance_down > Bot.HEIGHT/2

def __test_collision(self, bot_input):
    """
    test if the bot is colliding with objects.
    """
    return bot_input.vibration > Bot.COLLISION

def __handle_fall(self):
    """
    the behaviour if a potential fall is detected.
    """
    print "The bot is going to fall, move back."
    self.__set_display_lcd("The bot is going to fall, move back.")

def __handle_collision(self):
    """
    the behaviour if collisions are detected.
    """
    print "The bot collides with other objects, turn back and select new direction."
    self.__set_display_lcd("The bot collides with other objects, turn back and select new direction.")

def __handle_not_go_through(self):
    """
    the behaviour if the bot cannot pass the current path.
    """
    print "The bot cannot go through the current path, turn back and select new direction."
```

```
        self.__set_display_lcd("The bot cannot go through the current path, turn back and select new direction.")

# initialize the sensors
sensors = {
    "button": (SensorShim.DIGITAL, 8),
    "light": (SensorShim.ANALOG, 0),
    "vibration": (SensorShim.ANALOG, 1)
}
sensorObj = SensorShim(sensors)

def get_bot_input():
    """
    Get all necessary information from the grove pi sensors.
    """
    bot_input = BotInput()
    bot_input.distance_front = ultrasonicRead(2)
    #bot_input.distance_up = ultrasonicRead(3)
    #bot_input.distance_down = ultrasonicRead(4)
    #bot_input.distance_left = ultrasonicRead(5)
    #bot_input.distance_right = ultrasonicRead(6)
    bot_input.button = sensorObj.getValue("button")
    bot_input.light = sensorObj.getValue("light")
    bot_input.vibration = sensorObj.getValue("vibration")
    return bot_input

def main():
    bot = Bot()
    frequency = 5
    time_gap = 1.0 / frequency
    while True:
        # get all necessary data from grove pi and pass the information to bot for processing
        bot.process_input(get_bot_input())
        time.sleep(time_gap)

if __name__ == "__main__":
    main()
```

Appendix B – Original Proposal

G54UBI Coursework Proposal (2014/15)

Title: Automatic Toy Pickup Robot

Student ID: 4219232
Date: 2014/10/27

Summary

The aim of the project is to provide convenient tools for getting pieces of toys in areas where hands are hard to reach. The prototype will be focusing on the following functionalities:
1. Collision detection: The Robot shall be able to detect a collision between the device and an object. E.g. Wall, and movable object like the small piece of toy. The robot shall behave differently according to the type of the object. E.g. If it is a toy, the robot shall pick-up the toy.
2. Falling prevention: The Robot shall be able to detect a cliff and to avoid falling.
3. Target exploration: The Robot shall be able to find and locate toys in an area.

The prototype is not a complete product that can by itself conduct toy pickup missions. The robot movement, such as moving forward and backward, turning left and right, and picking up objects will be simulated by human at this phase. The main focus will be collision detection, falling prevention, and target exploration.
The prototype will be demonstrated with some kind of visual simulation, e.g. webpage front written in html and java script, desktop application written in C# or python.

Technologies and sensor data

Collision detection: accelerometer will be used to provide data. In this project, 2 types of collisions will be analysed: 1, collision with un-movable object (e.g. wall); 2, collision with movable object (e.g. a piece of toy).
Falling prevention: Ultrasonic ranger will be used. The sensor will be put at the bottom of the robot on this purpose.
Target exploration: Ultrasonic ranger and light sensor will be in use. Distance detection will be used to help identify object in a small area, and the light sensor will guide the exploration in dark space, where hands are very difficult to reach. G54UBI Coursework proposal

Project plan

My current plan for the development is:
Part 1: Week 1 & 2
Acquire some sensor data for algorithm development:
1) Collision detection data collection: Accelerometer sensor
2) Falling prevention data collection: Ultrasonic ranger
3) Target exploration: Ultrasonic ranger and light sensor

Part 2: Week 3
Visual simulation software development.
Part 3: Week 4 & 5
Algorithm development for Collision Detection, Falling Prevention and Target Exploration.
Part 4: Week 6 & 7
Algorithm test and evaluation.

Three main functionalities will be tested separately. All test will be conducted with visual simulation, and all test data will be collected for later analysis.
Part 5: Week 8
Writing up the report.

Skills and competencies

I have some level of programming experience in the past, and am confident with data analysis.