goo.gl/JjNpGZ

Roomba with Python

Import statements

```
1  from bluetooth import *
2
3  from Tkinter import *
4  import tkMessageBox
5  import tkSimpleDialog
6
7  import struct
8
```

Searching for a match

```
 9  addr = "5C:F3:70:76:C9:D4"
10
11  print("Searching for Roomba at %s" % addr)
12
13  # search for the SampleServer service
14  uuid = "00001101-0000-1000-8000-00805F9B34FB"
15  service_matches = find_service( uuid = uuid, address = addr )
16
17  if len(service_matches) == 0:
18      print("Couldn't find the Roomba =(")
19      sys.exit(0)
20
21  first_match = service_matches[0]
```

Creating a client socket

```
22  port = first_match["port"]
23  name = first_match["name"]
24  host = first_match["host"]
25
26  print("Connecting to \"%s\" on %s" % (name, host))
27
28  # Create the client socket
29  sock=BluetoothSocket( RFCOMM )
30  sock.connect((host, port))
31
32  print("Ready to send commands")
33
```

Configuring the GUI

```
33
34  TEXTWIDTH = 40 # window width, in characters
35  TEXTHEIGHT = 16 # window height, in lines
36
37  VELOCITYCHANGE = 400
38  ROTATIONCHANGE = 200
39
40  KEYPRESS = '2'
41  KEYRELEASE = '3'
42
43  helpText = """\
44  Supported Keys:
45  P\tPassive
46  S\tSafe
47  F\tFull
48  C\tClean
49  D\tDock
50  R\tReset
51  Space\tBeep
52  Arrows\tMotion
53
54  If nothing happens after you connect, try pressing 'P' and then 'S' to get into safe mode.
55  """
```

Creating a Class and Key callbacks

```
57  class TetheredDriveApp(Tk):
58      # static variables for keyboard callback -- I know, this is icky
59      callbackKeyUp = False
60      callbackKeyDown = False
61      callbackKeyLeft = False
62      callbackKeyRight = False
63      callbackKeyLastDriveCommand = ''
64
```

Defining an initializing method

```
65      def __init__(self):
66          Tk.__init__(self)
67          self.title("Roomba Control")
68          self.text = Text(self, height = TEXTHEIGHT, width = TEXTWIDTH, wrap = WORD)
69          self.text.pack(side=LEFT, fill=BOTH, expand=True)
70          self.text.insert(END, helpText)
71          self.text.config(state=DISABLED)
72
73          self.bind("<Key>", self.callbackKey)
74          self.bind("<KeyRelease>", self.callbackKey)
75
```

Sending a command via bluetooth

```
76      # sendCommandASCII takes a string of whitespace-separated, ASCII-encoded base 10 values to send
77      def sendCommandASCII(self, command):
78          cmd = ""
79          for v in command.split():
80              cmd += chr(int(v))
81
82          sock.send(cmd)
83
```

Keybinds

```
83
84      # A handler for keyboard events. Feel free to add more!
85      def callbackKey(self, event):
86          k = event.keysym.upper()
87          motionChange = False
88
89          if event.type == KEYPRESS:
90              if k == 'P':   # Passive
91                  self.sendCommandASCII('128')
92              elif k == 'S': # Safe
93                  self.sendCommandASCII('131')
94              elif k == 'F': # Full
95                  self.sendCommandASCII('132')
96              elif k == 'C': # Clean
97                  self.sendCommandASCII('135')
98              elif k == 'D': # Dock
99                  self.sendCommandASCII('143')
100             elif k == 'SPACE': # Beep
101                 self.sendCommandASCII('140 3 1 64 16 141 3')
102             elif k == 'R': # Reset
103                 self.sendCommandASCII('7')
104             elif k == 'UP':
105                 self.callbackKeyUp = True
106                 motionChange = True
107             elif k == 'DOWN':
108                 self.callbackKeyDown = True
109                 motionChange = True
110             elif k == 'LEFT':
111                 self.callbackKeyLeft = True
112                 motionChange = True
113             elif k == 'RIGHT':
114                 self.callbackKeyRight = True
115                 motionChange = True
116             else:
117                 print(str(k) +" not handled")
118     elif event.type == KEYRELEASE:
119         if k == 'UP':
120             self.callbackKeyUp = False
121             motionChange = True
122         elif k == 'DOWN':
123             self.callbackKeyDown = False
124             motionChange = True
125         elif k == 'LEFT':
126             self.callbackKeyLeft = False
127             motionChange = True
128         elif k == 'RIGHT':
129             self.callbackKeyRight = False
130             motionChange = True
131
```

Calculating and sending drive command

```
131
132         if motionChange == True:
133             velocity = 0
134             velocity += VELOCITYCHANGE if self.callbackKeyUp is True else 0
135             velocity -= VELOCITYCHANGE if self.callbackKeyDown is True else 0
136             rotation = 0
137             rotation += ROTATIONCHANGE if self.callbackKeyLeft is True else 0
138             rotation -= ROTATIONCHANGE if self.callbackKeyRight is True else 0
139
140             # compute left and right wheel velocities
141             vr = velocity + (rotation/2)
142             vl = velocity - (rotation/2)
143
144             # create drive command
145             cmd = struct.pack(">Bhh", 145, vr, vl)
146             if cmd != self.callbackKeyLastDriveCommand:
147                 sock.send(cmd)
148                 self.callbackKeyLastDriveCommand = cmd
```

Initializing GUI

```
149
150 if __name__ == "__main__":
151     app = TetheredDriveApp()
152     app.mainloop()
```