**Explanation of States**

1. **State 0.** “Dead”
   1. This is the state that the system is initialized into.
   2. A read LED shines.
   3. Motors are off.
   4. When the GUI is started, everything is greyed out, and the only option is to ‘initialize’ to begin.
2. State 1. “Primed/Armed/Stopped”
   1. This state is the ‘home base’ for the system.
   2. After ‘initialization,’ this state is entered.
   3. The motors are off.
   4. A yellow light shines.
   5. You can enter this state from all other states by clicking the ‘Stop’ button.
   6. When an object is detected, the system automatically enters this state.
   7. The system stays in this state until a button is clicked.
3. State 2. “Turn Left”
   1. To enter this state, click the ‘Turn Left’ button in the GUI.
   2. This state is *timed* and operates the motors.
   3. The pins connected to the h-bridge are set to cause ‘left’ angular rotation of the robot.
   4. After a period of time, the state is exited and returns to the previous state.
4. State 3. “Turn Right”
   1. To enter this state, click the ‘Turn Right’ button in the GUI.
   2. This state is *timed* and operates the motors.
   3. The pins connected to the h-bridge are set to cause ‘right’ angular rotation of the robot.
   4. After a period of time, the state is exited and returns to the previous state.
5. State 4. “Move Forward”
   1. To enter this state, click the ‘Forward’ button in the GUI.
   2. The h-bridge pins are set to rotate both motors in the same direction to move the robot forward.
   3. The ultrasonic sensor frequently records measurements; if measurements are less than a threshold, the system moves to the ‘Stop’ state to shut off the motors. Object avoidance is implemented in this state only.
   4. To manually exit this state, click any other button in the GUI.
6. State 5. “Move Backward” / “Reverse”
   1. To enter this state, click the ‘Reverse’ button in the GUI.
   2. The h-bridge pins are set to rotate both motors in the same direction to move the robot backward.
   3. To exit this state, click any other button in the GUI.

Graphical user interface, table

Description automatically generated with medium confidence

**Commands**

The ‘Manual Control’ section of the GUI is responsible for sending commands that move between states in the Arduino. Clicking a button switches to the state number shown on each button.

**4**

**5**

**3**

**2**

**1**

**0**

The radio buttons for speed only update a PWM pin value and do not cause a change of state.

Graphical user interface, table

Description automatically generated with medium confidence**Code In Action**

The GUI contains three main sections:

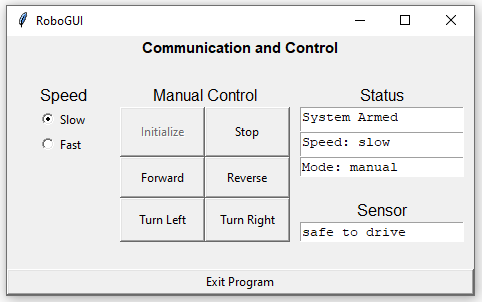
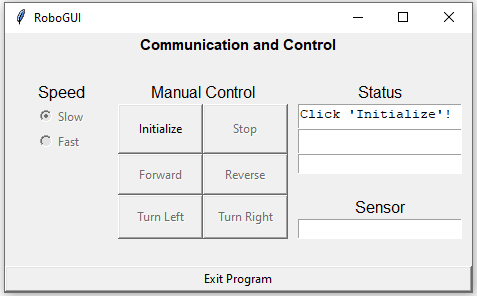
* Speed setting (left)
* Manual Control (center)
* Statuses (right)

The speed setting can be switched between ‘slow’ and ‘fast,’ which alters the PWM value sent to the motors.

The manual control buttons send commands via serial connection to the Arduino to switch between states.

The ‘Status’ and ‘Sensor’ section are used purely as means to display information to the user and are automatically updated based on system configurations and events that occur.

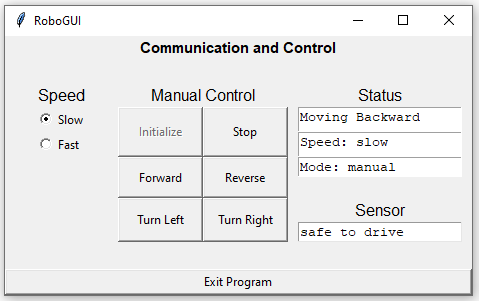
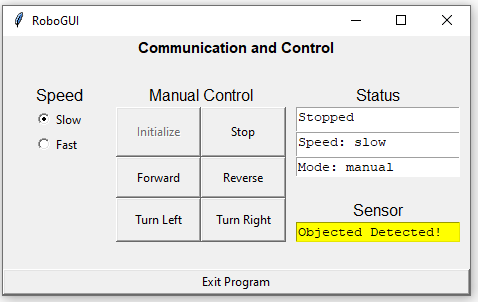
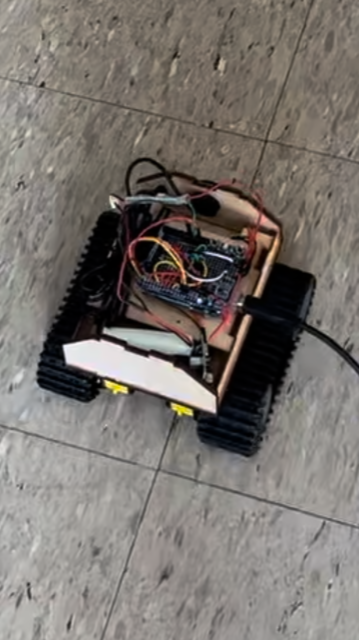
When the GUI is opened, all functionality is blocked until ‘Initialize’ is clicked. Then, regular operation begins.



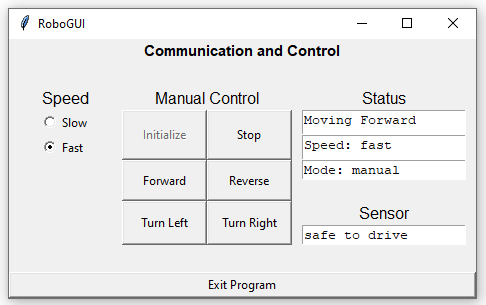
Graphical user interface, table

Description automatically generated with medium confidenceClicking different buttons in the ‘Manual Control’ section sends commands via serial connection to the Arduino, which switches the system case and updates the digital output pins that connect to the h-bridge for motor control. The GUI updates accordingly.

When the robot encroaches upon an object, the ultrasonic sensor detects it and stops the robot. The GUI updates.



The user is in control and can click any button in the GUI to free the robot from its ‘stopped’ state. Note that if the user tries to go forward, the robot will again see the object and quickly stop. Hence, backing up (reverse) is advised.



The speed of the robot can be adjusted. Clicking ‘slow’ or ‘fast’ sends a command to the Arduino via serial comms and sets the PWM value for the motor (the Pin that goes to the PWM input on the h-bridge).

**Appendix A: Python GUI Code**

# Mechatronics - Jason Noel & Phil Parisi - 6Dec2022

# Final Project - Wheeled Robot

############### NOTES

# Commands for Task 1

# i --> initialize

# l --> left

# r --> right

# f --> forward

# b --> back/reverse

# s --> stop

# a --> autonomous

# z --> null

# Other Parameters to Pass to Arudino

# 1 --> 'slow' speed mode (send as CHAR not int)

# 2 --> 'fast' speed mode (send as CHAR not int)

############### PACKAGES

import tkinter as tk

import serial

import time

############### SETUP

# Arduino Setup

arduino = serial.Serial(port='COM3', baudrate=9600, timeout=0.05)

valBytes = bytes('d','utf-8') #Convert to bytes

arduino.write(valBytes)

# Default Values

commandChar = 'z' #Command to switch states DEFAULT "OFF" = 'a'

progMode = 'manual' # 2 modes: 'manual' and 'auto'

currentSpeed = '1' # 2 levels: 'slow' = '1' and 'fast' = '2'

objectDetected = 0 # use to indicate when objects are detected

############### FUNCTIONS

# must define functions before we call them

def initialize():

global commandChar

#Clear and Post Status Message in GUI

outMsg = "System Armed"

writeTextReadOnly(outMsg, motionBox)

writeTextReadOnly("Speed: slow", speedBox)

writeTextReadOnly("Mode: manual", progModeBox)

writeTextReadOnly("safe to drive", sensorBox)

# Disable Start Button

initializeTaskButton.configure(state = 'disable')

# Enable Other Buttons

stopTaskButton.configure(state = 'normal')

forwardButton.configure(state = 'normal')

backwardButton.configure(state = 'normal')

turnRightButton.configure(state = 'normal')

turnLeftButton.configure(state = 'normal')

slowSpeedRadio.configure(state = 'normal')

fastSpeedRadio.configure(state = 'normal')

manualModeRadio.configure(state = 'normal')

autoModeRadio.configure(state = 'normal')

# Update Command

commandChar = 'i'

pyToSerial(commandChar)

# Infinite While Loop

runMainLoop()

def stop():

global commandChar

global objectDetected

# Clear and Post Status Message in GUI

outMsg = "Stopped"

writeTextReadOnly(outMsg, motionBox)

# Update Command

commandChar = 's'

pyToSerial(commandChar)

if progMode == "auto":

manualModeRadio.select()

updateProgMode()

# Clear Sensor if necessary

if objectDetected == 1:

objectDetection()

def moveForward():

global commandChar

global objectDetected

# Clear and Post Status Message in GUI

outMsg = "Moving Forward"

writeTextReadOnly(outMsg, motionBox)

# Update Command

commandChar = 'f'

pyToSerial(commandChar)

# Clear Sensor if necessary

if objectDetected == 1:

objectDetection()

def moveBackward():

global commandChar

global objectDetected

# Clear and Post Status Message in GUI

outMsg = "Moving Backward"

writeTextReadOnly(outMsg, motionBox)

# Update Command

commandChar = 'b'

pyToSerial(commandChar)

# Clear Sensor if necessary

if objectDetected == 1:

objectDetection()

def turnRight():

global commandChar

global objectDetected

# Clear and Post Status Message in GUI

outMsg = "Turning Right"

writeTextReadOnly(outMsg, motionBox)

# Update Command

commandChar = 'r'

pyToSerial(commandChar)

# Clear Sensor if necessary

if objectDetected == 1:

objectDetection()

def turnLeft():

global commandChar

global objectDetected

# Clear and Post Status Message in GUI

outMsg = "Turning Left"

writeTextReadOnly(outMsg, motionBox)

# Update Command

commandChar = 'l'

pyToSerial(commandChar)

# Clear Sensor if necessary

if objectDetected == 1:

objectDetection()

def exitApp():

# Stop the Motor

commandChar = 's'

pyToSerial(commandChar)

# Quit Application

#main.quit(),#main.after(1000,lambda:main.destroy())

main.destroy()

quit()

def writeTextReadOnly(outMsg, widget):

# keep the text boxes read only, no user writing

widget.configure(state = 'normal')

widget.delete('1.0', "end")

widget.insert("end", outMsg)

widget.configure(state = 'disabled')

def updateSpeed():

# when the radio button for speed is changed, this command is run

# the radioSpeed variable is automatically changed when the radio buttons hit

# slow, radioSpeed = "slow"

# fast, radioSpeed = "fast"

global currentSpeed

radioVal = radioSpeed.get() # "slow" or "fast"

if radioVal == "slow":

currentSpeed = '1'

elif radioVal == "fast":

currentSpeed = '2'

else:

currentSpeed = '1' # default to slowest speed

outMsg = "Speed: " + str(radioVal)

writeTextReadOnly(outMsg, speedBox)

pyToSerial(currentSpeed) # send '1' or '2' to arduino

def setDefaultsGUI():

# Set Radio Buttons

fastSpeedRadio.deselect()

slowSpeedRadio.select()

manualModeRadio.select()

autoModeRadio.deselect()

# Set Statuses

writeTextReadOnly(" ", speedBox)

writeTextReadOnly("Click 'Initialize'!", motionBox)

writeTextReadOnly(" ", progModeBox)

writeTextReadOnly(" ", sensorBox)

# Gray Out Buttons except Start

stopTaskButton.configure(state = 'disable')

forwardButton.configure(state = 'disable')

backwardButton.configure(state = 'disable')

turnRightButton.configure(state = 'disable')

turnLeftButton.configure(state = 'disable')

# Gray out all Radio Buttons

slowSpeedRadio.configure(state = 'disable')

fastSpeedRadio.configure(state = 'disable')

manualModeRadio.configure(state = 'disable')

autoModeRadio.configure(state = 'disable')

def updateProgMode():

global progMode

global commandChar

motionMsg = " "

if radioProgMode.get() == "manual":

if progMode == "auto":

writeTextReadOnly("Stopped", motionBox)

commandChar = 's' # is this right?

progMode = 'manual'

commandToGUI = 1

elif radioProgMode.get() == "auto":

if progMode == "manual":

writeTextReadOnly("Autonomous", motionBox)

progMode = 'auto'

commandToGUI = 0

commandChar = 'a'

else:

progMode = 'manual'

commandToGUI = 1

commandChar = 's'

writeTextReadOnly("Stopped")

# Update GUI

setAllGUIFeatures(commandToGUI)

modeMsg = "Mode: " + str(progMode)

writeTextReadOnly(modeMsg, progModeBox)

def setAllGUIFeatures(commandToGUI):

if commandToGUI:

val = 'normal'

else:

val = 'disable'

# Set GUI Features to 'normal' or 'disable'

forwardButton.configure(state = val)

backwardButton.configure(state = val)

turnRightButton.configure(state = val)

turnLeftButton.configure(state = val)

slowSpeedRadio.configure(state = val)

fastSpeedRadio.configure(state = val)

def pyToSerial(val):

val\_bytes = bytes(val,'utf-8') #Convert to bytes

arduino.write(val\_bytes) #Send to arduino

print('The encoded value sent to Arduino is: ' + str(val\_bytes))

time.sleep(0.05) #Delay to allow for transmission

def objectDetection():

global objectDetected

if objectDetected == 0:

sensorBox.config(bg='yellow')

writeTextReadOnly("Objected Detected!", sensorBox)

writeTextReadOnly("Stopped", motionBox)

objectDetected = 1

elif objectDetected == 1:

sensorBox.config(bg='white')

writeTextReadOnly("safe to drive", sensorBox)

#writeTextReadOnly("Stopped", motionBox)

objectDetected = 0

def runMainLoop():

counter = 0

while True:

# Update GUI Elements (replacement for main.mainloop() which blocks)

main.update\_idletasks()

main.update()

if (arduino.inWaiting()):

read\_value = arduino.readline()#Read received value from Arduino

print(read\_value)

objectDetection()

############### GUI SETUP

# Create Window

main = tk.Tk()

main.title('RoboGUI')

# Add Title

mainTitle = tk.Label(main, text = "Communication and Control", font = ("Arial",11,"bold"))

blank1Label = tk.Label(main)

# Labels

selectSpeedLabel = tk.Label(main, text = "Speed", font = ("Arial", 12))

manualControlLabel = tk.Label(main, text = "Manual Control", font = ("Arial", 12))

statusLabel = tk.Label(main, text = "Status", font = ("Arial", 12))

selectModeLabel = tk.Label(main, text = "Mode", font = ("Arial", 12))

sensorLabel = tk.Label(main, text = "Sensor", font = ("Arial", 12))

blank2Label = tk.Label(main)

# Text Boxes

motionBox = tk.Text(main, width = 20, height = 1)

speedBox = tk.Text(main, width = 20, height = 1)

progModeBox = tk.Text(main, width = 20, height = 1)

sensorBox = tk.Text(main, width = 20, height = 1)

# Buttons

initializeTaskButton = tk.Button(main, text = "Initialize", width = 8, command = initialize)

forwardButton = tk.Button(main, text = 'Forward', width = 8, height = 2, command = moveForward)

backwardButton = tk.Button(main, text = 'Reverse', width = 8, height = 2, command = moveBackward)

turnRightButton = tk.Button(main, text = "Turn Right", width = 8, height = 2, command = turnRight)

turnLeftButton = tk.Button(main, text = "Turn Left", width = 8, height = 2, command = turnLeft)

stopTaskButton = tk.Button(main, text = "Stop", width = 8,command = stop)

exitAppButton = tk.Button(main, text = "Exit Program", command = exitApp)

# RadioButtons

radioSpeed = tk.StringVar()

slowSpeedRadio = tk.Radiobutton(main, text="Slow", variable = radioSpeed, value = "slow", command = updateSpeed)

fastSpeedRadio = tk.Radiobutton(main, text="Fast", variable = radioSpeed, value = "fast", command = updateSpeed)

radioProgMode = tk.StringVar()

manualModeRadio = tk.Radiobutton(main, text="Manual", variable = radioProgMode, value = "manual", command = updateProgMode)

autoModeRadio = tk.Radiobutton(main, text="Auto", variable = radioProgMode, value = "auto", command = updateProgMode)

##################### Layout

mainTitle.grid(row = 0, column = 0, columnspan = 5)

blank1Label.grid(row = 1, column = 0, columnspan = 5, pady = 1)

# Speed

selectSpeedLabel.grid(row = 2, column = 0, columnspan = 2, padx = 30)

slowSpeedRadio.grid(row = 3, column = 1, sticky = 'w')

fastSpeedRadio.grid(row = 4, column = 1, sticky = 'w')

# Mode

#selectModeLabel.grid(row = 6, column = 0, columnspan = 2, padx = 30)

#manualModeRadio.grid(row = 7, column = 1, sticky = 'w')

#autoModeRadio.grid(row = 8, column = 1, sticky = 'w')

# Manual Control

manualControlLabel.grid(row = 2, column = 2, columnspan = 2, padx = 30)

initializeTaskButton.grid(row = 3, column = 2, rowspan = 2, sticky = 'nesw')

stopTaskButton.grid(row = 3, column = 3, rowspan = 2, sticky = 'nesw')

forwardButton.grid(row = 5, column = 2, rowspan = 2, sticky = 'nesw')

backwardButton.grid(row = 5, column = 3, rowspan = 2, sticky = 'nesw')

turnLeftButton.grid(row = 7, column = 2, rowspan = 2, sticky = 'nesw')

turnRightButton.grid(row = 7, column = 3, rowspan = 2, sticky = 'nesw')

# Status

statusLabel.grid(row = 2, column = 4, padx = 30)

motionBox.grid(row = 3, column = 4, sticky = 'ns', padx = 10)

speedBox.grid(row = 4, column = 4, sticky = 'ns')

progModeBox.grid(row = 5, column = 4, sticky = 'n', padx = 10)

# Sensor Reading

sensorLabel.grid(row = 7, column = 4)

sensorBox.grid(row = 8, column = 4, sticky = 'ns', padx = 10)

# Exit Button

blank2Label.grid(row = 10, column = 0, columnspan = 5, pady = 3)

exitAppButton.grid(row = 11, column = 0, columnspan = 5, sticky = 'ew')

######################## BEGIN ACTUAL PROGRAM

setDefaultsGUI() # run GUI defaults

main.mainloop() #start/open GUI

**Appendix B: Arduino Code**

// Jason Noel & Phil Parisi - Mechatronics Final Project

// 'z' --> null state (does nothing, holds current state)

// Manual States

byte state0Entry = false; // 0 --> 'dead' (system off)

byte state1Entry = false; // 1 --> primed

byte state2Entry = false; // 2 --> turning left

byte state3Entry = false; // 3 --> turning right

byte state4Entry = false; // 4 --> drive forward

byte state5Entry = false; // 5 --> drive backward

// Immutable 'Variables'

#define bridgeForwardRight 4 // (Hbridge connection 4A)

#define bridgeBackwardRight 2 // (Hbridge connection 3A)

#define bridgeForwardLeft 7 // (Hbridge connection 1A)

#define bridgeBackwardLeft 8 // (Hbridge connection 2A)

#define bridgePWM 9 //PWM pin (Hbridge connection 1,2EN and 3,4EN)

#define redLED 6

#define yellowLED 5

#define slowMotorPWM 170 // these probably need adjustment based on what it takes to move the motors

#define fastMotorPWM 255

#define serialCheckTimeInterval 50 // time in ms

#define turnTimeInterval 1500 // time in ms NEEDS TO BE TUNED

#define echoPin 11 // attach pin to Echo pin of JSN-SR04T sensor (this is the PWM)

#define trigPin 12 // attach to to Trig pin of JSN-SR04T

#define minimumSensorDistance 25 // centimeters

//#define maxSensorHits 15 // hits below minimumSensorDistance to trigger robot stop

#define sensorPingTime 10 // microseconds

#define maxSensorHitCounts 2

// Mutable 'Global Variables'

byte state; // State and NextState of 1st task transition diagram

byte nextState;

byte returnToState;

byte sensorTimerOn;

char command; // User command

unsigned long serialTime;

unsigned long turnTime;

unsigned long sensorTimer;

long objectDistance;

long duration;

long distance;

int sensorHitCounts;

// Function Declarations (Prototypes)

void controlTask(void); // Task state transition diagram function

void controlHbridgePWM();

unsigned long getMilliTimeNow(void); // Returns time in ms units

unsigned long getMicroTimeNow(void); // Returns time in us units

void motorSetSpeed(int val);

void turnLeft();

void turnRight();

void moveForward();

void moveBackward();

void objectAvoidance(long distance);

void sensorPing();

// Setup Function

void setup() {

// Build In LED

pinMode(LED\_BUILTIN, OUTPUT);

pinMode(redLED, OUTPUT);

digitalWrite(redLED, LOW);

pinMode(yellowLED, OUTPUT);

digitalWrite(yellowLED, LOW);

// H-Bridge Pin Setup

pinMode(bridgeForwardRight, OUTPUT);

pinMode(bridgeBackwardRight, OUTPUT); // HIGH and LOW values

pinMode(bridgeForwardLeft, OUTPUT);

pinMode(bridgeBackwardRight, OUTPUT);

pinMode(bridgePWM, OUTPUT); // PWM for speed control

// H-Bridge Pin Initial Values [all low]

digitalWrite(bridgeForwardRight, LOW);

digitalWrite(bridgeBackwardRight, LOW);

digitalWrite(bridgeForwardLeft, LOW);

digitalWrite(bridgeBackwardLeft, LOW);

motorSetSpeed(slowMotorPWM); // sets bridgePWM pin to slowMotorPWM speed as default

// Sensor Setup

digitalWrite(trigPin, LOW);

digitalWrite(echoPin, LOW);

sensorTimerOn = false;

sensorHitCounts = 0;

// Control Flow Setup

char command = 'z'; // User command

nextState = 0; // start in null 'dead' state

Serial.begin(9600);

}

// Main Loop

void loop() {

// Starting Timer for Serial Connection

serialTime = getMilliTimeNow(); // for serial check time interval

// Infinite While Loop

while (2 > 1) // Start infinite loop

{

if ((getMilliTimeNow() - serialTime) >= serialCheckTimeInterval) // This if statement is used to not check the serial port all the time

{

serialTime = getMilliTimeNow();

// Code to read command (a single char) from serial port here

if (Serial.available() > 0) //Wait for a char to show up

{

command = Serial.read(); // Read one byte from serial buffer

}

}

// Tasks for Every Loop

controlTask(); // Call ControlTask 1 (looks for commands of the letter format like 's' 'f' 'b')

controlHbridgePWM(); // allow commands sent from GUI to update the PWM signal for hbridge (commands of the form '1' for slow and '2' for fast)

// Reset command

command = 'z'; // reset this so commands are executed only once

}

}

// Definition of ControlTask Function

void controlTask(void) {

state = nextState;

// Jump between STATES

switch (state) // Go to current state

{

case 0: // System Off 'Dead'

if (state0Entry == false) // if not in the state, get in it!

{

state0Entry = true;

stopMotors();

digitalWrite(redLED, HIGH);

}

if (command == 'i') // 'initialize', move to State 1!

{

nextState = 1; // change current state

state0Entry = false; // we're leaving the state, set it to false

digitalWrite(redLED, LOW);

}

break;

case 1: // Primed & Stopped

if (state1Entry == false) // if not in the state, get in it!

{

state1Entry = true; // we are in state 1

stopMotors();

digitalWrite(yellowLED, HIGH);

}

if (command == 'f') // 'forward', move to state 4 (move forward)

{

nextState = 4;

state1Entry = false;

digitalWrite(yellowLED, LOW);

}

if (command == 'b') // 'reverse', move to state 5 (move backward)

{

nextState = 5;

state1Entry = false;

digitalWrite(yellowLED, LOW);

}

if (command == 'd') // 'die', go back to initialize

{

nextState = 0;

state1Entry = false;

digitalWrite(yellowLED, LOW);

}

if (command == 'l') // 'left', move to state 2 (turn left)

{

returnToState = 1;

nextState = 2;

state1Entry = false;

digitalWrite(yellowLED, LOW);

}

if (command == 'r') // 'right', move to state 3 (turn right)

{

returnToState = 1;

nextState = 3;

state1Entry = false;

digitalWrite(yellowLED, LOW);

}

break;

case 2: // turn left 90deg

if (state2Entry == false) // if not in the state, get into it

{

turnTime = getMilliTimeNow();

state2Entry = true;

turnLeft();

}

if (command == 's') // Stop Task!

{

nextState = 1; // get into state 1

state2Entry = false; // get out of state 2

turnTime = 0;

}

if ((getMilliTimeNow() - turnTime) >= turnTimeInterval) // we've been learning for long enough!

{

nextState = returnToState; // go back to previous state

state2Entry = false; // get out of current state

turnTime = 0;

}

break;

case 3: // turn right 90deg

if (state3Entry == false) // if not in the state, get into it

{

turnTime = getMilliTimeNow();

state3Entry = true;

turnRight();

}

if (command == 's') // stop

{

nextState = 1; //

state3Entry = false; // get out of state 2

turnTime = 0;

}

if ((getMilliTimeNow() - turnTime) >= turnTimeInterval) // we've been turning for long enough!

{

nextState = returnToState; // go back to previous state

state3Entry = false; // get out of current state

turnTime = 0;

}

break;

case 4: // drive forward

if (state4Entry == false) // if not in the state, get into it

{

state4Entry = true;

driveForward();

digitalWrite(LED\_BUILTIN, HIGH);

}

if (state4Entry == true) // do this every loop when we're in the state

{

sensorPing(); // ping with ultrasonic sensor

/\*

if ( (getMilliTimeNow() >= 10000) && (getMilliTimeNow() <= 10100) )

{

Serial.print('p');

}

\*/

}

if (command == 's') // stop

{

nextState = 1;

state4Entry = false; // get out of state 4

digitalWrite(LED\_BUILTIN, LOW);

}

if (command == 'b') // go backwards

{

nextState = 5;

state4Entry = false;

digitalWrite(LED\_BUILTIN, LOW);

}

if (command == 'd') // 'die'

{

nextState = 0;

state4Entry = false;

digitalWrite(LED\_BUILTIN, LOW);

}

if (command == 'l') // 'left', move to state 2 (turn left)

{

returnToState = 4;

nextState = 2;

state4Entry = false;

digitalWrite(LED\_BUILTIN, LOW);

}

if (command == 'r') // 'right', move to state 3 (turn right)

{

returnToState = 4;

nextState = 3;

state4Entry = false;

digitalWrite(LED\_BUILTIN, LOW);

}

if (nextState != 4)

{

state4Entry = false;

digitalWrite(LED\_BUILTIN, LOW);

}

break;

case 5: // drive backward

if (state5Entry == false) // if not in the state, get into it

{

state5Entry = true;

driveBackward();

}

if (command == 's') // stop

{

nextState = 1;

state5Entry = false; // get out of state 5

}

if (command == 'f') // go forwards

{

nextState = 4;

state5Entry = false;

}

if (command == 'd') // 'die'

{

nextState = 0;

state5Entry = false;

}

if (command == 'l') // 'left', move to state 2 (turn left)

{

returnToState = 5;

nextState = 2;

state5Entry = false;

}

if (command == 'r') // 'right', move to state 3 (turn right)

{

returnToState = 5;

nextState = 3;

state5Entry = false;

}

break;

}

}

unsigned long getMilliTimeNow(void) // Returns time in ms units

{

return (millis());

}

unsigned long getMicroTimeNow(void) // Returns time in us units

{

return (micros());

}

void driveForward()

{

// to move forward...

digitalWrite(bridgeForwardRight, HIGH); // right side foward

digitalWrite(bridgeBackwardRight, LOW);

digitalWrite(bridgeForwardLeft, HIGH); // left side foward

digitalWrite(bridgeBackwardLeft, LOW);

}

void driveBackward()

{

// to move backward...

digitalWrite(bridgeForwardRight, LOW); // right side backward

digitalWrite(bridgeBackwardRight, HIGH);

digitalWrite(bridgeForwardLeft, LOW); // left side backward

digitalWrite(bridgeBackwardLeft, HIGH);

}

void turnLeft()

{

// to turn left...

digitalWrite(bridgeForwardRight, LOW); // right side backward

digitalWrite(bridgeBackwardRight, HIGH);

digitalWrite(bridgeForwardLeft, HIGH); // left side forward

digitalWrite(bridgeBackwardLeft, LOW);

}

void turnRight()

{

// to turn right...

digitalWrite(bridgeForwardRight, HIGH); // right side foward

digitalWrite(bridgeBackwardRight, LOW);

digitalWrite(bridgeForwardLeft, LOW); // left side backward

digitalWrite(bridgeBackwardLeft, HIGH);

}

void stopMotors()

{

// to stop motors...

digitalWrite(bridgeForwardRight, LOW); // right side backward

digitalWrite(bridgeBackwardRight, LOW);

digitalWrite(bridgeForwardLeft, LOW); // left side backward

digitalWrite(bridgeBackwardLeft, LOW);

}

void controlHbridgePWM()

{

if (command == '1') // slow

{

motorSetSpeed(slowMotorPWM);

}

if (command == '2') // fast

{

motorSetSpeed(fastMotorPWM);

}

}

void motorSetSpeed(int val)

{

analogWrite(bridgePWM, val);

}

void sensorPing() // function from Sensor Machine

{

if (sensorTimerOn == false) // timer is off, start next cycle

{

//distance = 1000; // generically high value to not trigger stopping

sensorTimerOn = true;

sensorTimer = getMicroTimeNow();

digitalWrite(trigPin, HIGH);

}

if (sensorTimerOn == true) // timer is on, check if enough time has elapsed

{

if ((getMicroTimeNow() - sensorTimer) >= sensorPingTime) // if enough time elapsed, get dist

{

digitalWrite(trigPin, LOW);

duration = pulseIn(echoPin, HIGH);

distance = duration \* 0.0343 / 2;

sensorTimerOn = false;

objectAvoidance(distance);

distance = 1000; // reset distance

}

}

}

void objectAvoidance(long distance)

{

if (distance <= minimumSensorDistance)

{

sensorHitCounts = sensorHitCounts + 1;

}

if (sensorHitCounts >= maxSensorHitCounts) // allows us to deal with noisy/random hits that aren't true

{

Serial.print('p'); // tell the GUI that we are too close to something!

nextState = 1;

sensorHitCounts = 0;

}

}