**server\_main.py**

#!/usr/bin/env pybricks-micropython

from pybricks.hubs import EV3Brick

from pybricks.ev3devices import (Motor, TouchSensor, ColorSensor, InfraredSensor, UltrasonicSensor, GyroSensor)

from pybricks.parameters import Port, Stop, Direction, Button, Color

from pybricks.tools import wait, StopWatch, DataLog

from pybricks.robotics import DriveBase

from pybricks.media.ev3dev import SoundFile, ImageFile

from pybricks.messaging import BluetoothMailboxServer, TextMailbox

from \_thread import allocate\_lock

from time import sleep

# Create your objects here.

ev3 = EV3Brick()

left\_motor = Motor(Port.B)

right\_motor = Motor(Port.C)

color = ColorSensor(Port.S4)

# Write your program here.

ev3.speaker.beep()

# Declare constants here.

WHEEL = 17.6

TRACK = 11.7

SPEED = 102

# Note the timeout parameter doesn't seem to work

# This wait checks for a message and then just returns

# Compared to wait() which will sit there and wait

# for a message

def wait\_with\_timeout(mbox,name):

"""Waits until ``mbox`` receives a value."""

lock = allocate\_lock()

lock.acquire()

with mbox.\_connection.\_lock:

mbox.\_connection.\_updates[name] = lock

try:

#timeout parameter doesn't work as

#expected so I'm just setting

#the wait flag to be zero

return lock.acquire(0,1)

finally:

with mbox.\_connection.\_lock:

del mbox.\_connection.\_updates[name]

def setupConnection():

server = BluetoothMailboxServer()

mbox = TextMailbox('greeting', server)

# The server must be started before the client!

print('waiting for connection...')

server.wait\_for\_connection()

print('connected!')

return mbox

def playSound(note):

if note == "q":

return

elif note == "a":

ev3.speaker.beep(880,100)

elif note == "c":

ev3.speaker.beep(523,100)

elif note == "g":

ev3.speaker.beep(783,100)

def server():

mbox = setupConnection()

lastColor = ""

while 1:

# wait\_with\_timeout(mbox,mbox.name) # Non-blocking

mbox.wait() # Blocking

msg = mbox.read()

if msg != None:

print("Message received is: "+msg)

# Assumes there are only two tokens

cmd,arg = mbox.read().split(":")

if cmd == "play":

playSound(arg)

elif cmd == "forward":

moveForward(arg)

elif cmd == "turn":

turn90(arg)

elif cmd == "wasd":

wasd(arg)

elif cmd == "sensor":

nextColor = color.color()

while nextColor == lastColor:

nextColor = color.color()

lastColor = nextColor

mbox.send(color.color())

elif cmd == "motors":

left\_speed, right\_speed = arg.split(",")

motors(int(left\_speed), int(right\_speed))

mbox.send("Motor speeds set")

else:

print("Received invalid cmd "+cmd)

# Clears msg and the mailbox

msg = None

mbox.\_connection.\_mailboxes={}

def moveForward(dist):

"""

Moves the robot forward a specified distance for task 1

"""

dist = float(dist)

run\_time = dist / (WHEEL/360) / SPEED

left\_motor.run(SPEED)

right\_motor.run(SPEED)

sleep(run\_time)

left\_motor.run(0)

right\_motor.run(0)

def turn90(dir):

"""

Turns the robot 90 degrees in a specified direction for task 1

"""

if dir == "left":

left\_motor.run(-37.584)

right\_motor.run(37.584)

sleep(5)

left\_motor.run(0)

right\_motor.run(0)

elif dir == "right":

left\_motor.run(37.584)

right\_motor.run(-37.584)

sleep(5)

left\_motor.run(0)

right\_motor.run(0)

def wasd(button):

"""

Moves the robot based on the most recent WASD button pressed for task 4

"""

if button == "w":

left\_motor.run(SPEED)

right\_motor.run(SPEED)

elif button == "a":

left\_motor.run(-SPEED)

right\_motor.run(SPEED)

elif button == "s":

left\_motor.run(-SPEED)

right\_motor.run(-SPEED)

elif button == "d":

left\_motor.run(SPEED)

right\_motor.run(-SPEED)

elif button == "space":

left\_motor.run(0)

right\_motor.run(0)

def motors(left\_speed, right\_speed):

"""

Sets the motors to given speeds for task 5

"""

left\_motor.run(left\_speed)

right\_motor.run(right\_speed)

def main():

server()

if \_\_name\_\_ == '\_\_main\_\_':

main()

**client\_main.py**

#!/usr/bin/env python3

from pybricks.messaging import BluetoothMailboxClient, TextMailbox

from stateMachine import StateMachine

import keyboard

# Client code to be run on a pc

SERVER = '24:71:89:4a:86:c4'

mboxClient = BluetoothMailboxClient()

mbox = TextMailbox('greeting', mboxClient)

def is\_number(n):

try:

float(n)

except ValueError:

return False

return True

def print\_values\_color():

"""

This function is used to print out test values from the color sensor

"""

mbox.send("sensor:dummy")

mbox.wait()

color = mbox.read()

print("Color: " + color)

def task\_five\_turn\_left():

"""

This function is used as the turn left action for the FSM

"""

mbox.send("motors:0,100")

mbox.wait()

def task\_five\_turn\_right():

"""

This function is used as the turn right action for the FSM

"""

mbox.send("motors:100,0")

mbox.wait()

def task\_five\_on\_tape\_condition():

"""

This function is used to check the on tape condition for the FSM

"""

mbox.send("sensor:dummy")

mbox.wait()

color = mbox.read()

return color == "Color.WHITE"

def task\_five\_off\_tape\_condition():

"""

This function is used to check the off tape condition for the FSM

"""

mbox.send("sensor:dummy")

mbox.wait()

color = mbox.read()

return color != "Color.WHITE"

def task\_five():

"""

This is the main execution loop for the task five state machine

"""

state\_machine = StateMachine()

state\_machine.add\_state("Turn\_Left", task\_five\_turn\_left, default=True) \

.add\_state("Turn\_Right", task\_five\_turn\_right) \

.add\_transition("Turn\_Left", "Turn\_Right", task\_five\_on\_tape\_condition) \

.add\_transition("Turn\_Right", "Turn\_Left", task\_five\_off\_tape\_condition)

while True:

state\_machine.run()

def client():

print('establishing connection...')

mboxClient.connect(SERVER)

print('connected!')

while 1:

cmd = input("Enter a cmd (play or buttons): ")

if cmd == "play":

freq = input("Enter a frequency")

msg = cmd+":"+freq

print("Sending message "+msg)

mbox.send(msg)

elif cmd == "buttons":

msg = cmd+":dummy"

print("Sending message "+msg)

mbox.send(msg)

mbox.wait() # wait for a response

b = mbox.read()

print("lasted button pressed was "+str(b))

elif cmd == "forward":

dist = input("Enter a distance (in cm)")

while not is\_number(dist):

dist = input("Not a valid distance. Try again")

msg = cmd+":"+dist

print("Sending message "+msg)

mbox.send(msg)

elif cmd == "turn":

direction = input("Enter a direction (left/right)")

while direction != "left" and direction != "right":

direction = input("Not a valid direction. Try again")

msg = cmd+":"+direction

print("Sending message "+msg)

mbox.send(msg)

elif cmd == "wasd":

print("Press one of the WASD keys to move the robot. Press space when you are finished to stop it.")

stop = False

last\_msg = ""

while not stop:

if keyboard.is\_pressed("w"):

msg = cmd + ":w"

elif keyboard.is\_pressed("a"):

msg = cmd + ":a"

elif keyboard.is\_pressed("s"):

msg = cmd + ":s"

elif keyboard.is\_pressed("d"):

msg = cmd + ":d"

elif keyboard.is\_pressed("space"):

msg = cmd + ":space"

stop = True

else:

msg = last\_msg

if msg != last\_msg:

mbox.send(msg)

last\_msg = msg

elif cmd == "line":

task\_five()

elif cmd == "color":

print\_values\_color()

else:

print("Unrecognized cmd: "+cmd)

def main():

client()

if \_\_name\_\_ == '\_\_main\_\_':

main()

**stateMachine.py**

class StateTransition:

"""

This class wraps a State Transition entity to store the required data

"""

def \_\_init\_\_(self, from\_state, to\_state, condition):

self.from\_state = from\_state

self.to\_state = to\_state

self.condition = condition

class StateMachine:

"""

This class encompasses logic to create a FSM using provided action and condition

functions

"""

def \_\_init\_\_(self):

self.states = {}

self.transitions = []

self.default\_state = None

self.current\_state = None

def add\_state(self, name, action\_func, default=False):

"""

This function adds a state to the state machine, with an associated function to

execute when this state is active. The function should not loop, and should be

passed in without parenthesis.

Example:

def my\_func():

pass

add\_state("MyFunc", my\_func)

"""

self.states[name] = action\_func

if default:

self.default\_state = name

return self

def add\_transition(self, from\_state, to\_state, condition\_func):

"""

This function adds a transition between two states, with an associated function

that is called to check whether the transition should occur. The condition should

not loop, and should be passed in without parenthesis.

Example:

def my\_condition():

pass

add\_transition("From", "To", my\_condition)

"""

transition = StateTransition(from\_state, to\_state, condition\_func)

self.transitions.append(transition)

return self

def run(self):

"""

This function is used as the primary execution loop for the defined FSM

"""

# Confirm that both the default and current states have values

if self.default\_state is None:

self.default\_state = list(self.states.keys())[0]

if self.current\_state is None:

self.current\_state = self.default\_state

self.states[self.current\_state]()

# Search through the transitions to see if any conditions are met

for transition in self.transitions:

if transition.from\_state == self.current\_state:

# If the stored condition is true, change states

if transition.condition():

self.current\_state = transition.to\_state

# Call the function stored for the state

self.states[self.current\_state]()

**Task\_3\_Server.py**

#!/usr/bin/env pybricks-micropython

from pybricks.hubs import EV3Brick

from pybricks.ev3devices import (Motor, TouchSensor, ColorSensor,

InfraredSensor, UltrasonicSensor, GyroSensor)

from pybricks.parameters import Port, Stop, Direction, Button, Color

from pybricks.tools import wait, StopWatch, DataLog

from pybricks.robotics import DriveBase

from pybricks.media.ev3dev import SoundFile, ImageFile

from pybricks.messaging import BluetoothMailboxServer, TextMailbox

from \_thread import allocate\_lock

# Create your objects here.

ev3 = EV3Brick()

motor\_one = Motor(Port.A)

motor\_two = Motor(Port.D)

touch = TouchSensor(Port.S1)

sonic = UltrasonicSensor(Port.S2)

gyro = GyroSensor(Port.S3)

rgb = ColorSensor(Port.S4)

# Write your program here.

ev3.speaker.beep()

# SERVER SIDE

#Note the timeout parameter doesn't seem to work

#This wait checks for a message and then just returns

#Compared to wait() which will sit there and wait

#for a message

def wait\_with\_timeout(mbox,name):

"""Waits until ``mbox`` receives a value."""

lock = allocate\_lock()

lock.acquire()

with mbox.\_connection.\_lock:

mbox.\_connection.\_updates[name] = lock

try:

#timeout parameter doesn't work as

#expected so I'm just setting

#the wait flag to be zero

return lock.acquire(0,1)

finally:

with mbox.\_connection.\_lock:

del mbox.\_connection.\_updates[name]

def ping():

server = BluetoothMailboxServer()

mbox = TextMailbox('greeting', server)

# The server must be started before the client!

print('waiting for connection...')

server.wait\_for\_connection()

print('connected!')

# In this program, the server waits for the client to send the first message

# and then sends a reply.

mbox.wait()

print(mbox.read())

mbox.send('hello to you!')

def setupConnection():

server = BluetoothMailboxServer()

mbox = TextMailbox('greeting', server)

# The server must be started before the client!

print('waiting for connection...')

server.wait\_for\_connection()

print('connected!')

return mbox

def receiveMsg():

mbox = setupConnection()

while 1:

mbox = setupConnection()

mbox.wait()

msg = mbox.read()

print("Message received is: "+msg)

if msg == "q":

break

def playNotes():

mbox = setupConnection()

while 1:

wait\_with\_timeout(mbox,mbox.name)

msg = mbox.read()

if msg != None:

print("Message received is: "+msg)

if msg == "q":

break

elif msg == "a":

ev3.speaker.beep(880,100)

elif msg == "c":

ev3.speaker.beep(523,100)

elif msg == "g":

ev3.speaker.beep(783,100)

mbox.\_connection.\_mailboxes={}

msg = None

def server():

mbox = setupConnection()

pressed = ""

while 1:

wait\_with\_timeout(mbox,mbox.name) #non-blocking

#wait() #blocking

msg = mbox.read()

#if pressed != []:

# print("pressed is "+str(pressed))

if msg != None:

print("Message received is: "+msg)

#assumes there are only two tokens

cmd,arg = mbox.read().split(":")

if cmd == "play":

#print("arg is "+arg)

if arg == "q":

break

elif arg == "a":

ev3.speaker.beep(880,100)

elif arg == "c":

ev3.speaker.beep(523,100)

elif arg == "g":

ev3.speaker.beep(783,100)

elif cmd == "buttons":

#print("pressed is "+str(pressed))

mbox.send(pressed)

else:

print("Received invalid cmd "+cmd)

#clears msg and the mailbox

msg = None

mbox.\_connection.\_mailboxes={}

buttons = ev3.buttons.pressed()

#gets the buttons pressed if any

if buttons != []:

pressed = "" #clears pressed

for i in range(len(buttons)):

pressed+=str(buttons[i])

if i!=len(buttons)-1:

pressed+=":"

def printButtons():

pressed = ""

while True:

buttons = ev3.buttons.pressed()

print(buttons)

#gets the buttons pressed if any

if buttons != []:

pressed = "" #clears pressed

for i in range(len(buttons)):

pressed+=str(buttons[i])

if i!=len(buttons)-1:

pressed+=":"

print("Pressed is "+pressed)

def sendSensorData():

mbox = setupConnection()

while 1:

# Phase 1

mbox.wait()

cmd,arg = mbox.read().split(":")

if cmd == "sensors":

port = arg.upper()

print('Sensor data requested for Port '+ port +'.')

# Phase 2

if port == 'S1':

sensor = 'Touch Sensor'

elif port == 'S2':

sensor = 'Sonic Sensor'

elif port == 'S3':

sensor = 'Gyro Sensor'

elif port == 'S4':

sensor = 'Color Sensor'

else:

sensor = 'Invalid Port Number'

# Phase 3

if sensor == 'Touch Sensor':

data = str(touch.pressed())

elif sensor == 'Sonic Sensor':

data = str(sonic.distance()) + ' mm'

elif sensor == 'Gyro Sensor':

data = str(gyro.speed()) + ' deg/s'

elif sensor == 'Color Sensor':

data = str(rgb.color())

else:

data = 'No Data Available'

mbox.send(sensor + ':\t' + data)

def sendDataStream():

mbox = setupConnection()

while 1:

print("Received continuous sensor data stream request.")

while True:

data\_stream = ("Touch Sensor: "+str(touch.pressed())+"\t\tSonic Sensor: "+str(sonic.distance())+

" mm"+"\t\tGyro Sensor: "+str(gyro.speed())+" deg/s"+"\t\tColor Sensor: "+str(rgb.color()))

mbox.send(data\_stream)

def main():

#sendDataStream()

sendSensorData()

#receiveMsg()

#playNotes()

#server()

#printButtons()

if \_\_name\_\_ == '\_\_main\_\_':

main()

**Task\_3\_Client.py**

#!/usr/bin/env python3

from pybricks.messaging import BluetoothMailboxClient, TextMailbox

# client code to be run on a pc

SERVER = '24:71:89:4f:02:f7'

def ping():

client = BluetoothMailboxClient()

mbox = TextMailbox('greeting', client)

print('establishing connection...')

client.connect(SERVER)

print('connected!')

# In this program, the client sends the first message and then waits for the

# server to reply.

mbox.send('hello!')

mbox.wait()

print(mbox.read())

def sendMsg():

client = BluetoothMailboxClient()

mbox = TextMailbox('greeting', client)

print('establishing connection...')

client.connect(SERVER)

print('connected!')

while 1:

cmd = input("Enter a letter: ")

print("Sent command was: " + cmd)

mbox.send(cmd)

def client():

client = BluetoothMailboxClient()

mbox = TextMailbox('greeting', client)

print('establishing connection...')

client.connect(SERVER)

print('connected!')

while 1:

cmd = input("Enter a cmd (play or buttons): ")

if cmd == "play":

freq = input("Enter a frequency")

msg = cmd + ":" + freq

print("Sending message " + msg)

mbox.send(cmd + ":" + freq)

elif cmd == "buttons":

msg = cmd + ":" + "dummy"

print("Sending message " + msg)

mbox.send(msg)

mbox.wait() # wait for a response

b = mbox.read()

print("lasted button pressed was " + str(b))

else:

print("Unrecognized cmd: " + cmd)

def requestSensorData():

client = BluetoothMailboxClient()

mbox = TextMailbox('greeting', client)

print('establishing connection...')

client.connect(SERVER)

print('connected!')

while 1:

cmd = input("Enter the port number (S1-S4): ")

print("Sensor for port number " + cmd + " has been requested.")

mbox.send("sensors:" + cmd)

mbox.wait()

b = mbox.read()

print(str(b))

def requestDataStream():

client = BluetoothMailboxClient()

mbox = TextMailbox('greeting', client)

print('establishing connection...')

client.connect(SERVER)

print('connected!')

while 1:

print("Requested Data Stream.")

while True:

mbox.wait()

b = mbox.read()

print(str(b))

def main():

#requestDataStream()

requestSensorData()

#ping()

#sendMsg()

#client()

if \_\_name\_\_ == '\_\_main\_\_':

main()