Interaction

Introduction

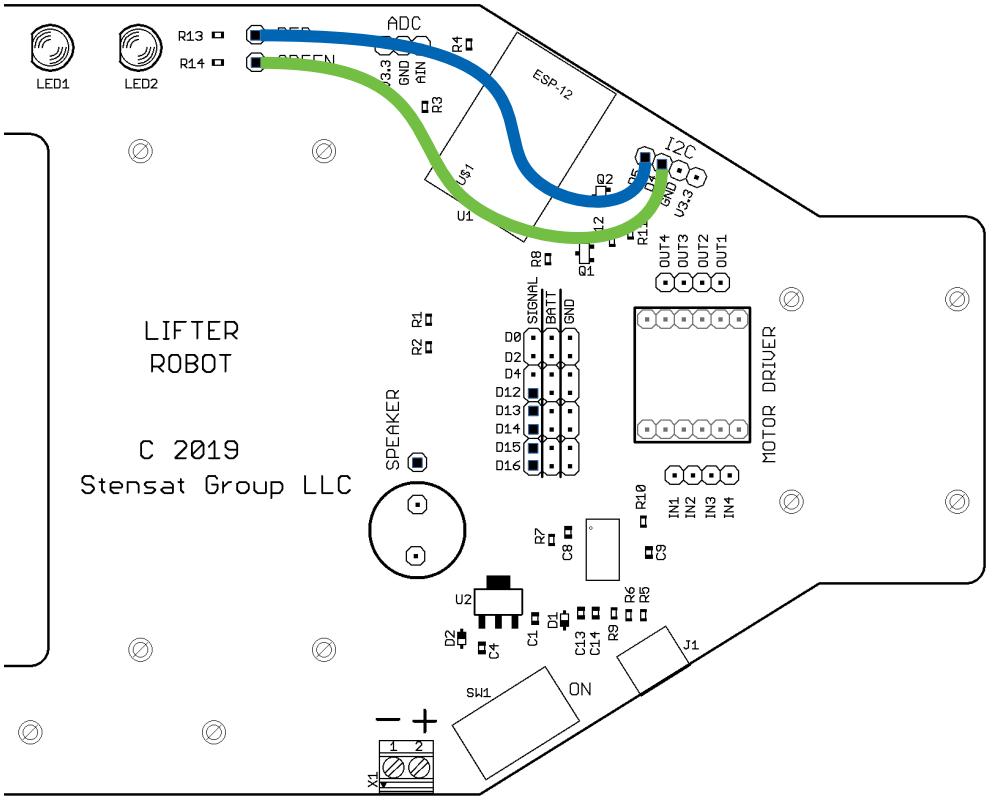
- This lesson will teach how to get python code to interact with the robot.
- You will be able to use the keyboard and create a simple graphical user interface.

Turning LEDs on and Off

- The LEDs will be controlled with the keyboard. The robot code will listen for commands through the USB interface from a python program.
- The python program will detect key presses on the keyboard and send commands to turn on and off each LED.
- A new library is needed for python called pyserial.
- Open a power shell in Windows.
- Enter the command **pip3 install pyserial** and press the enter key.
- The library will be installed. Once completed, exit the power shell.

LED Control

- Connect the digital pins 4 and 5 to the two LEDs as shown.
- The red LED will be controlled by pin 5. The green LED will be controlled by pin 4.
- The LEDs will be controlled by pressing keys on a keyboard. The A key will turn on the red LED. The S key will turn off the red LED. The Q key will turn on the green LED and the W key will turn off the green LED.



Key Control

- Starting with the robot code, open the Arduino program and start with a new program.
- Enter the code to the right.

```
void setup() {
  Serial.begin(115200);
  pinMode(4,OUTPUT);
  pinMode(5,OUTPUT);
}

void loop() {
  if(Serial.available() > 0) {
    int a = Serial.read();
    switch(a) {
      case 'a': digitalWrite(5,HIGH);
                break;
      case 's': digitalWrite(5,LOW);
                break;
      case 'q': digitalWrite(4,HIGH);
                break;
      case 'w': digitalWrite(4,LOW);
                break;
    }
  }
}
```

Key Control

- The setup() function should look familiar. The serial interface which interacts through the USB port is initialized.
- The two digital pins connected to the LEDs are set to output.

```
void setup() {
  Serial.begin(115200);
  pinMode(4,OUTPUT);
  pinMode(5,OUTPUT);
}

void loop() {
  if(Serial.available() > 0) {
    int a = Serial.read();
    switch(a) {
      case 'a': digitalWrite(5,HIGH);
                break;
      case 's': digitalWrite(5,LOW);
                break;
      case 'q': digitalWrite(4,HIGH);
                break;
      case 'w': digitalWrite(4,LOW);
                break;
    }
  }
}
```

Key Control

- The first function in the the **loop()** function checks is any data has been sent.
- **Serial.available()** checks if any command has been sent from the python program. More than one can occur so the comparison in the **if()** statement is for any number greater than zero. If a command was sent, the code between the bold brackets gets executed otherwise, nothing is executed and the program checks again.

```
void setup() {
  Serial.begin(115200);
  pinMode(4,OUTPUT);
  pinMode(5,OUTPUT);
}

void loop() {
  if(Serial.available() > 0) {
    int a = Serial.read();
    switch(a) {
      case 'a': digitalWrite(5,HIGH);
                break;
      case 's': digitalWrite(5,LOW);
                break;
      case 'q': digitalWrite(4,HIGH);
                break;
      case 'w': digitalWrite(4,LOW);
                break;
    }
  }
}
```

Key Control

- When a command is received, the next step is to get the command.
- Variable **a** is created and then assigned the command value from the serial interface.

```
void setup() {
  Serial.begin(115200);
  pinMode(4,OUTPUT);
  pinMode(5,OUTPUT);
}

void loop() {
  if(Serial.available() > 0) {
    int a = Serial.read();
    switch(a) {
      case 'a': digitalWrite(5,HIGH);
                break;
      case 's': digitalWrite(5,LOW);
                break;
      case 'q': digitalWrite(4,HIGH);
                break;
      case 'w': digitalWrite(4,LOW);
                break;
    }
  }
}
```

Key Control

- The **switch()** statement is a simplification of **if()** statements. Instead of checking for each possible value that variable a could be, the **switch()** statement lists the possible values and the instructions to execute when the value matches the variable value.
- The case statement has the value to check to the right. For this program, the code is looking for key presses from the keyboard. What is received is the letter of the key. This is with the Caps Lock off. A single quote indicates it is a letter.

```
void setup() {
  Serial.begin(115200);
  pinMode(4,OUTPUT);
  pinMode(5,OUTPUT);
}

void loop() {
  if(Serial.available() > 0) {
    int a = Serial.read();
    switch(a) {
      case 'a': digitalWrite(5,HIGH);
                break;
      case 's': digitalWrite(5,LOW);
                break;
      case 'q': digitalWrite(4,HIGH);
                break;
      case 'w': digitalWrite(4,LOW);
                break;
    }
  }
}
```

Key Control

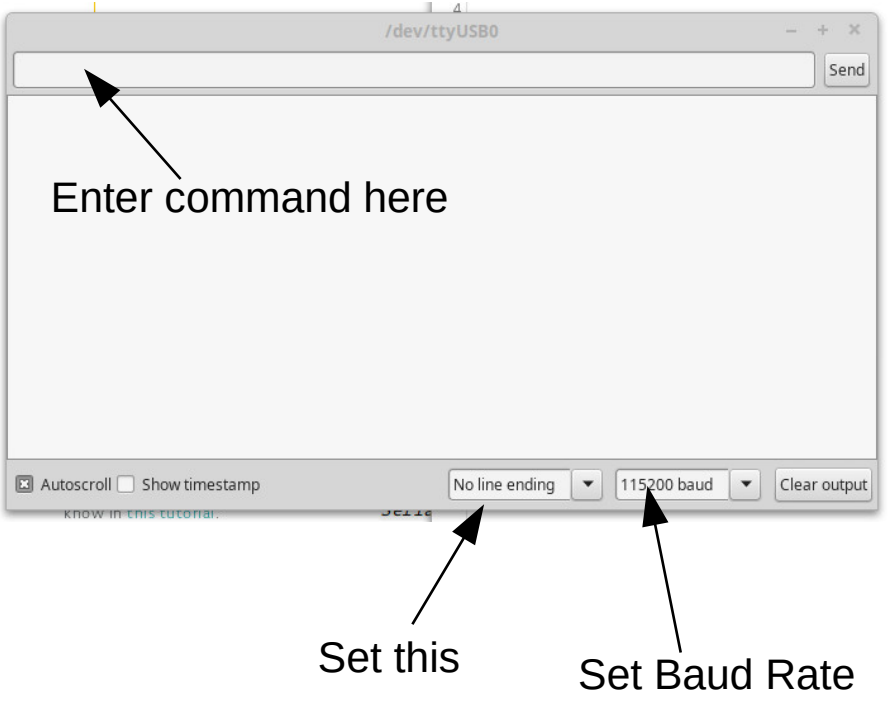
- After the colon, the code is executed if the match is made. You can have many lines of code.
- The **break** statement tells the code to skip the rest and exit the **switch()** statement. If **break** was not included, all the code of the other case statements would be executed below where the case was executed.
- If **'q'** was detected, the program would execute at the **case 'q' :** statement and continue on through the **case 'w'** statement without a **break** to stop the execution.

```
void setup() {
  Serial.begin(115200);
  pinMode(4,OUTPUT);
  pinMode(5,OUTPUT);
}

void loop() {
  if(Serial.available() > 0) {
    int a = Serial.read();
    switch(a) {
      case 'a': digitalWrite(5,HIGH);
                break;
      case 's': digitalWrite(5,LOW);
                break;
      case 'q': digitalWrite(4,HIGH);
                break;
      case 'w': digitalWrite(4,LOW);
                break;
    }
  }
}
```

Key Control

- Upload the code in arduino to the robot.
- Open the serial monitor. Make sure it is configured as shown to the right.
- Set the bottom left tab to **No line ending.**
- Make sure the baud rate is set to 115200.
- In the top part, enter the letter a and press the enter key. The red LED should have turned on. Try the other letters.



Python Key Control

- Open the Python IDLE program. In the file menu, select New to open a program editor.
- Enter the program to the right.

```
import serial
import pygame

s = serial.Serial("COM4",115200)
pygame.init()
scn = pygame.display.set_mode((800,400))

while 1:
  for event in pygame.event.get():
    if event.type == pygame.QUIT:
      pygame.quit()
      quit()
    if event.type == pygame.KEYDOWN:
      if even.key == pygame.K_a:
        s.write(b'a')
      if event.type == pygame.K_s:
        s.write(b's')
      if event.type == pygame.K_q:
        s.write(b'q')
      if event.type == pygame.K_w:
        s.write(b'w')
```

Python Key Control

- The first library imported is the serial library. This allows the python program to talk to the robot through the USB port.
- Specify the same COM port as the Arduino IDE uses to upload code to the robot.

```
import serial
import pygame

s = serial.Serial("COM4",115200)
pygame.init()
scn = pygame.display.set_mode((800,400))

while 1:
  for event in pygame.event.get():
    if event.type == pygame.QUIT:
      pygame.quit()
      quit()
    if event.type == pygame.KEYDOWN:
      if even.key == pygame.K_a:
        s.write(b'a')
      if event.type == pygame.K_s:
        s.write(b's')
      if event.type == pygame.K_q:
        s.write(b'q')
      if event.type == pygame.K_w:
        s.write(b'w')
```

Python Key Control

- The pygame library is initialized the same as before and a window is created with the size of 800 pixels by 400 pixels.

```
import serial
import pygame

s = serial.Serial("COM4",115200)
pygame.init()
scn = pygame.display.set_mode((800,400))

while 1:
  for event in pygame.event.get():
    if event.type == pygame.QUIT:
      pygame.quit()
      quit()
    if event.type == pygame.KEYDOWN:
      if even.key == pygame.K_a:
        s.write(b'a')
      if event.type == pygame.K_s:
        s.write(b's')
      if event.type == pygame.K_q:
        s.write(b'q')
      if event.type == pygame.K_w:
        s.write(b'w')
```

Python Key Control

- In the while loop, the checks for events. The first type of event checked is if the close button on the window has been clicked. If true, the pygame library quits and the program quits.

```
import serial
import pygame

s = serial.Serial("COM4",115200)
pygame.init()
scn = pygame.display.set_mode((800,400))

while 1:
  for event in pygame.event.get():
    if event.type == pygame.QUIT:
      pygame.quit()
      quit()
    if event.type == pygame.KEYDOWN:
      if even.key == pygame.K_a:
        s.write(b'a')
      if event.type == pygame.K_s:
        s.write(b's')
      if event.type == pygame.K_q:
        s.write(b'q')
      if event.type == pygame.K_w:
        s.write(b'w')
```

Python Key Control

- The second event is detecting the press of a key on the keyboard. If true, then the program checks which key was pressed.
- **pygame.K_a** represents the **A** key pressed with the Caps Lock off and not pressing the shift key. If you want to detect the shifted **A** key, use **pygame.K_A**.

```
import serial
import pygame

s = serial.Serial("COM4",115200)
pygame.init()
scn = pygame.display.set_mode((800,400))

while 1:
  for event in pygame.event.get():
    if event.type == pygame.QUIT:
      pygame.quit()
      quit()
    if event.type == pygame.KEYDOWN:
      if even.key == pygame.K_a:
        s.write(b'a')
      if event.type == pygame.K_s:
        s.write(b's')
      if event.type == pygame.K_q:
        s.write(b'q')
      if event.type == pygame.K_w:
        s.write(b'w')
```

Python Key Control

- When the correct key is pressed, the program will send out the command which is a single letter to the robot.
- Notice the syntax is **b'a'**. The **b** in front tells python to use a byte size character. Normally python uses unicode to identify letters which is 16-bits. The serial library does not support unicode.

```
import serial
import pygame

s = serial.Serial("COM4",115200)
pygame.init()
scn = pygame.display.set_mode((800,400))

while 1:
  for event in pygame.event.get():
    if event.type == pygame.QUIT:
      pygame.quit()
      quit()
    if event.type == pygame.KEYDOWN:
      if even.key == pygame.K_a:
        s.write(b'a')
      if event.type == pygame.K_s:
        s.write(b's')
      if event.type == pygame.K_q:
        s.write(b'q')
      if event.type == pygame.K_w:
        s.write(b'w')
```

WiFi

- The wifi integrated in the processor provides a wireless way to communicate with the processor. The processor will be configured as an access point. This means it becomes a local network where your laptop connects. It is also possible to have a tablet connect to the processor. In this lesson, you will learn how to control digital pins. This will require you to write code on the laptop.

WiFi

- **WiFi** is a local area wireless computer network. It is also known as wireless local area network. **WiFI** is a standard for allowing computers to interact with each other using radio signals. A **wireless access point** is a device that connects a wireless network to a wired network. It can also provide a local isolated network not connected to the internet or other wired network. Access points usually have a network router and can provide network addresses or IP addresses to any device that connects.

WiFi

- **SSID** – is a unique identifier for the WiFi network. It can have up to 32 characters and is case sensitive. This allows multiple WiFi access points in the same area without interfering with each other.
- **IP Address** – is the internet protocol address assigned to each device on the network. There are two standards, IP-4 and IP-6. IP-4 is used here. The address consists of four sets of numbers separated by a decimal point. Each number has a range of 0 to 255. Example 192.168.1.10.
- **DHCP** – is Dynamic Host Configuration Protocol. This protocol allows a WiFi router to assign an IP address to any device that connects to the WiFi network. This is done automatically.
- **TCP** – is Transmission Control Protocol. This is one of the main network protocols used by any device on any WiFi network or the internet. The protocol enables two devices to establish a connection to each other and exchange data. The protocol guarantees delivery of data and that the data is delivered in the same order sent. The sender sends a data packet, when the receivers gets the packet, it sends an acknowledgment If the receiver doesn't receive the packet, the sender will send again after a time out period.
- **UDP** – is user datagram Protocol. This protocol is a stateless protocol. No connection needs to be made and packets received are not acknowledged. The sender just sends a packet to an IP address and port. There is no guaranteere the receiver actually received any packets. Data packets can be sent much more quickly because there is no handshaking.

WiFi

- There are two parts to the WIFI operation. Configuration which sets up the module to operate properly. Data operation where the module receives data and can send data. The WIFI module will be configured to operate as an access point. This allows another computer to connect to the module and communicate with the module. More than one WIFI access point can be in the same area and operate independent of each other as long as their SSID are different. In this lesson, the WIFI module will be configured as an access point and allow TCP connections.

WiFi TCP

- This section will show how to wirelessly control the LEDs using TCP packets.
- This requires the python program to connect to the robot.
- The robot will operate as an access point so the computer running python must connect its WiFi to the robot access point.

WiFi TCP

- The first half of the Arduino program is shown to the right. You can modify the existing LED control program.
- The first line imports the WiFi library.
- The second line declares a client object. This is used to talk with the python program.
- The third line creates a server object that listens for clients on port 80.

```
#include <ESP8266WiFi.h>

WiFiClient client;
WiFiServer server(80);

void setup()
{
  Serial.begin(115200);
  pinMode(4,OUTPUT);
  pinMode(5,OUTPUT);
  WiFi.mode(WIFI_AP);
  WiFi.softAP("nameofboard");
  server.begin();
}
```

WiFi TCP

- In the **setup()** function, after setting up the serial interface and the digital pins, the WiFi is set to access point mode.
- Then the access point is turned on with the **WiFi.softAP()** function. The parameter in quotes is the SSID. Create your own.
- The last line activates the server so connections can be established.

```
#include <ESP8266WiFi.h>

WiFiClient client;
WiFiServer server(80);

void setup()
{
  Serial.begin(115200);
  pinMode(4,OUTPUT);
  pinMode(5,OUTPUT);
  WiFi.mode(WIFI_AP);
  WiFi.softAP("nameofboard");
  server.begin();
}
```

WiFi TCP

- In the **loop()** function, the first thing done is to see if a client has connected to the robot. That is done with the first two lines.
- First, client is assigned a value from **server.available()**. If no clients connected, then the value is zero. If there is a connection, then the value is not zero.
- The **if()** statement is true when the value is not zero. If a client did connect, the code within the **if()** statement is executed.

```
void loop() {
  client = server.available();
  if(client) {
    while(client.connected()) {
      while(!client.available()) {
        if(!client.connected()) break;
        delay(1);
      }
      int a = client.read();
      switch(a) {
        case 'a': digitalWrite(5,HIGH);
                  break;
        case 's': digitalWrite(5,LOW);
                  break;
        case 'q': digitalWrite(4,HIGH);
                  break;
        case 'w': digitalWrite(4,LOW);
                  break;
      }
    }
  }
}
```

WiFi TCP

- The next part is the **while()** loop. This is required since the client will stay connected for a while. The **while()** loop checks if the client is still connected. As long as the client is still connected, the code within the brackets gets executed.

```
void loop() {
  client = server.available();
  if(client) {
    while(client.connected()) {
      while(!client.available()) {
        if(!client.connected()) break;
        delay(1);
      }
      int a = client.read();
      switch(a) {
        case 'a': digitalWrite(5,HIGH);
                  break;
        case 's': digitalWrite(5,LOW);
                  break;
        case 'q': digitalWrite(4,HIGH);
                  break;
        case 'w': digitalWrite(4,LOW);
                  break;
      }
    }
  }
}
```

WiFi TCP

- The next **while()** loop checks if any packets are received from the client. As long as no packets are received, the **while()** loop executes. Inside that **while()** loop, it checks to make sure the connection is still valid. If not, the program breaks out of the **while()** loop. Then the program has a 1 ms delay. This is needed so the system can service the WiFi. If it was not included, the program would crash. It's how the software works.

```
void loop() {
  client = server.available();
  if(client) {
    while(client.connected()) {
      while(!client.available()) {
        if(!client.connected()) break;
        delay(1);
      }
      int a = client.read();
      switch(a) {
        case 'a': digitalWrite(5,HIGH);
                  break;
        case 's': digitalWrite(5,LOW);
                  break;
        case 'q': digitalWrite(4,HIGH);
                  break;
        case 'w': digitalWrite(4,LOW);
                  break;
      }
    }
  }
}
```

WiFi TCP

- Once a packet has been received from the client, the code reads a byte from the client packet and then uses it in the switch() statement just like the previous program using the serial interface.
- Upload the program to the robot.

```
void loop() {
  client = server.available();
  if(client) {
    while(client.connected()) {
      while(!client.available()) {
        if(!client.connected()) break;
        delay(1);
      }
      int a = client.read();
      switch(a) {
        case 'a': digitalWrite(5,HIGH);
                  break;
        case 's': digitalWrite(5,LOW);
                  break;
        case 'q': digitalWrite(4,HIGH);
                  break;
        case 'w': digitalWrite(4,LOW);
                  break;
      }
    }
  }
}
```

WiFi TCP

- The python program changes a little bit. The serial library is replace with the socket library which allows access to network functions.
- The network connection is configured in the second highlighted line. It is configured for TCP connection.
- The third highlighted line makes the actual connection to the robot. With the robot set up as the access point, it has the default IP address of 192.168.4.1.

```
import socket
import pygame

s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
s.connect(('192.168.4.1',80))
pygame.init()
scn = pygame.display.set_mode((800,400))

while 1:
  for event in pygame.event.get():
    if event.type == pygame.QUIT:
      s.close()
      pygame.quit()
      quit()
    if event.type == pygame.KEYDOWN:
      if even.key == pygame.K_a:
        s.sendall(b'a')
      if event.type == pygame.K_s:
        s.sendall(b's')
      if event.type == pygame.K_q:
        s.sendall(b'q')
      if event.type == pygame.K_w:
        s.sendall(b'w')
```

WiFi TCP

- There is one small change to the quit event. Since a network connection is being established, it also has to be closed when the program quits. If not, then the robot code will not know that the client has disconnected.

```
import socket
import pygame

s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
s.connect(('192.168.4.1',80))
pygame.init()
scn = pygame.display.set_mode((800,400))

while 1:
  for event in pygame.event.get():
    if event.type == pygame.QUIT:
      s.close()
      pygame.quit()
      quit()
    if event.type == pygame.KEYDOWN:
      if even.key == pygame.K_a:
        s.sendall(b'a')
      if event.type == pygame.K_s:
        s.sendall(b's')
      if event.type == pygame.K_q:
        s.sendall(b'q')
      if event.type == pygame.K_w:
        s.sendall(b'w')
```

WiFi TCP

- The other change to the python code required is the changing the **s.write()** to **s.sendall().**
- The socket library does not have a **write()** function. It has a **sendall()** function.
- Connect the computer WiFi to the robot access point. Windows will sit for a while looking like it is trying to connect but what it really is doing is trying to contact a server on the internet. Once the network icon changes, it is ready.

```
import socket
import pygame

s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
s.connect((‘192.168.4.1’,80))
pygame.init()
scn = pygame.display.set_mode((800,400))

while 1:
  for event in pygame.event.get():
    if event.type == pygame.QUIT:
      s.close()
      pygame.quit()
      quit()
    if event.type == pygame.KEYDOWN:
      if even.key == pygame.K_a:
        s.sendall(b‘a’)
      if event.type == pygame.K_s:
        s.sendall(b‘s’)
      if event.type == pygame.K_q:
        s.sendall(b‘q’)
      if event.type == pygame.K_w:
        s.sendall(b‘w’)
```

WiFi TCP

- After connecting to the robot, run the the python program.
- Press the A and S keys. The red LED should turn on and off.
- Try the other keys.
- Congratulations, you have a working WiFi connection.
- As an exercise, add another set of keys to turn the speaker on and off. Pick a tone.
- As a further exercise, use several keys to play notes on the speaker.

WiFi UDP

- Next, UDP packets will be used in this example. UDP packets do not require a connection. the program sending packets, sends packets not knowing if they were received. The benefit is that packets can be sent very quickly. The potential issue is that some packets may get lost due to radio interference or something else.

WiFi UDP

- The Arduino program needs to be modified to support UDP.
- The highlighted code shows all the changes.
- A new include file is added to support UDP packets.
- A variable array is created to accept UDP packets.
- And **server.begin()** is replace with **udp.begin()** which also specifies the port that packets will be sent to.

```
#include <ESP8266WiFi.h>
#include <WiFiUdp.h>

WiFiUDP udp;

char cmd[256];

void setup()
{
  Serial.begin(115200);
  pinMode(4,OUTPUT);
  pinMode(5,OUTPUT);
  WiFi.mode(WIFI_AP);
  WiFi.softAP("nameofboard");
  udp.begin(80);
}
```

WiFi UDP

- In the beginning of the **loop()** function, the code checks for a UDP packet. If one has arrived, the function returns the number of bytes received.
- The **if()** statement executes if bytes have been received.

```
void loop() {
  int ps = udp.parsePacket();
  if(ps) {
    udp.read(cmd,256);
    int a = cmd[0];
    switch(a) {
      case 'a': digitalWrite(5,HIGH);
                break;
      case 's': digitalWrite(5,LOW);
                break;
      case 'q': digitalWrite(4,HIGH);
                break;
      case 'w': digitalWrite(4,LOW);
                break;
    }
  }
}
```

WiFi UDP

- Once a packet has been received and detected, the packet is read into the array. The commands are a single byte so only the first byte will have the command.
- The second highlighted line assigns the variable **a** the first byte of the packet which contains the command.
- The rest of the code is the same as before.

```
void loop() {
  int ps = udp.parsePacket();
  if(ps) {
    udp.read(cmd,256);
    int a = cmd[0];
    switch(a) {
      case 'a': digitalWrite(5,HIGH);
                break;
      case 's': digitalWrite(5,LOW);
                break;
      case 'q': digitalWrite(4,HIGH);
                break;
      case 'w': digitalWrite(4,LOW);
                break;
    }
  }
}
```

WiFi UDP

- The Python code changes a little bit from the TCP code.
- The socket declaration is changed. The second argument is changed from SOCK_STREAM to SOCK_DGRAM. This indicates UDP packets will be used.
- The second highlighted line sets the variable address to the address and port number of the robot.

```
import socket
import pygame

s = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
address = ('192.168.4.1',80)
pygame.init()
scn = pygame.display.set_mode((800,400))

while 1:
  for event in pygame.event.get():
    if event.type == pygame.QUIT:
      s.close()
      pygame.quit()
      quit()
    if event.type == pygame.KEYDOWN:
      if even.key == pygame.K_a:
        s.sendto(b'a'.address)
      if event.type == pygame.K_s:
        s.sendto(b's',address)
      if event.type == pygame.K_q:
        s.sendto(b'q',address)
      if event.type == pygame.K_w:
        s.sendto(b'w',address)
```

WiFi UDP

- The other change is replacing **sendall()** with **sendto()**.
- The **sendto()** function requires a second parameter which is the address where the packet is being sent.
- Upload the updated arduino code, connect the WiFi to the robot and then run the python program. It should behave the same way.
- This shows two ways networking can be used.

```
import socket
import pygame

s = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
address = ('192.168.4.1',80)
pygame.init()
scn = pygame.display.set_mode((800,400))

while 1:
  for event in pygame.event.get():
    if event.type == pygame.QUIT:
      s.close()
      pygame.quit()
      quit()
    if event.type == pygame.KEYDOWN:
      if even.key == pygame.K_a:
        s.sendto(b'a'.address)
      if event.type == pygame.K_s:
        s.sendto(b's',address)
      if event.type == pygame.K_q:
        s.sendto(b'q',address)
      if event.type == pygame.K_w:
        s.sendto(b'w',address)
```

GUI WiFi

- In this section a Graphical User Interface will be created to control the LEDs. The window will be 800 x 400 pixels.
- The position of the buttons are shown.
- The red button starts at pixel location (600,100) and is 100 pixels wide and tall.
- The green button starts at location (600,250) and is also 100 pixels wide and tall.
- The values shown will be used in the python program.

GUI WiFi

- Sticking with the last arduino program, the python program will be updated to provide two buttons in the window to turn an LED on and off.
- The python program will be changed to provide the GUI interface.
- Two rectangles will be created and they will be the buttons.

```
import socket
import pygame

s = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
address = ('192.168.4.1',80)
pygame.init()
scn = pygame.display.set_mode((800,400))

while 1:
  for event in pygame.event.get():
    if event.type == pygame.QUIT:
      s.close()
      pygame.quit()
      quit()
    if event.type == pygame.MOUSEBUTTONDOWN:
```

GUI WiFi

- The python program will be modified to provide the buttons to control both LEDs.
- The changes are highlighted in bold type.
- Two variables, button1 and button2, are used to keep track of the state of the button. The buttons will toggle on and off.

```
import socket
import pygame

button1 = 0
button2 = 0

s = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
address = ('192.168.4.1',80)
pygame.init()
scn = pygame.display.set_mode((800,400))
font = pygame.font.SysFont("Arial",20)
b1 = font.render("RED",True,(0,0,0))
b2 = font.render("GREEN",True,(0,0,0))

while 1:
  for event in pygame.event.get():
    if event.type == pygame.QUIT:
      s.close()
      pygame.quit()
      quit()
    if event.type == pygame.MOUSEBUTTONDOWN:
```

GUI WiFi

- A font is generated for the window. The Arial font will be used and set to size 20 pixels.
- Below, the text is rendered and stored in variables b1 and b2. Pygame requires the text to be rendered before it is placed in the window. The text colors are set to black.

```
import socket
import pygame

button1 = 0
button2 = 0

s = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
address = ('192.168.4.1',80)
pygame.init()
scn = pygame.display.set_mode((800,400))
font = pygame.font.SysFont("Arial",20)
b1 = font.render("RED",True,(0,0,0))
b2 = font.render("GREEN",True,(0,0,0))

while 1:
  for event in pygame.event.get():
    if event.type == pygame.QUIT:
      s.close()
      pygame.quit()
      quit()
    if event.type == pygame.MOUSEBUTTONDOWN:
```

GUI WiFi

- In the while loop, the KEYDOWN event has been replaced with MOUSEBUTTONDOWN event. This checks if the mount button was pressed.
- If the mouse button was pressed, the coordinates of the mouse are captured into variables mX and mY. A function can return more than 1 result.

```
while 1:
  for event in pygame.event.get():
    if event.type == pygame.QUIT:
      s.close()
      pygame.quit()
      quit()
    if event.type == pygame.MOUSEBUTTONDOWN:
      mX,mY = pygame.mouse.get_pos()
      if (mX > 600) and (mX < 700):
        if(mY > 100) and (mY < 200):
          button1 = 1 - button1
        if(mY > 250) and (mY < 350):
          button2 = 1 - button2
```

GUI WiFi

- Next, the position of the mouse is compared with the location of the buttons. Since the mouse buttons are aligned vertically, the X direction is checked. If the mouse is between the left and right side of the button, the program then checks which button the mouse is within.
- The button state is then updated. A little math trick is done here. Since the state is either 1 or 0, you can subtract the button state from 1 and it will toggle from 1 to 0.

```
while 1:
  for event in pygame.event.get():
    if event.type == pygame.QUIT:
      s.close()
      pygame.quit()
      quit()
    if event.type == pygame.MOUSEBUTTONDOWN:
      mX,mY = pygame.mouse.get_pos()
      if (mX > 600) and (mX < 700):
        if(mY > 100) and (mY < 200):
          button1 = 1 - button1
        if(mY > 250) and (mY < 350):
          button2 = 1 - button2
```

GUI WiFi

- After checking the input and updating the button state, the window is to be drawn.
- First the window is cleared to all black.
- Next, the button state is checked. If 0, the button color is set to dark gray, if 1, the button is set to the LED color. Also, the command is sent to the robot.

```
scn.fill((0,0,0))
 if button1 == 0:
     b1_color = (100,100,100)
     s.write(b's')
 else:
     b1_color = (200,0,0)
     s.write(b'a')
 if button2 == 0:
     b2_color = (100,100,100)
     s.write(b'w')
 else:
     b2_color = (0,200,0)
     s.write(b'q')
 pygame.draw.rect(scn,b1_color,(600,100,100,100))
 pygame.draw.rect(scn,b2_color,(600,250,100,100))
 scn.blit(b1,(620,150))
 scn.blit(b2,(610,300))
 pygame.display.flip()
```

GUI WiFi

- At the end, the buttons are drawn.
- The two rectangle buttons are drawn using the specified colors.
- Next, the text is drawn on top of the rectangle buttons.
- Lastly, the display is updated.

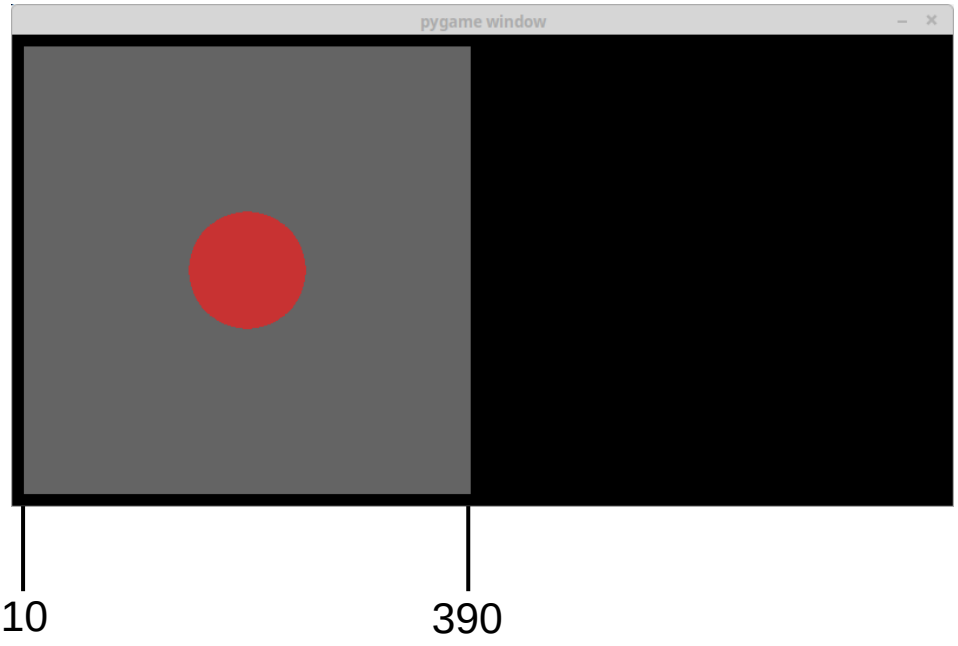
```
scn.fill((0,0,0))
 if button1 == 0:
     b1_color = (100,100,100)
     s.write(b's')
 else:
     b1_color = (200,0,0)
     s.write(b'a')
 if button2 == 0:
     b2_color = (100,100,100)
     s.write(b'w')
 else:
     b2_color = (0,200,0)
     s.write(b'q')
 pygame.draw.rect(scn,b1_color,(600,100,100,100))
 pygame.draw.rect(scn,b2_color,(600,250,100,100))
 scn.blit(b1,(620,150))
 scn.blit(b2,(610,300))
 pygame.display.flip()
```

GUI WiFi

- Try the python program. Don’t forget to connect the computer WiFi to the robot access point.

GUI Driving

- In this section, you will learn how to control the motors through the GUI. A joystick will be created. We will start with a new Arduino program to accept the joystick position and calculate the direction and speed of the motors.
- The joystick uses a rectangle for the background to show the boundaries of the joystick.
- The joystick knob is a red circle.
- The X and Y positions are the same values.

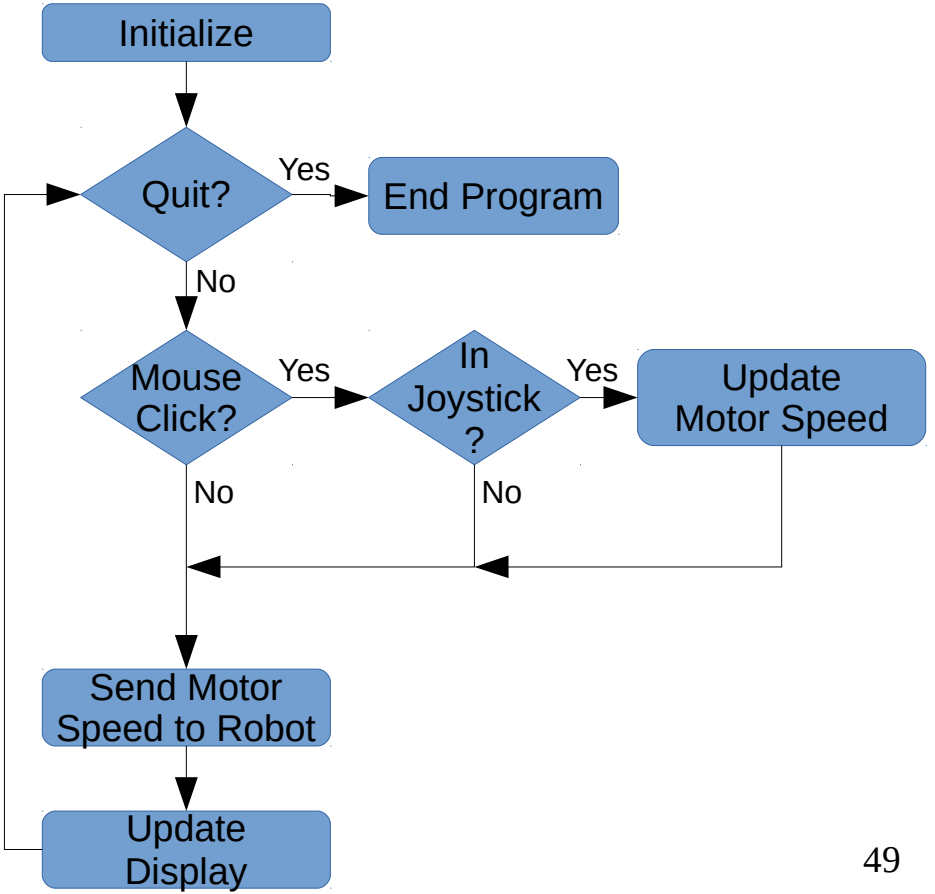


GUI Driving

- The python GUI program will take the mouse position within the joystick area and send it to the robot to set the speed in text format.
- The format will be simple as shown below

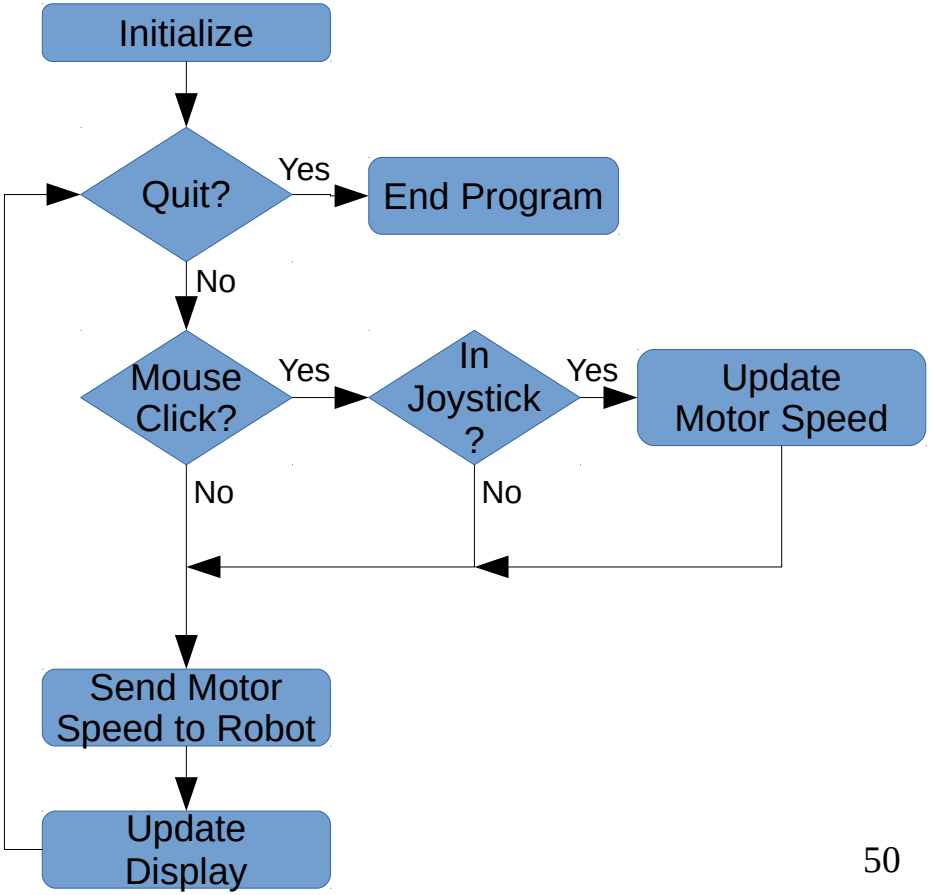
  200 200

- The first number is the X position of the joystick and the second number is the Y position.



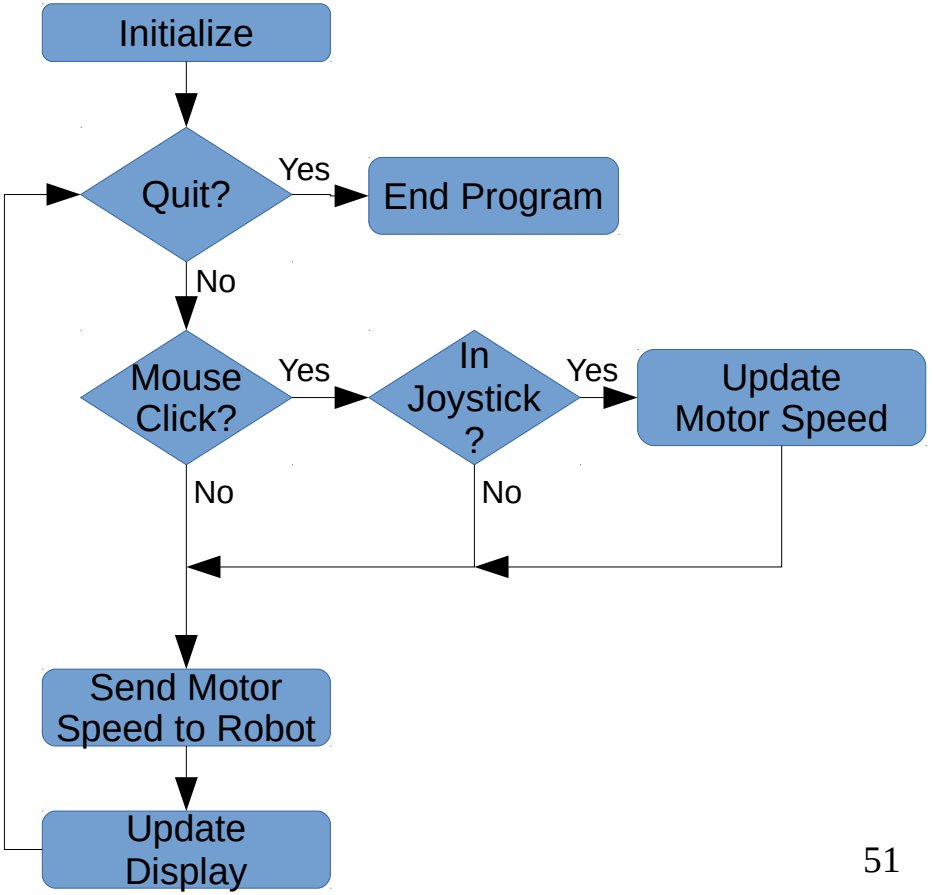
GUI Driving

- The flow chart to the right shows how the python program is to work.
- First, the program initializes the display window and network connection.
- Then the program checks for inputs.
  - First is the check if the program is to quick.
  - Next, the mouse button is checked.



GUI Driving

- If the mouse button is pressed, the next step is to check if the mouse is within the joystick range.
- If the mouse is within the joystick range, then the speed control of the robot is updated.
- Lastly, the speed control is sent to the robot and the GUI display is updated.



GUI Driving

- First, let's make the GUI and test it.
- Afterwards, the network will be added using UDP.
- The first part is setting up pygame, creating the window and checking for input.
- Variables **jx** and **jy** are created and will be used for the joystick knob position. It is set to the center of the joystick.

```
import pygame
import socket

pygame.init()

screen = pygame.display.set_mode((800,400))
jx = 200
jy = 200

while 1:
  for event in pygame.event.get():
    if event.type == pygame.QUIT:
      s.close()
      pygame.quit()
      quit()
```

GUI Driving

- The second part will draw the joystick with the knob in the center.
- Run the program and verify the joystick is rendered.

```
import pygame
import socket

pygame.init()
screen = pygame.display.set_mode((800,400))

while 1:
  for event in pygame.event.get():
    if event.type == pygame.QUIT:
      s.close()
      pygame.quit()
      quit()

  scn.fill((0,0,0))
  pygame.draw.rect((100,100,100),(10,10,380,380),4)
  pygame.draw.circle((200,0,0),(jx,jy),50)
  pygame.display.update()
```

GUI Driving

- Next, the mouse will control the joystick. Insert the code in bold where it is shown.
- The first bold line checks if the left mouse button is pressed. This can be used on touch screens too.
- If the mouse button is pressed, the second line gets the position of the mouse.

```
import pygame
import socket

pygame.init()
screen = pygame.display.set_mode((800,400))

while 1:
  for event in pygame.event.get():
    if event.type == pygame.QUIT:
      s.close()
      pygame.quit()
      quit()
  if pygame.mouse.get_pressed()[0] == True:
    mX,mY = pygame.mouse.get_pos()
    if (mX >10) and (mX < 390):
      if (mY >10) and (mY <390):
        jx = mX
        jy = mY

  scn.fill((0,0,0))
  pygame.draw.rect((100,100,100),(10,10,380,380),4)
  pygame.draw.circle((200,0,0),(jx,jy),50)
  pygame.display.update()
```

GUI Driving

- If the mouse is within the bounds of the joystick, the position of the joystick knob is updated to the mouse position.
- Try the program again and see if the knob can be moved around.

```
import pygame
import socket

pygame.init()
screen = pygame.display.set_mode((800,400))

while 1:
  for event in pygame.event.get():
    if event.type == pygame.QUIT:
      s.close()
      pygame.quit()
      quit()
  if pygame.mouse.get_pressed()[0] == True:
    mX,mY = pygame.mouse.get_pos()
    if (mX >10) and (mX < 390):
      if (mY >10) and (mY <390):
        jx = mX
        jy = mY

  scn.fill((0,0,0))
  pygame.draw.rect((100,100,100),(10,10,380,380),4)
  pygame.draw.circle((200,0,0),(jx,jy),50)
  pygame.display.update()
```

GUI Driving

- Go back to the editor and add the bold text.
- If the mouse button is not pressed, the else statement is executed. The joystick knob is put back to the center when the mouse button is not pressed.

```
import pygame
import socket

pygame.init()
screen = pygame.display.set_mode((800,400))

while 1:
  for event in pygame.event.get():
    if event.type == pygame.QUIT:
      s.close()
      pygame.quit()
      quit()
  if pygame.mouse.get_pressed()[0] == True:
    mX,mY = pygame.mouse.get_pos()
    if (mX >10) and (mX < 390):
      if (mY >10) and (mY <390):
        jx = mX
        jy = mY
  else:
    jx = 200
    jy = 200

  scn.fill((0,0,0))
```

GUI Driving

- Moving to the Arduino software, a program needs to be written to accept commands from the GUI program.
- The GUI program will send two values, **jx** an **jy**. The robot needs to convert those values to speed and direction control of the wheels.
- The code to the right is the setup for the robot. UDP packets will be used.
- The digital pins for the motors are set to outputs.

```
#include <ESP8266WiFi.h>
#include <WiFiUdp.h>

WiFiUDP udp;

char cmd[256];
int x,y;

void setup()
{
  Serial.begin(115200);
  pinMode(13,OUTPUT);
  pinMode(14,OUTPUT);
  pinMode(15,OUTPUT);
  pinMode(16,OUTPUT);
  WiFi.mode(WIFI_AP);
  WiFi.softAP("nameofrobot");
  udp.begin(80);
}
```

GUI Driving

- The beginning of the **loop()** function is similar to the UDP program for the LED control.
- **memset()** function sets the contents of a variable array to all zeros.
- After **udp.read()**, the packet read is converted from text into two values and stored in variables x and y.
- Then the variables are mapped from the position in the GUI to a range of -1023 to 1023 which is the maximum duty cycle.

```
void loop() {
  int ps = udp.parsePacket();
  if(ps) {
    memset(cmd,0,256);
    udp.read(cmd,256);
    sscanf(cmd,"%d %d",&x,&y);
    x = map(left,10,390,-1023,1023);
    y = map(right,10,390,-1023,1023);
    int left = x + y;
     int right = x - y;
     left = constrain(left,-1023,1023);
    right = constrain(right,-1023,1023);
    if(left < 0) {
      analogWrite(13,-left);
      analogWrite(14,0);
    } else {
      analogWrite(13,0);
      analogWrite(14,left);
    }
```

GUI Driving

- The beginning of the **loop()** function is similar to the UDP program for the LED control.
- **memset()** function sets the contents of a variable array to all zeros. The first parameter is the variable **cmd**, the second parameter is the value to set the array. The third parameter specifies the size of the array.
- The array needs to be cleared before new data is inserted otherwise old data can still be included.

```
void loop() {
  int ps = udp.parsePacket();
  if(ps) {
    memset(cmd,0,256);
    udp.read(cmd,256);
    sscanf(cmd,"%d %d",&x,&y);
    x = map(left,10,390,-1023,1023);
    y = map(right,10,390,-1023,1023);
    int left = x + y;
    int right = x - y;
    left = constrain(left,-1023,1023);
    right = constrain(right,-1023,1023);
    if(left < 0) {
      analogWrite(13,-left);
      analogWrite(14,0);
    } else {
      analogWrite(13,0);
      analogWrite(14,left);
    }
```

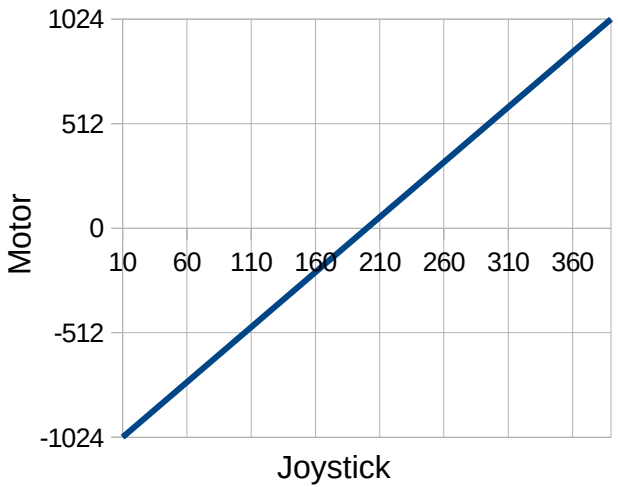
GUI Driving

- The **udp.read()** function fills the array with the packet data received.
- The **scanf()** function converts the text to two numbers and puts them in variables **x** and **y**.
- The first parameter is the array that holds the text. The second parameter specifies the format. %d indicates integer type number. The third and fourth parameters specify the variables to put the numbers. Notice the &. It is needed to reference the variable locations.

```
void loop() {
  int ps = udp.parsePacket();
  if(ps) {
    udp.read(cmd,256);
    sscanf(cmd,"%d %d",&x,&y);
    x = map(x,10,390,-1023,1023);
    y = map(y,10,390,-1023,1023);
    int left = x + y;
    int right = x - y;
    left = constrain(left,-1023,1023);
    right = constrain(right,-1023,1023);
    if(left < 0) {
      analogWrite(13,-left);
      analogWrite(14,0);
    } else {
      analogWrite(13,0);
      analogWrite(14,left);
    }
```

GUI Driving

- The **map()** function is used to convert the range of values to another range.
- The range of 10 to 390 will be stretched to -1023 to 1023.




```
void loop() {
  int ps = udp.parsePacket();
  if(ps) {
    udp.read(cmd,256);
    sscanf(cmd,"%d %d",&x,&y);
    x = map(x,10,390,-1023,1023);
    y = map(y,10,390,-1023,1023);
    int left = x + y;
    int right = x - y;
    left = constrain(left,-1023,1023);
    right = constrain(right,-1023,1023);
    if(left < 0) {
      analogWrite(13,-left);
      analogWrite(14,0);
    } else {
      analogWrite(13,0);
      analogWrite(14,left);
    }
```

GUI Driving

- The output of the **map()** function is used to control the speed of the motors and the direction. A negative number indicates reverse.
- After converting to a speed control and direction, the joystick position is mixed with the two equation to calculate the speed of each motor.
- The calculations can have results exceeding 1023 so the constrain() functions keeps the results from exceeding 1032 or -1023.

```
void loop() {
  int ps = udp.parsePacket();
  if(ps) {
    udp.read(cmd,256);
    sscanf(cmd,"%d %d",&x,&y);
    x = map(x,10,390,-1023,1023);
    y = map(y,10,390,-1023,1023);
    int left = x + y;
    int right = x - y;
    left = constrain(left,-1023,1023);
    right = constrain(right,-1023,1023);
    if(left < 0) {
      analogWrite(13,-left);
      analogWrite(14,0);
    } else {
      analogWrite(13,0);
      analogWrite(14,left);
    }
```

GUI Driving

- Next, the left motor speed is set. If the speed is negative, the motor is set to operate in reverse. The speed is negated since **analogWrite()** can only use positive numbers.
- If the speed is positive, the motors are operated in the forward direction.

```
void loop() {
  int ps = udp.parsePacket();
  if(ps) {
    udp.read(cmd,256);
    sscanf(cmd,"%d %d",&x,&y);
    x = map(x,10,390,-1023,1023);
    y = map(y,10,390,-1023,1023);
    int left = x + y;
     int right = x - y;
     left = constrain(left,-1023,1023);
    right = constrain(right,-1023,1023);
    if(left < 0) {
      analogWrite(13,-left);
      analogWrite(14,0);
    } else {
      analogWrite(13,0);
      analogWrite(14,left);
    }
```

GUI Driving

- The right motor is controlled the same way.

```
    if(right < 0) {
      analogWrite(16,-right);
      analogWrite(15,0);
    } else {
      analogWrite(16,0);
      analogWrite(15,right);
    }
  }
}
```

Updating GUI Program

- The joystick GUI program needs to be updated to include sending UDP packets to control the robot.
- Near the top of the program, add the bold text. This configures the socket connection.
- The second bold line is the IP address of the robot and the port number.

```
import pygame
import socket

pygame.init()
screen = pygame.display.set_mode((800,400))
s = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
robot = ('192.168.4.1',80)
while 1:
  for event in pygame.event.get():
    if event.type == pygame.QUIT:
      s.close()
      pygame.quit()
      quit()
  if pygame.mouse.get_pressed()[0] == True:
    mX,mY = pygame.mouse.get_pos()
    if (mX >10) and (mX < 390):
      if (mY >10) and (mY <390):
        jx = mX
        jy = mY
  else:
    jx = 200
    jy = 200

  scn.fill((0,0,0))
```

Updating GUI Program

- The bold lines inserted after checking the inputs build a command using the values of jx and jy. The first bold line, variable cmd is created as a string to include jx and jy with a space between the two.
- The second line sends the command to the robot. the string cmd is encoded into a simple character array and sent to the robot.
- The last line slows down the program. The robot cannot receive too many packets too fast.

```
if pygame.mouse.get_pressed()[0] == True:
  mX,mY = pygame.mouse.get_pos()
  if (mX >10) and (mX < 390):
    if (mY >10) and (mY <390):
      jx = mX
      jy = mY
else:
  jx = 200
  jy = 200

cmd = str(jx) + " " + str(jy)
s.sendto(cmd.encode(),robot)
scn.fill((0,0,0))
pygame.draw.rect((100,100,100),(10,10,380,380),4)
pygame.draw.circle((200,0,0),(jx,jy),50)
pygame.display.update()
pygame.time.wait(25)
```

GUI Driver

- Load the Arduino program.
- Connect the laptop to the robot.
- Run the python program.
- Click anywhere inside the joystick and the joystick knob will instantly move.
- Holding the mouse button, the joystick knob will follow.